Technische Universiteit Delft
Faculteit Elektrotechniek, Wiskunde en Informatica
Delft Institute of Applied Mathematics

Resource-constrained Machine Scheduling

Verslag ten behoeve van het
Delft Institute of Applied Mathematics
als onderdeel ter verkrijging

van de graad van

BACHELOR OF SCIENCE
in
TECHNISCHE WISKUNDE

door

NOORTJE BONENKAMP

Delft, Nederland
April, 2017

Summary

Background Scheduling is a form of decision-making that plays a very important role in manufacturing industries. It is the method by which work gets assigned to resources that compute it. In this thesis we want to assign jobs (cycles of a lithography process) to different machines in such a way that the machines are running for as little time as possible. The difficulty lies in the fact that some jobs require the same resource, and thus cannot be processed simultaneously.

Results It appears that the following scheduling problems are NP-hard:

- $P2\|C_{max}$ and $R2\|C_{max}$
- $P3|res \cdot 11, pj = 1|C_{max}$ and $R3|res \cdot 11, pj = 1|C_{max}$
- $P3|res \cdot 11, p_j = 1|\sum C_j$ and $R3|res \cdot 11, pj = 1|\sum C_j$

We try to approximate an optimal solution for the scheduling problem $Rm|res \cdot 11|\sum C_j$ by calculating a ratio with a lower bound, where we ignore resource constraints, and an upper bound, that is found by a Greedy algorithm (w.r.t. resource constraints).

Conclusions It appears that Greedy is not a very good approximation algorithm for this problem, since the ratio in which the optimal Total Completion Time should lie is about 8,33% of the average calculated Total Completion Time.

Contents

Chapter 1

Introduction

Machine scheduling

Scheduling is a form of decision-making that plays a very important role in manufacturing industries. Nowadays, companies have to meet shipping dates committed to customers and they have to schedule activities in such a way that resources available are used in an efficient manner. Scheduling deals with the allocation of resources to tasks over given time periods. The goal is to optimize one or more objectives. In this thesis, the resources are ASML machines on which different jobs get processed: these are cycles of a lithographic manufacturing process.

Lithography

NXP Semiconductors B.V. is a Dutch global semiconductor manufacturer. NXP manufactures a wide range of integrated circuits (IC) for different applications. Contact-less payments, connected cars, wearable devices are a few applications of the technologies in which these ICs can be used.

An integrated circuit is a set of electronic circuits on one small flat piece ("chip") of semiconductor material, interconnected by a network of metal wiring in different layers. The best-known semiconductor material is silicon. The silicon gets cut into thin slices (wafers) that undergo different chemical treatments (see figure 1).

One of the techniques that can process silicon is chemical etching, in a process called *lithography* (which is ancient Greek for *writing on stone*). The lithographic manufacturing process of an IC consists of approximately 30 cycles, where each cycle represents a layer. Most of the layers are used for the interconnecting metal wiring that have to connect the functional parts, defined in the first layers, without interfering with any of the wiring in the other layers. The composition of the first layers of the device requires an extreme level of accuracy in the positioning of the

layers.

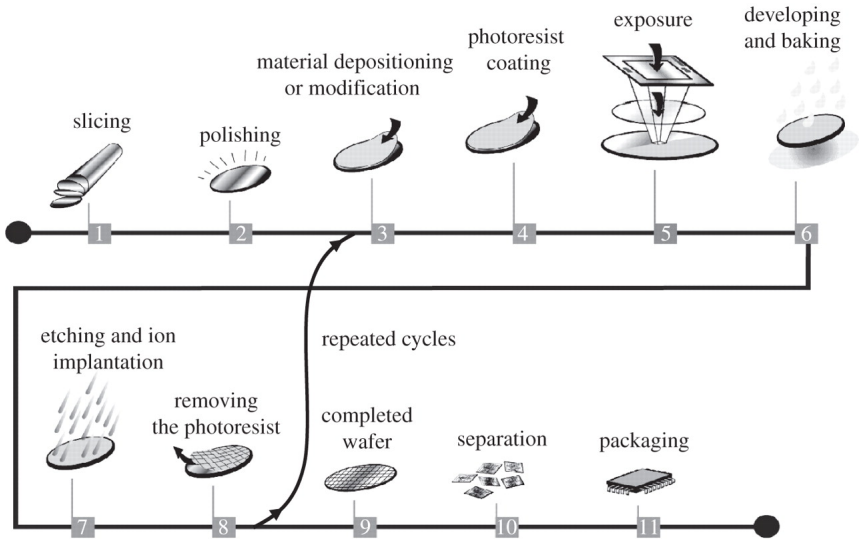

Figure 1.1: IC production

These steps can be executed by a so-called wafer scanner (see figure 1.3). The scanner determines the details and position of the pattern structures. In figure 1.2, this scanning principle is explained. The original pattern of a specific layer of the IC is defined in a chromium layer on a *reticle*, a six-inch square that is made unique for every layer.

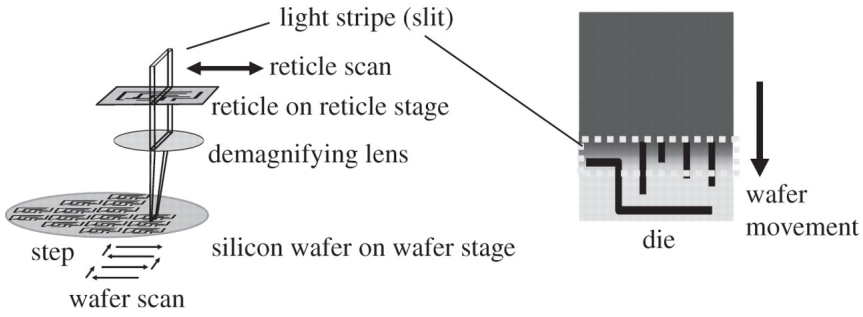

Figure 1.2: Scanning principle of a wafer scanner. Light creates a pattern on the reticle. The pattern is imaged on the wafer by a lens. A movement of the reticle and wafer together results in an image of the pattern on the wafer.

The reticles are the primary input of the product engineers to chip manufacturing. Making one reticle costs about 5000$- 10000$, and that does not include making the lay-out!

The research

We want to create schedules for the cycles of the lithographic manufacturing process in such a way that the machines are running for as little time as possible. It is interesting to make these schedules as effective as possible, because the whole process and the machines are both very expensive. There are certain jobs that share the same reticle. Since the reticles are too expensive to make more for these jobs, we want to create schedules where jobs that share the same reticle (from now on referred to as 'resource') do not get processed simultaneously.

Firstly, we will go more into different scheduling problems without resources in depth. Which of these problems can be solved by an algorithm in polynomial time? After that, we will consider different objectives for scheduling with resources and ask ourselves the same question. In the last chapter we will try to approximate solutions for the problem of the lithographic manufacturing process, based on different instances with the data from NXP.

Figure 1.3: Wafer scanner from ASML with the main modules for exposing a wafer. The illuminator (1) directs the exposure light to the reticle on the reticle stage (2). The pattern of the reticle, defined in a chromium layer on a quartz plate, is imaged by the projection lens (3) onto the wafer (4) located on the wafer stage (5). The wafer handler (6) exchanges the wafer after each exposure while the reticle handler (7) exchanges the reticle for each different layer that is required for a functioning IC. *(Courtesy of ASML.)*

Chapter 2

Preliminaries

2.1 Notation and framework

Before we can consider scheduling problems, it is necessary to introduce some notation. What do they look like?

Every scheduling problem is described by a triplet $\alpha|\beta\ |\gamma$. Parameter α specifies the machine environment, parameter β the details of processing characteristics and constraints and parameter γ specifies the function that has to be minimized.

The number of jobs will be denoted by n and the number of machines by m. We assume that both n and m are both always finite. Furthermore, the subscript j refers to a job and subscript i refers to a machine. So (i, j) means the operation of job j on machine i.

The following data can apply to j:

notation	**meaning**
p_{ij}	processing time of job j on machine i
w_j	weight of job j (how important this job is, compared to other jobs in the system)

Parameter α, the machine environment, always consists of one variable. It is possible to have the following environments:

- **Single machine** (1) This is the simplest possibility: there is only one machine on which all the jobs are processed.

