CHALMERS | GÖTEBORG UNIVERSITY

MASTER'S THESIS

Resource utilization in a multitask cell

Tomas Jansson

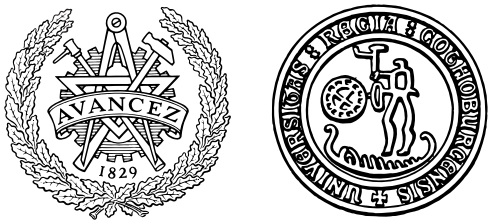
Department of Mathematics
CHALMERS UNIVERSITY OF TECHNOLOGY
GÖTEBORG UNIVERSITY
Göteborg Sweden 2006

Thesis for the Degree of Master of Science

Resource utilization in a multitask cell

Tomas Jansson

CHALMERS | GÖTEBORG UNIVERSITY



Department of Mathematics
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg, Sweden
Göteborg, May 2006

Matematiskt centrum
Göteborg 2006

Abstract

Volvo Aero has recently invested in a new production cell containing ten resources. The production cell is supposed to carry out a large variety of jobs since five of the resources are multipurpose machines that are able to process three different types of operations. The cell contains, except from the multipurpose machines, three set up and tear down stations, one automatic deburring machine and one manual deburring station. The purpose of the thesis was to study how the jobs will be scheduled with the default priority function that is delivered with the multitask cell, compared to optimal scheduling and see if there is any unused potential. To study this a mathematical model was implemented in AMPL together with numerous heuristics trying to do the scheduling as well as possible, in a reasonable amount of time. The focus of the heuristics has been to minimize the total lateness, a variable that Volvo Aero wants to keep as low as possible. The scheduling problem can be described as a job shop problem with multipurpose machines which is among the hardest combinatorial problems within optimization.

The results from the study show that there is a lot of unused potential that will be wasted if the default function is used. The scheduling could be improved considerably if the existing function is changed for one that is adapted to the production at Volvo Aero. Implementing an optimization based heuristic could improve the scheduling even further, but is much more cumbersome to implement in the real system. How the heuristic should work needs to be studied further to be applicable in reality.

Sammanfattning

Volvo Aero har nyligen investerat i en ny produktionscell som innehåller tio resurser. Eftersom fem av resurserna är multifunktionella maskiner som kan utföra tre olika typer av operationer är multitaskcellen tänkt att utföra en stor variation av jobb. I multitaskcellen finns det, förutom de multifunktionella maskinerna, tre riggnings- och rivningsstationer, en automatisk gradningsrobot och en station där manuell gradning utförs. Syftet med denna studie var att studera hur effektivt jobben kommer att schemaläggas med standardprioriteringsfunktionen som den levereras med jämfört med optimal schemaläggning, samt om det finns någon outnyttjad potential. För att studera detta har en matematisk modell samt ett flertal olika heuristiker implementerats i AMPL i ett försök att göra schemaläggningen så effektivt som möjligt, inom en rimlig tid. Fokus har legat på att minimera antalet förseningstimmar, en variabel som man hos Volvo Aero vill ha så låg som möjligt. Schemaläggningsproblemet kan beskrivas som ett job shop-problem med multifunktionella maskiner vilket är bland de svåraste kombinatoriska problemen inom optimering.

Resultaten från denna studie visar att det finns oanvänd potential som går förlorad om standardprioriteringsfunktionen används. Enbart genom att byta prioriteringsfunktione mot en som är anpassad för produktion på Volvo Aero kan schemaläggningen av jobben effektiviseras avsevärt. Implementeringen av en optimeringsheuristik i systemet skulle kunna effektivisera schemaläggningen ytterligare, men är besvärligare att realisera. Hur denna heuristik skulle fungera behövs också studeras ytterligare för att vara praktiskt tillämpbart.

Acknowledgments

I would like to thank Professor Michael Patriksson, my supervisor and examiner at Chalmers; Håkan Colliander, my supervisor at Volvo Aero; Torgny Almgren and Sofia Holm at Volvo Aero; and Ann-Brith Strömberg at Fraunhofer Chalmers Centre for their help and support.