- **Identical machines in parallel** (Pm) There are m identical parallel machines, i.e. $p_j = p_{ij},\ \forall i \in 1, \ldots m.$

- **Machines in parallel with different speeds** (Qm) In this case ("uniform machines"), we have m machines in parallel with different speeds.

- **Unrelated machines in parallel** (Rm) We have m totally unrelated machines in parallel; the processing time of job i on machine j is totally random. This is the machine environment for the problem that is discussed in the introduction.

For processing characteristics and constraints, β in the triplet, it is possible to have more variables at the same time. Variables that can occur in the β field are (amongst others):

- **Preemptions** (*prmp*) In this case, it is not necessary to keep a job on the machine until it is processed. Preemptions indicate that a job can be removed when it is not finished yet, and replaced by another job. When the removed job is put back, the time that it already spent before it got removed is not lost. When preemptions are allowed, *prmp* occurs in the β field. In our case this does not occur in the field, which means that preemptions are not allowed: jobs cannot be interrupted on our machines.

- **Resources** It is possible that the jobs require specific resources to be processed on the machines. This is also the case for our problem; the reticles are these specific resources. Assume there are l resources: $R_1, \cdots R_l$. Every resource R_h has a size s_h; this is the total amount of R_h on a certain moment. For every resource R_h and job J_j a certain amount of R_h is required: *requirement*, r_{hj}. A schedule is feasible when every t satisfies the following resource constraints (for the index set S_t of jobs that get performed on t):

$$\sum_{j \in S_t} r_{hj} \leq s_h$$

In the β field we describe the presence of resources as follows:

$$res\lambda\sigma\rho$$

- When λ is a positive integer, the amount of resources, l, is constant and equal to λ. When $\lambda = \cdot$, we find that l is a part of the input.
- When σ is a positive integer, we find that the size of all the resources, s_h, is constant and equal to σ. When $\sigma = \cdot$, all the s_h are a part of the input.
- When ρ is a positive integer, all the resource requirements, $r_{hj}(r_{hij})$ have a constant upper limit, equal to ρ. When $\rho = \cdot$, nothing is known about such an upper limit.

The object that has to be minimized is always a function of the completion times of the jobs. The completion time of the operation on job j on machine i is denoted by C_{ij}. The time job j leaves the system is denoted by C_j; this is equal to the completion time on the last machine that operates on job j.

Objective functions that can be minimized are:

- the **Total weighted completion time** ($\sum w_j C_j$) The total weighted completion (TCT) time is the sum of all the weighted completion times of the jobs. The weight w_j of job j is basically an importance factor. A user can influence the completion time of his job by assigning a higher weight to it. Minimizing the TCT is related to the inventory costs.

- the **Makespan** ($C_{\max}$) This is equal to the completion time of the last job ($= \max_{j \in J} C_j$). Minimizing the makespan is related to the overall use of a multiprocesser, and individual job weights are ignored.

We want to minimize both the makespan and the TCT, since they are both are interesting for the manufacturer. We will do this by making *schedules*. A schedule refers to an allocation of jobs within a more complicated setting of machines. Our goal is finding a feasible schedule which minimizes the objective function.

It is possible that we cannot find an algorithm to optimize (one of) these objective functions. In that case, there are ways we can *approximate* an optimal solution. We will explain more about this in the next section.

2.2 Complexity

Sometimes algorithms for certain problems can be used for other problems as well. A *reduction* is an algorithm for transforming one problem into another problem and can be used to show that the second problem is *at least* as difficult as the first. It can even be possible to create a whole chain of reductions.

Reductions can be used to determine complexity classes. We consider two different complexity classes: P and NP. P is a class of decision problems that can be *solved* by a polynomial algorithm.[1] NP is a class of decision problems whose solutions can be *verified* within polynomial time.[2] These are called the NP-complete problems. Furthermore, a problem is called NP-hard when an NP-complete problem can be reduced to it in polynomial time. So very informal, we say that NP-hard problems are *at least as hard as the NP-complete problems* (but they do not have to be in NP). Not all is lost when a problem is NP-hard: by using so-called *approximation algorithms* we can approximate solutions to problems.

This is of course a very interesting topic, but why should it bother us in Scheduling? Imagine having a complicated scheduling problem. Once you can prove that it is NP-hard, you can devote your efforts to finding good approximation algorithms, which could save you a lot of time (and headache).

Apparently, reductions can be made in some of our cases. For instance, $1\|\sum C_j$ is a special case of $1\|\sum w_jC_j$, which means that an algorithm for $1\|\sum w_jC_j$ can also be used for $1\|\sum C_j$. Reductions makes it interesting to look at the impact of changing a single element in an objective function on the complexity of the function. When we examine the presence of resources on the complexity of objective function later in this thesis, we will have a closer look at this.

[1] An algorithm is called polynomial if its output can be computed in at most $O(p(|x|)$ step where p is a polynomial and $|x|$ is the length of the input x.

[2] P $\subseteq$ NP, but whether or not P=NP is a major unsolved problem in computer science and one of the seven Millennium Prize Problems.

Chapter 3

Scheduling without resources

Before we get to the more complicated scheduling problems in this thesis, we first take a look at scheduling problems *without* resource constraints. It is interesting to address them to find out just how complicated scheduling problems get after adding resource constraints. We will consider three different machine environments in this chapter: single machine models, (identical) machines in parallel and unrelated machines in parallel.

3.1 Single Machine Models

The single machine environment models often have properties that models with machines in parallel do not have. Results for problems in the single machine environment are usually quite easy to find and they can often provide insights into the more complicated machine environments. It is therefore that single machine models are very important in scheduling.

Minimizing the Makespan

There is not much to minimizing the makespan in case of a single machine environment: it is clear that the completion of the last job occurs at the makespan $C_{max} = \sum p_j$, which is independent of the schedule.

Minimizing the Total Weighted Completion Time

Let us discuss the following problem: $1||\sum w_j C_j$.

The following Theorem about the *Weighted Shortest Processing Time first (WSPT)* rule makes it easy to solve this problem. This rule states that the jobs should be ordered on the machine

in time such that $\frac{w_j}{p_j}$ decreases.

Theorem 3.1.1 (Pinedo)**.** *The WSPT rule is optimal for* $1||\sum w_j C_j$.

Proof. We will prove this by contradiction. Assume that we have an optimal schedule S, that is not ordered by its smallest $\frac{w_j}{p_j}$ first. Then we find that there are at least two adjacent jobs, say job j followed by job k, such that

$$\frac{w_j}{p_j} < \frac{w_k}{p_k}$$

Assume that the processing time of job j starts at time t. Now, switch j and k. Call the new schedule S'. In the original schedule S, we have that j starts its processing time on t, but in S' we have that job k starts its processing time on t and is followed by j. All the other jobs in S' remain in their original position from S. That means that the total weighted completion time of the jobs that are processed before job j en k do not change due to this interchange. This is also the case for the jobs that are processed after job j and k. The difference in the values of the objectives under schedules S and S' is only due to job j and k. Figure 3.1 clarifies this. In schedule S, we find that the total weighted completion time of jobs j and k is equal to

$$(t + p_j)w_j + (t + p_j + p_k)w_k,$$

and in schedule S', we find that the total weighted completion time of jobs j and k is equal to

$$(t + p_k)w_k + (t + p_k + p_j)w_j.$$

If we would still have $\frac{w_j}{p_j} < \frac{w_k}{p_k}$, the sum of the two weighted completion times under S' is less than under S and S is not optimal: we find a contradiction. □

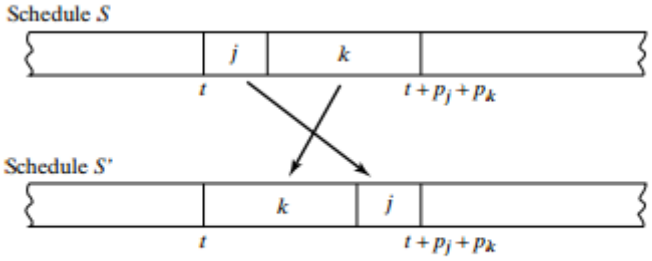

Figure 3.1: An interchange between jobs j and k. *(Source: Pinedo)*

The computation time to order the jobs according to the $WSPT$ rule is equal to $O(nlog(n))$ (as any other sorting). [9]

3.2 Machines in parallel

In this section we work with identical machines in parallel. Again, we will take a look at minimizing the makespan and the TCT.

Minimizing the Makespan

One of the most basic scheduling problems is

$$Pm\|C_{max}.$$

The variable m represents the number of identical machines. The input consists of n jobs with processing times $p_1, \dots p_n$. We want to assign the jobs to the m machines in such a way that the last job to be finished completes as early as possible (and this completion time is equal to the makespan).

We can say the following about $P2\|C_{max}$:

Theorem 3.2.1. *$P2\|C_{max}$ is NP hard*

Proof. We will show that $P2\|C_{max}$ is NP hard by reducing the NP-hard problem PARTITION.

PARTITION is the following problem:

Given positive integers $a_1, \dots a_t$ and

$$b = \frac{1}{2}\sum_{j=1}^{t} a_j$$

do there exist two disjoint subsets S_1 and S_2 such that

$$\sum_{j \in S_i} a_j = b$$

for $i = 1, 2$?

For the reduction of PARTITION into $P2\|C_{max}$, take

$$\begin{aligned} n &= t, \\ p_j &= a_j, \\ z = \frac{1}{2}\sum_{j=1}^{t} a_j &= b, \end{aligned}$$

where z is the optimum.

To confirm that this is a reduction, we need to show that the answer is "yes" for the instance of PARTITION if and only if the answer is "yes" for the instance of $P2\|C_{max}$. Let I' be the instance of PARTITION and I be the instance of $P2\|C_{max}$. Say that we have an I' where the answer is "yes": two disjoint subsets S_1 and S_2 exist. By the definition of S_i we find:

$$\sum_{j \in S_i} p_j = b$$

for $i = 1, 2$. Similarly, if we are given a schedule that solves $P2\|C_{max}$, we can construct the partition S_1, S_2 as well. □

It is possible to approximate an optimal schedule for the problem $Pm\|C_{max}$, by using the following rule:

De *Longest Processing Time first* rule

The LPT rule divides the jobs over the machines in the following way: On $t = 0$ the m jobs with the longest processing time get assigned to the m machines. As soon as one of the jobs is finished, the next job with the longest processing time (that is not yet assigned to a machine) will get assigned to that machine. So the idea is that all the short jobs are dividing the load over the machines in the end of the schedule.

Theorem 3.2.2 (Pinedo[9]). *For $Pm\ |C_{max}$ we have:*

$$\frac{C_{max}(LPT)}{C_{max}(OPT)} \leq \frac{4}{3} - \frac{1}{3m}$$

Minimizing the Total Completion Time

In Theorem 3.1.1 we proved that for a single machine the Shortest Processing Time first rule is optimal for the Total Completion Time (if we assume that $w_1 = w_2 = \cdots = w_n = 1$, the WSPT rule becomes the SPT rule, which is still optimal). We proved this by contradiction. But there is an alternative way of proving this (which is useful for us):

If we let $p_{(j)}$ imply the processing time of the job in the jth position, we find that the total completion time can be expressed as

$$\sum C_j = np_{(1)} + (n-1)p_{(2)} + \cdots + 2p_{(n-1)} + p_{(n)}$$

We find n coefficients n, $n-1$, ..., 1 that have to be assigned to n different processing times in such a way that the sum of the products is minimized. Obviously , for this we choose to assign the highest coefficient, n, to the smallest processing time, p_n, the second highest coefficient, $n-1$, to the second smallest processing time, $p_{(n-1)}$, and so it goes on. [4] So again we find that SPT is optimal.

The nice thing about proving the theorem with this method, is that we can now extend the proof to a parallel machine setting. We find the following:

Theorem 3.2.3 (Pinedo). *The SPT rule is optimal for $Pm\|\sum C_j$*

Proof. In this case, when we optimize the TCT, there are nm coefficients that the processing times can be assigned to: mn's, $m(n-1)$'s , ..., m ones. The processing times should be assigned in such a way that we minimize the sum of the products. Say that n/m is an integer. If it is not an integer, we are allowed to add 'dummy jobs', with a processing time equal to zero. The set of the m longest processing times have to be assigned to the m ones, the set of the m second longest processing times have to be assigned to the m twos, and so it goes on. By this method,

the m longest jobs will be processed on different machine. This goes according to an SPT schedule: the smallest job has to go first, on machine 1, the second smallest one on machine 2, etc. Also, the (m+1)th smallest job follows the smallest job on machine 1, the (m+2)th smallest job follows the second smallest on machine 2, and so it goes on. We find that the SPT schedule is an optimal schedule for this objective function. □

Remember, in the first chapter we have shown that the more general WSPT rule was optimal for the single machine environment (Theorem3.1.1). But this theorem can not be extended to parallel machines. This can be shown by a simple example.

Example 3.2.1.

Say we have the following problem (three jobs that have to be scheduled on two machines).

$jobs$	1	2	3
p_j	1	1	3
w_j	1	1	3

The WSPT Rule would make us schedule job 1 and 2 at $t = 0$ and job 3 at $t = 1$. We find that $\sum w_j C_j = 14$. But if we schedule job 3 at $t = 0$ on one of the machines and jobs 1 and 2 on the other machine, we find that $\sum w_j C_j = 12$. So this is an example where WSPT does not work.

However, WSPT is a good heuristic for parallel machines with an approximation factor [5]

$$\frac{\sum w_j C_j(WSPT)}{\sum w_j C_j(OPT)} < \frac{1}{2}(1 + \sqrt{2}).$$

3.3 Unrelated machines

Now that we have considered single machine environments and machines in parallel, it is time to focus on the machine environment that we introduced in the beginning of this thesis: unrelated machines. We can use some of the results that we found for parallel machines: for instance when we want to minimize the makespan.

Minimizing the Makespan

From Theorem 3.2.1, it follows that $Rm\|C_{max}$ (with a fixed number of unrelated machines) is NP-hard, since $Pm\|C_{max}$ reduces to $Rm\|C_{max}$. Lenstra, Shmoys and Tardos proposed a 2-approximation algorithm for $Rm\|C_{max}$. [6]

Minimizing the Total Completion Time

Consider the function $Rm\|\sum C_j$; machines in parallel that are completely unrelated.

Theorem 3.3.1. *$Rm\|\sum C_j$ is solvable within polynomial time.*

Proof. We will show this by formulating this problem as an Linear Problem with n jobs and nm positions. If a job j is scheduled on machine i and $k-1$ jobs follow on this machine, then job j contributes kp_{ij} to the objective value. (Since the 'weight' is job-dependent, we cannot simply sort the 'weights'.)

Let

$$x_{ikj} = \begin{cases} 1, & \text{when job } j \text{ is scheduled as } k\text{th until last job on machine } i \\ 0, & \text{otherwise} \end{cases}$$

We can formulate our problem as the following ILP.

min

$$\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{n} kp_{ij}x_{ikj}$$

s.t.

$$\begin{aligned} \sum_{i=1}^{m}\sum_{k=1}^{n} x_{ikj} &= 1, \qquad && j = 1,\dots,n \\ \sum_{j=1}^{n} x_{ikj} &\le 1, && i = 1,\dots m,\ \ k = 1,\dots,n \\ x_{ikj} &\in \{0,1\} && i = 1,\dots m,\ \ k = 1,\dots,n\ \ j = 1,\dots,n. \end{aligned}$$

Now every job is scheduled once and every position on the machines has one job at most. When job j is assigned to position k on machine i, the corresponding cost value is kp_{ij}.

Because of the total unimodularity property[1], the constraints do not have to require that the variables are either 0 or 1, thus we can replace $x_{ikj} \in \{0,1\}$ with $x_{ikj} \ge 0$. Since an LP can be solved in polynomial time, our problem can be solved in polynomial time. □

[1] A matrix has the total unimodularity property if the determinant of every square submatrix within the matrix has value -1, 0 or 1. It is not difficult to verify that this is the case with the matrix we find when solving $Rm\|\sum C_j$.

Chapter 4

Scheduling with resources

In the first chapter, we only considered objective functions where there were zero restrictions or constraints. In this chapter, we will go back to the problem stated in the introduction of the thesis, and have a look at the complexity of different objective functions under resource constraints. We assume that every operation on a job requires a unique resource. Every machine can process one job at the same time and every job can be processed on one machine at the same time. The goal is finding an optimal schedule, under these resource constraints.

4.1 Machines in parallel

Complexity

We consider the scheduling problem $Pm|res \cdot 11, p_j = 1|C_{max}$, machines in parallel with processing times that are all equal to 1, subject to resource constraints that state that every job requires a unique resource. We can say the following about $P3|res \cdot 11, p_j = 1|C_{max}$:

Theorem 4.1.1 (Blazewich & Lenstra)**.** *$P3|res \cdot 11, p_j = 1|C_{max}$ is NP-hard.*

Proof. We will show that the NP-complete problem PARTITION INTO TRIANGLES can be reduced to $P3|res \cdot 11, p_j = 1|C_{max}$.

PARTITION INTO TRIANGLES *is the following problem:*

Given: *A graph $G = (V, E)$, with $|V| = 3q$ for an integer q.*

Question: *Can the vertices of G be divided in q disjoint sets with each three vertices, $V_1, V_2, \ldots, V_q$, in such a way that every V_i contains the three vertices of a triangle in G ?*

For every graph G with vertices V and edges E we find that functions and constraints of the

type $res \cdot 11$ can be constructed as follows:
- for every vertex $j \in V$ introduce a job J_j;
- for every pair of vertices $j, k \notin E$, introduce a resource $R_{\{j,k\}}$ of size $s_{\{j,k\}} = 1$, with $r_{\{j,k\}j} = r_{\{j,k\}k} = 1$, and $r_{\{j,k\}i} = 0$ for $i \in E \backslash \{j, k\}$.

This means that two jobs can be processed simultaneously if and only if the corresponding vertices are adjacent.

For every instance of our problem we can construct an instance of
$P3|res \cdot 11, p_j = 1|C_{max}$ like illustrated above.

PARTITION INTO TRIANGLES has a solution if and only if there is a feasible schedule with $C_{max} \leq q$.

□

(A similar reduction can be done for $Q2|res \cdot 11, p_j = 1|C_{max}$: two machines in parallel with different speeds. We refer to appendix A for this reduction.)

If we now assume that there is *one resource per job*, we find that the reduction of PARTITION INTO TRIANGLES does not work anymore. This is because we initially introduced a resource for every *pair* or vertices. But if we assume one resource per job, every vertex (since we defined the vertices as jobs) can only adjoin one edge. It even appears that when we assume *one resource per job*, the following problem is solvable within polynomial time.

Theorem 4.1.2. *$Pn|res \cdot 11, p_j = 1$, one per job$|\sum C_j$ is solvable within polynomial time.*

Proof. We will prove this by showing that $Pn|res \cdot 11, p_j = 1$, *one per job*$|\sum C_j$ can be formulated as a min cost flow problem. We create a directed graph $G = (V, C)$.

V denotes a set of

- vertices corresponding to the jobs
- the resources and positions
- the machines and positions.

This set also contains two additional vertices: a source and a sink. We will call the source s and the sink t.

C is a set of arcs that connects

- 'dummy' vertex s to every job

- every job to possible positions of its corresponding resource
- all the positions of different resources to possible machine positions
- all the machine positions to 'dummy' vertex t.

The values of the arcs are as follows: All the arcs except the ones connecting machine positions with t have a value equal to 0. The arcs connecting machine position to t have a value that is equal to their position (because we assumed that $p_j = 1, \forall j \in J$). So the arc connecting M_{11} with t has a value of 1, the arc connecting M_{12} with t has a value of 2, etc. The arc connecting M_{21} with t has a value of 1 again, and so it goes on. See figure 2.1. The maximum flow on all arcs is equal to 1.

Minimizing the flow in this network is equal to minimizing the TCT: when a job gets processed on a machine on position k, its completion time adds up with $1 + 2 + 3 + \cdots + k$. This is in correspondence with the values on the arcs leading from $machine + position$ to vertex t.

Thus, $Pn|res \cdot 11, p_j = 1,$ *one per job*$|\sum C_j$ can be solved in polynomial time, since min cost flow can be solved in polynomial time. [7]

□

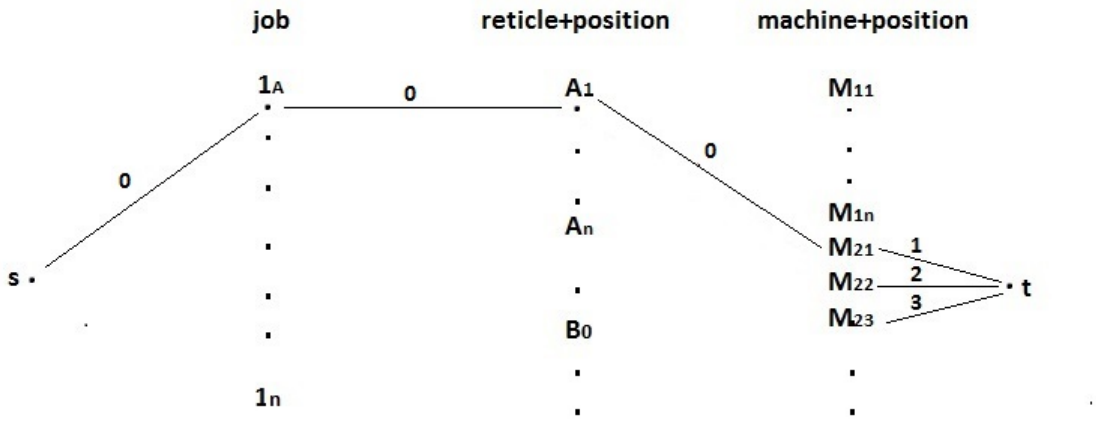

Figure 4.1: Graph for finding a schedule for $Pn|res \cdot 11, p_j = 1,$ *one per job*$|\sum C_j$

Now, what can we state about $Pn|res \cdot 11,$ *one per job*$|\sum C_j$? At the moment we think that this problem is also NP-hard. But this is only a presumption, since we have not been able to prove anything about its complexity.

4.2 Unrelated machines

In the previous section we proved that $P3|res \cdot 11, p_j = 1|C_{max}$ is NP-hard. Clearly, it follows that $R3|res \cdot 11, p_j = 1|C_{max}$ is also NP-hard.

When we do not want to minimize C_{max}, but $\sum C_j$, we use the fact that the machines are saturated in the transformation. The NP-hardness of Theorem 4.1.1 also applies to $\sum C_j$. For theorem 3.1 we find that $C_{max} \leq t$ if and only if $\sum C_j \leq \frac{3}{2}t(t+1)$. [1]

Corollary 4.2.1. *$P3|res \cdot 11, p_j = 1| \sum C_j$ is NP-hard.*

From Corollary 4.2.1 it follows that $R3|res \cdot 11, p_j = 1| \sum C_j$ is also NP-hard.

Like in the case of parallel machines, we cannot show that $Rm|res \cdot 11, p_j = 1$, *one per job*$| \sum C_j$ is NP-complete. Unfortunately we cannot solve this problem with the flow network of Theorem 4.1.2, because in this case the machines are completely unrelated. So the complexity of this problem remains an open question for now.

Chapter 5

Greedy approximations

After all these proofs, let us go back to the original problem as stated in the introduction: we have totally unrelated machines and jobs with different processing times, that all require a unique resource. How can we approximate an optimal solution for the objective functions $Rm|res \cdot 11|\sum C_j$ and $Rm|res \cdot 11|C_{max}$?

We will try to approximate a minimal TCT by using a so-called Greedy algorithm.

Greedy works (very general) as follows: in every iteration you choose the local optimum. For our problem this means that every iteration we calculate what the completion times of all the jobs are that we can schedule (by scanning all the jobs and all the machines). When a job shares a resource with other jobs, it is possible that in certain iterations a job has to 'wait' for its resource. The time it has to wait gets added to its completion time in that iteration. At the end of every iteration we schedule the job with the smallest completion time and delete that job from the list of remaining jobs.

We did this by implementing a code in `MATLAB` (Appendix C), which works as follows. Like we said, every iteration we consider which job has the smallest completion time on which machine (by also considering the time it possibly has to 'wait' for its resource to be available). We created a loop that scans the matrix (`"machinetijdenmatrix"`: the *machinetime* of different machines and the processing times of the jobs are added in this matrix) with the completion times for remaining jobs in that iteration. When a job gets scheduled on a machine, the job gets added to the schedule and deleted from the list of remaining jobs. The machine time of that machine gets updated with the processing time of this job.

We let 90 different instances run trough this code. Every one of these 90 instances represents 8h of historical production in the NXP production plant. The results can be found in column '**TCT with res. GREEDY**' in table B.1 in Appendix B.

We were wondering how many resource conflicts actually appear when we do not take the resource constraints into consideration. That is why we have also implemented a Greedy algorithm that calculated the TCT, but without considering these constraints. Besides ignoring resource

constraints, it works the same as the first code. These results can be found in column '**TCT without res. GREEDY**' in table B.1. We also added a column in which the number of resource conflicts for that iteration can be found ('**number of res. conflicts'**).

With these two algorithms we also calculated what the makespan would be after running these codes. We also added these values to the table: it is interesting to see the difference between the value of the makespan with and without resource constraints. For many problems it appears that the makespan values do not differ that much from each other, even though sometimes a lot of resource conflicts appear when we do not take these constraints in consideration.

Lastly, we added a column to the table with a lower bound for the TCT ('**TCT without res. LOWER BOUND**'). These values are also calculated with a `MATLAB` code, created by Teun Janssen, who wrote a code that optimized the TCT without considering resource constraints. This code used the result of Theorem 4.1.2.

Let us look at one of the instances for more clarity. In `instance_1.xls`, there are 358 jobs with 2073 eligible machine-job combinations for the 29 machines we have. The jobs use, all together, 301 different recourses.

When we find the Greedy solution for the objective function **without** considering the resources, we find the following schedule:

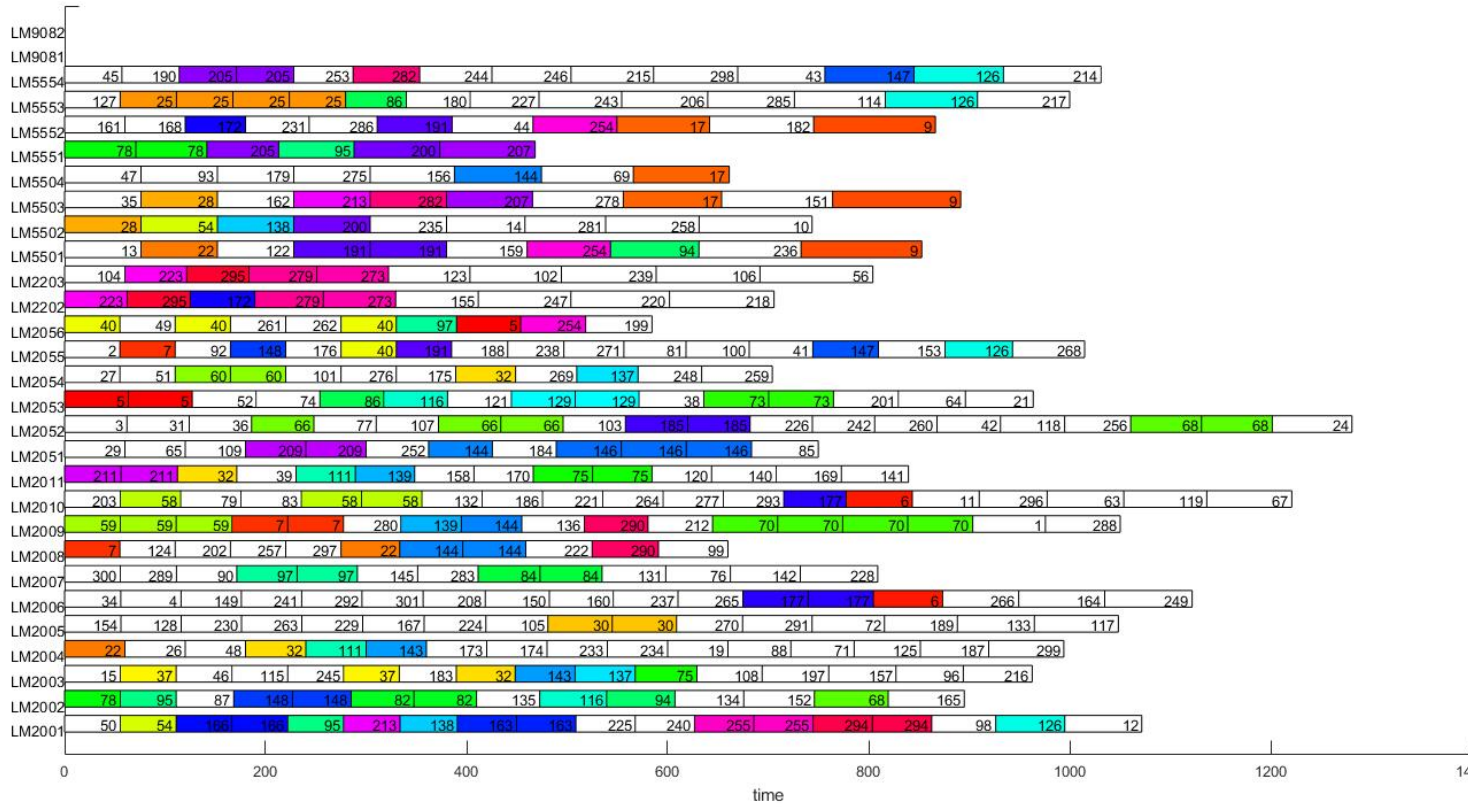

Figure 5.1: Schedule for instance 1 without resource constraints

The x-axis represents time and the y-axis represents the 29 different machines. Each rectangle represents a job. When two rectangles have the same number (colors are used to make it more clear), they share the same reticle.

The schedule in figure 5.1 has a makespan of $1280,8401$ seconds and a total completion time of $169949,9265$ seconds. In this schedule there are 43 resource conflicts - which means that 43 jobs are scheduled simultaneously with other jobs that require the same resource (see figure 5).

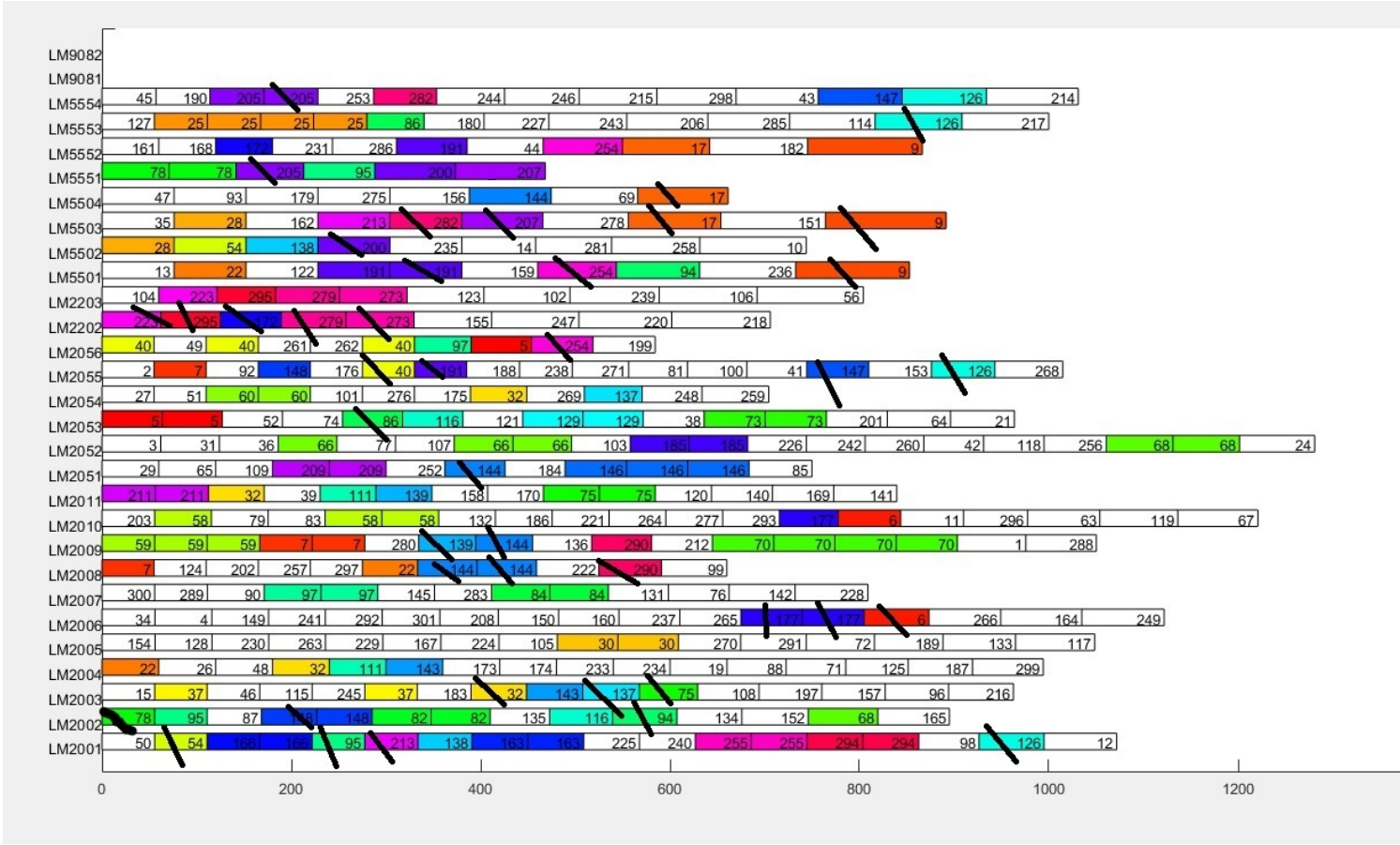

Figure 5.2: Schedule with resource constraints, denoted by black streaks

When we approximate an optimal solution for the objective function **with** respect to resource constraints with this Greedy algorithm, we find the following schedule:

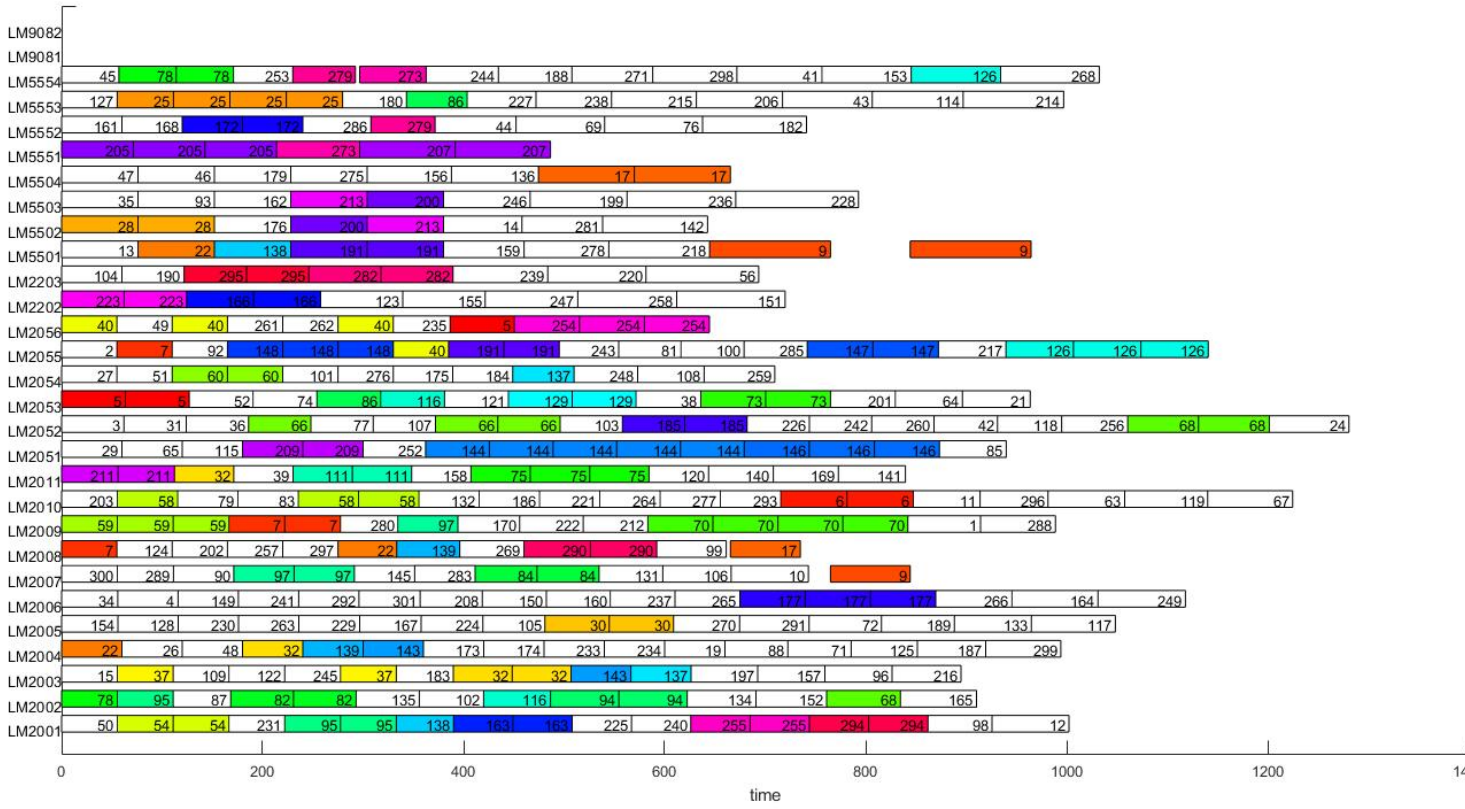

Figure 5.3: Schedule for instance 1, w.r.t. resource constraints

The schedule in figure 5.2 has a makespan of $1280,8401$ seconds and a total completion time of $170200,7562$ seconds. In this schedule, there are no resource conflicts. Furthermore, the optimal TCT (calculated without resource constraints), is equal to 163050.

We find a ratio for the optimal TCT, with the optimal TCT (without constraints) as lower bound and the TCT found in the second schedule (with resource constraints) as upper bound:

$TCT_{optimal,\ no\ res\ constr} \leq TCT_{OPT} \leq TCT_{greedy,\ with\ res\ constr}$.

So, for the first instance of our data we find $163050 \leq TCT_{OPT} \leq 170200$.

It appears that, overall, this is quite a large ratio, as can be seen in table 5.1: the ratio is 8,33% of the TCT.

Table 5.1: Avarage lower and upper bound of the 90 instances, compared to TCT size

	average lower bound without res	**average with res**	**average ratio size**	**average overall**	**% difference of overall average**
TCT	118806	125965	7159	122386	8,33%

We would like to consider a few special instances. Firstly, let us have a look at the instance with the smallest TCT optimum-ratio, instance 30. For this instance it is quite a small ratio, as can be seen in table 5.2.

Table 5.2: Instance 30: lower and upper bound, compared to TCT size

	average lower bound without res	**average with res**	**average ratio size**	**average overall**	**% difference of overall average**
TCT	119730	122320	2590	121025	2,14%

The instance with the largest ratio is instance 12. See table 5.3. Figure 5.4 and 5.5 make it clear why this is the case. The instance consists of 5 jobs that all share the same resource. Apparently, for cases like this, our method of creating this ratio is very unuseful (at least when considering the TCT).

Table 5.3: Instance 12: lower and upper bound, compared to TCT size

	average lower bound without res	**average with res**	**average ratio size**	**average overall**	**% difference of overall average**
TCT	285	840	2590	562,5	98,67%

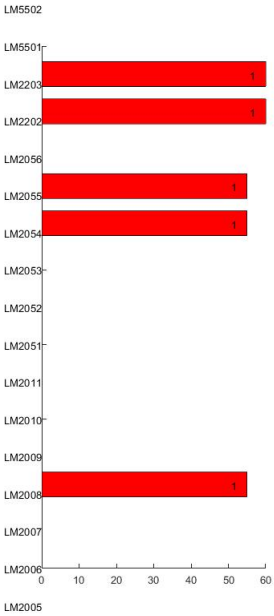

Figure 5.4: Schedule for instance 12 without resource constraints

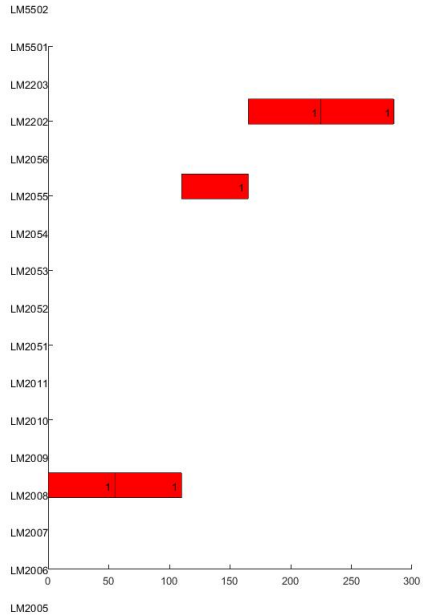

Figure 5.5: Schedule for instance 12 with respect to resource constraints

Chapter 6

Conclusion and remarks

A Greedy algorithm for our schedule problem $Rn|res \cdot 11|\sum C_j$, in combination with the lower bound for the TCT, returns quite a large ratio in which the optimal solution can be found. It is therefore probably not the best approximation algorithm, but nonetheless it was interesting to compare objective functions of problems with resource constraints to problems without resource constraints.

Of course there are some shortcomings with the way we approached our scheduling problems in the first place:

- We did not take into consideration that it takes time to move reticles in between machines
- In the whole thesis we ignored priority constraints. We did not take any into account when approximating solutions, but this is not very realistic: some jobs should always be finished before others. Also the SPT rule, which was optimal for $1||\sum C_j$ does not take any priority constraints into account. So when priority constraints do get considered, this does not guarantee an optimal solution.
- We did not take into account how long the machines can work non-stop. After what amount of time are they switched off? We sort of assumed that they would always be working, without breaks.

For future studies, it can be interesting to take the following questions into consideration:

- What is the actual optimal value for the TCT and the makespan when we do take reticles into account?
- What will happen with our approximations and the complexity when we fix reticles on to machines (so the reticles cannot be moved)?
- With the Greedy approximations, we focussed on minimizing the TCT, and found that

the ratio for an optimal solution was quite large. But how large is the ratio in which an optimal makespan should lie, when working with this Greedy algorithm?

- How much money would you actually save when you use the schedules that optimize the TCT or makespan?

- The complexity of the problems$Rm|res\cdot 11, p_j = 1,$ *one per job*$|\sum C_j$ and $Pm|res\cdot 11, p_j = 1,$ *one per job*$|\sum C_j$ remains an open question for us. It could be interesting to go more into depth in these problems.

Appendix A

Reductions

Machines in parallel with different speeds

We find the following for two machines in parallel with different speeds:

Theorem A.0.1 (Blazewich & Lenstra, theorem 3)**.** *$Q2|res \cdot 11, p_j = 1|C_{max}$ is NP-hard.*

Proof. We will show that the NP-complete problem PARTITION INTO PATHS OF LENGTH 2 can be transformed to $Q2|res \cdot 11, p_j = 1|C_{max}$.

PARTITION INTO PATHS OF LENGTH 2 *is the following problem:*

***Given**: A graph $G = (V, E)$ with $|V| = 3t$, for an integer t.*

***Question:** Can V be divided in t disjoint subsets, each with three vertices and with a maximum of two vertices that are not adjacent?*

Like in the previous proof: for every graph G with vertices V and edges E we find that functions and constraints of the type $res \cdot 11$ can be constructed as follows:
- for every vertex $j \in V$ introduce a job J_j;
- for every pair of vertices $j, k \notin E$, introduce a resource $R_{\{j,k\}}$ of size $s_{\{j,k\}} = 1$, with $r_{\{j,k\}j} = r_{\{j,k\}k} = 1$, and $r_{\{j,k\}i} = 0$ for $i \in E$.

Two jobs can be executed simultaneously if and only if the corresponding vertices are adjacent.

For every instance of the problem we can construct an instance of
$Q2|res \cdot 11, p_j = 1|C_{max}$, with machinespeeds $q_1 = 2, q_2 = 1$.

PARTITION INTO PATHS OF LENGTH 2 has a solution if and only if there is a feasible schedule with $C_{max} \leq t$. □

Appendix B

Greedy Approximations

Table B.1: A ratio for optimal makespan and TCT for all instances

instance number	**number of res. conflicts**	**makespan without res. GREEDY**	**makespan with res. GREEDY**	**TCT without res. GREEDY**	**TCT without res. LOWER BOUND**	**TCT with res. GREEDY**
1	43	1281	1281	169950	163050	170200
2	25	1443	1443	143610	134370	143610
3	55	1159	1278	161760	156580	163630
4	25	1295	1295	122040	114240	122980
5	22	1256	1256	129320	122020	130470
6	40	1160	1212	126980	123100	128280
7	36	886	948	99873	95666	101380
8	1	128	128	304	304	357
12	4	60	285	285	285	840
13	19	269	663	7038	6509	9776
14	26	1185	1185	84887	75535	84815
15	38	1091	1093	135750	131560	136440
16	16	1448	1515	116910	105900	117800
17	40	1283	1280	133170	122760	134150
18	38	1335	1370	165250	153280	165310
19	19	1151	1149	103060	94989	103560
20	18	1287	1294	94334	84989	95500
21	40	1336	1321	141830	130000	142680
22	43	1425	1425	156140	144990	156830
23	36	1422	1422	169960	160580	170830
24	39	1149	1147	152960	146700	153300

instance number	**number of res. conflicts**	**makespan without res. GREEDY**	**makespan with res. GREEDY**	**TCT without res. GREEDY**	**TCT without res. LOWER BOUND**	**TCT with res. GREEDY**
25	52	1291	1351	168380	160100	168460
26	35	1207	1270	153000	147920	153900
27	32	1292	1290	134930	128390	135690
28	44	1291	1295	156260	147580	156340
29	29	1209	1227	115340	108770	115610
30	33	1091	1023	122380	119730	122320
31	19	916	970	63641	59551	64198
32	29	1423	1359	140180	133450	140170
33	31	1150	1139	123050	116290	122860
34	29	1309	1374	126480	119750	127270
35	15	870	810	33182	29606	33261
36	0	510	510	4458	3859	4458
37	9	427	419	17102	15800	17392
38	58	1101	1220	131790	126890	133780
39	41	1438	1495	177210	169010	179020
40	28	889	899	77992	74696	79695
41	52	1250	1353	137940	129960	140840
42	32	1153	1147	144390	137710	145150
43	37	1146	1146	126150	120480	126290
44	35	1139	1139	146270	142450	146720
45	49	999	1063	141930	138790	142340
46	35	1383	1323	145420	138730	146320
47	45	1144	1144	168200	163180	169060
48	47	1236	1236	140260	131820	141470
49	35	1233	1222	131740	125580	132700
50	30	1171	1161	131630	124720	131680
51	29	1045	1097	116430	111590	117450
52	26	1521	1521	123720	115080	124560
53	32	1654	1653	144130	135260	145130
54	31	1444	1444	156950	148090	157800
55	32	1545	1552	150590	143060	151520
56	25	1294	1291	150220	142810	150450
57	49	1056	1124	129990	125010	131880
58	33	1336	1307	149790	139410	149980
59	39	1408	1342	127210	120110	127940
60	37	1151	1151	121230	116800	122440
61	23	1330	1349	151580	142440	151920

instance number	**number of res. conflicts**	**makespan without res. GREEDY**	**makespan with res. GREEDY**	**TCT without res. GREEDY**	**TCT without res. LOWER BOUND**	**TCT with res. GREEDY**
62	35	1279	1279	158060	150850	158010
63	23	1285	1349	123480	116630	123700
64	32	1371	1371	132690	127150	132720
65	32	1405	1405	171380	164690	171610
66	32	1326	1336	150320	143440	151000
67	31	1326	1322	130880	122580	131460
68	28	1274	1275	126900	117740	126310
69	31	1336	1336	152590	144890	153560
70	23	1241	1241	110060	103350	109970
71	24	1564	1564	135980	130080	136580
72	28	1407	1407	142110	136260	142710
73	36	1053	1153	114450	107950	116320
74	35	1134	1170	124500	119450	126020
75	27	1205	1216	149950	143720	150100
76	32	1235	1235	129510	122680	129790
77	44	1285	1346	123960	119990	125560
78	40	1302	1364	144340	137980	145220
79	13	1191	1191	102080	98006	101940
80	27	1272	1296	139150	128870	139680
81	30	1366	1371	145710	136800	146390
82	28	1360	1425	134560	123550	135570
83	35	1281	1281	125900	118580	127880
84	40	1171	1218	137300	129140	138220
85	17	1295	1293	140500	129220	140240
86	32	1247	1237	144990	137790	145750
87	22	1032	1032	82386	77277	82820
88	27	1219	1279	151000	143190	151880
89	31	1470	1470	122680	117190	123550
90	39	1111	1113	129700	125040	130010
91	59	1361	1404	122900	113840	123870
92	42	1196	1200	134490	125690	135760
93	32	966	1026	90046	85665	90950

Appendix C

MATLAB codes

```
close all
clear all

%Input information
% pathadd is the place where the .xls file is located
name='instance_12';
pathadd='NXP_Data/DIA_RUN_STATS_LM_SVG_201610_alternative_times/';
[~,~,Data]=xlsread([pathadd,name,'.xls']);

%Create Matlab input
[n,m]=size(Data);

[JT,lots,reticleNumb,reticleName,reticleUsage,OriginalS,MachineNames]
    = Data2JobInput(Data);

P=JT;
[n,m]=size(P);
noneligible=0;
for i=1:n
    for j=1:m
        if P(i,j)<=0
            P(i,j)=0;
            noneligible=noneligible+1;
        end
    end
end

machineTime=zeros(1,m);
P_temp=P(1:n,:);
schedule=zeros(n,2);
machine_location=ones(1,m);
reticles=zeros(n,1);
```

```
completiontime=zeros(m,20);
schedule_time=zeros(1,n);
machinetijdenmatrix=zeros(n,m);
eindtijdjob=zeros(m,25);
machineTime_temp=zeros(1,m);
resources_eindtijd=zeros(1,n);

%met resources
for i=1:n
    [antwoord,jobs]=min(machinetijdenmatrix+P_temp+(P_temp==0)*10^6);
        %neem de job met de minimale eindtijd en filter de nullen
        eruit
    [~,machine]=min(antwoord);
        %de machine waar deze job vervolgens opgaat is degene waar hij
         het snelst klaar is
    job=jobs(machine);  %benoem deze job

    %[~,machine]=min(machineTime+(sum(P_temp)==0)*10^6);

    P_temp(job,:)=zeros(1,m);
        %verwijder de geschedulede job
    %   update schedule
    schedule(job,:)=[machine,machine_location(machine)];
    schedule_time(job)=machinetijdenmatrix(job,machine);

    %update machineTime
    machineTime(machine)=machinetijdenmatrix(job,machine)+P(job,
        machine);

    resource=reticleNumb(job);
    index_rec=find(resource==reticleNumb);
    machinetijdenmatrix(:,machine)=max(machinetijdenmatrix(:,machine)
        ,ones(n,1)*machineTime(machine));
    for j=1:length(index_rec)
        machinetijdenmatrix(index_rec(j),:)=max(machinetijdenmatrix(
            index_rec(j),:),ones(1,m)*(machineTime(machine)));
    end

    %   schedulematrix(machine,machine_location(machine))=job_length;
    machine_location(machine)=machine_location(machine)+1;
    eindtijdjob(machine,machine_location(machine))=machineTime(
        machine);
end

    Fig1=schedulePlot(schedule,schedule_time,P,reticleNumb,
        reticleUsage,MachineNames,1);
    print(Fig1,'-dpng','-r400',[pathadd,'Pictures\',name,'
        _Schedule_Original'])
```

```
%uitrekenen tijden met res
makespan1=max(max(eindtijdjob));
TCT1=sum(sum(eindtijdjob));

display(['The TCT for a Greedy schedule s.t. resource constraints is
    equal to ',num2str(TCT1),'.'])
display(['The Makespan for a Greedy schedule s.t. resource
    constraints is equal to ',num2str(makespan1),'.'])


%zonder resources
machineTime=zeros(1,m);
P_temp=P(1:n,:);
schedule=zeros(n,2);
machine_location=ones(1,m);
reticles=zeros(n,1);
schedule_time=zeros(1,n);
machinetijdenmatrix=zeros(n,m);
eindtijdjob=zeros(m,25);

for i=1:n
    [antwoord,jobs]=min(machinetijdenmatrix+P_temp+(P_temp==0)*10^6);
        %neem de job met de minimale eindtijd en filter de nullen
        eruit
    [~,machine]=min(antwoord);
        %de machine waar deze job vervolgens opgaat is degene waar hij
         het snelst klaar is
    job=jobs(machine);%benoem deze job
    %[~,machine]=min(machineTime+(sum(P_temp)==0)*10^6);

    P_temp(job,:)=zeros(1,m);
        %verwijder de geschedulede job
    %   update schedule
    schedule(job,:)=[machine,machine_location(machine)];
    schedule_time(job)=machineTime(machine);
    %update machineTime
    machineTime(machine)=machineTime(machine)+P(job,machine);
    %schedulematrix(machine,machine_location(machine))=job_length;
    %machinetijdenmatrix(:,machine)=ones(n,1)*machineTime(machine);
          %update de eindtijden van de machines door de kolom te
        updaten
    machinetijdenmatrix(:,machine)=max(machinetijdenmatrix(:,machine)
        ,ones(n,1)*machineTime(machine));

    eindtijdjob(machine,machine_location(machine))=machineTime(
        machine);
    machine_location(machine)=machine_location(machine)+1;
end
```

```
        Fig2=schedulePlot(schedule,schedule_time,P,reticleNumb,
            reticleUsage,MachineNames,2);
        print(Fig2,'-dpng','-r400',[pathadd,'Pictures\',name,'
            _Schedule_Original_2'])

%uitrekenen rijden zonder res
makespan2=max(max(eindtijdjob));
TCT2=sum(sum(eindtijdjob));

display(['The TCT for a Greedy schedule without resource constraints
    is equal to ',num2str(TCT2),'.'])
display(['The Makespan for a Greedy schedule without resource
    constraints is equal to ',num2str(makespan2),'.'])

%ratio voor optimum
display(['The TCT for the optimal schedule s.t. resource constraints
    lies in between ',num2str(TCT2),' and ',num2str(TCT1),'.'])
display(['The makespan for the optimal schedule s.t. resource
    constraints lies in between ',num2str(makespan2),' and ',num2str(
    makespan1),'.'])


[n,m]=size(P);
maxPos=max(schedule(:,2));
ScheduleTime=zeros(m,maxPos);
zeroJobs=0;
for i=1:n
    if (schedule(i,1)>0) && (schedule(i,1)<(m+1)) && (schedule(i,2)
        >0)
        ScheduleTime(schedule(i,1),schedule(i,2))=P(i,schedule(i,1));
        if P(i,schedule(i,1))==0;
            zeroJobs=zeroJobs+1;
        end
    end
end

[makespan_2,TCT_2,machineTime_2,reticleConflicts_2,zeroJobs_2] =
    schedulePerformance(schedule,P,reticleNumb,reticleUsage);

display(['The number of reticle conflicts is equal to ',num2str(
    length(reticleConflicts_2)),'.'])

T = {length(reticleConflicts_2),makespan2,makespan1,TCT2,TCT1}

xlswrite('impl.xls',T);
```

References

[1] Jacek Blazewicz, Jan Karel Lenstra, and AHG Rinnooy Kan. "Scheduling subject to resource constraints: classification and complexity". In: *Discrete Applied Mathematics* 5.1 (1983), pp. 11–24.

[2] CW Duin and E Van Der Sluis. "On the complexity of adjacent resource scheduling". In: *Journal of Scheduling* 9.1 (2006), pp. 49–62.

[3] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 29. wh freeman New York, 2002.

[4] Godfrey Harold Hardy, John Edensor Littlewood, and George Pólya. *Inequalities*. Cambridge university press, 1952.

[5] Tsuyoshi Kawaguchi and Seiki Kyan. "Worst case bound of an LRF schedule for the mean weighted flow-time problem". In: *SIAM Journal on Computing* 15.4 (1986), pp. 1119–1129.

[6] Jan Karel Lenstra, David B Shmoys, and Éva Tardos. "Approximation algorithms for scheduling unrelated parallel machines". In: *Mathematical programming* 46.1 (1990), pp. 259–271.

[7] James B Orlin. "A polynomial time primal network simplex algorithm for minimum cost flows". In: *Mathematical Programming* 78.2 (1997), pp. 109–129.

[8] James H Patterson et al. "Computational experience with a backtracking algorithm for solving a general class of precedence and resource-constrained scheduling problems". In: *European Journal of Operational Research* 49.1 (1990), pp. 68–79.

[9] Michael Pinedo. *Scheduling*. Springer, 2015.

[10] Robert-H Munnig Schmidt. "Ultra-precision engineering in lithographic exposure equipment for the semiconductor industry". In: *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences* 370.1973 (2012), pp. 3950–3972.

[11] Douglas Brent West et al. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, 2001.