Of course I would like to thank my wonderful girlfriend who has always been there for me during the years at Chalmers.

Contents

Chapter 1

Introduction

1.1 The Company

Volvo Aero is a subsidiary of AB Volvo. The company's main area is developing and producing components with a high technology content for aircraft and rocket engines. Service and maintenance of their products is also an increasingly important part of their business. Volvo Aero has its head quarters at Trollhättan, Sweden, where this project has taken place. Except from Trollhättan they have two more offices in Sweden, one in Norway and three in the USA.

1.2 Background

At the moment of writing Volvo Aero is about to build a new production cell, which is an extensive project. The new production cell will contain ten resources, machines and work stations, when finished. Five of these resources are multipurpose machines, called multitask machines, that can carry out a variety of operations; the multitask machines work in parallel instead of just one dedicated for every product. The multitask cell is delivered with a default priority function that is not adapted for Volvo Aero. This is a totally new concept for Volvo Aero that rises new questions. The main question is how well the jobs will be scheduled in the multitask cell; is the default priority function satisfying? Also, the lead times for the products will vary from time to time depending on the number of jobs in the multitask cell, which makes it hard to set exact due dates.

1.3 Purpose of the thesis

The purpose of this thesis is to study the resource utilization in the new production cell at Volvo Aero. The production cell is, as mentioned above, delivered with a priority function and is a rather naive but fast algorithm for job scheduling. Priority functions are easy to implement and work rather well, even if there are disturbances in the system. That is why they are often used in industry. The main objectives are to study how well the jobs in the production cell will be scheduled using the default function, and to determine whether there will be any unused potential in the production cell.

The production cell will be studied from a mathematical point of view. A mathematical model of the multitask cell will be implemented where the default function together with some alternatives could be simulated. Some optimization heuristics that will use the mathematical model will be developed and implemented to compare to the default function. The total lateness will be used to measure the quality of the result for the different heuristics and functions.

Hopefully the study results in an algorithm or another priority function that can be used preferably instead of the existing priority function, but the main objective is to show that there is unused potential in the production cell.

1.4 Outline

Chapter 2 introduces scheduling problems in general, a description of the multitask cell, and complexity classes. In Chapter 3 a complete optmization model of the production cell is described. Because of the complexity of the problem it would take too long to find an optimal schedule for larger examples. However, it is possible to create optimal schedules for smaller examples; to show the potential an example is shown in Section 3.4. Chapter 4 presents the different heuristics that were implemented and tested. The existing priority function and some other priority functions that were tested are presented in Section 4.1. How the model of the production cell, the heuristics and a visualization module for creating Gantt charts of the solutions was implemented can be found in Chapter 5. The results from the tests are found in Chapter 6. The final part of the report is the discussion and conclusions part in Chapter 7.

Chapter 2

Scheduling problems

Scheduling problems have been studied for ages since there is a lot of time and money to save from using effective scheduling approaches. Some examples where scheduling is used are: to minimize the length of the routes for logistic companies, to have effective personnel scheduling for airline companies and to minimize the lead times for manufacturing companies.

The area that has got the most attention in the area of scheduling is static scheduling. However, in later years it has been more and more common with dynamic scheduling problems. One of the reasons is the introduction of the Just-In-Time (JIT) systems and the fact that companies strive to be more flexible and not have too much capital locked up in raw material etc. The main difference between static and dynamic scheduling is that in a static environment all the parameters, such as arrival times of products and processing times for the operations, are available from the start. In dynamic scheduling these type of parameters can change during execution time, for example when jobs are late. This implies that in a dynamic scheduling environment it is possible to schedule jobs where the exact arrival time is not always available. Most manufacturing problems are of the dynamic type since they have some stochastic variables, such as machine breakdowns or that an operation is stalled for some reason, making the exact arrival time hard to predict. A semi-dynamic optimization method (see Section 4.2) was tested in one of the attempts to create an optimal schedule. The method is only quasi-dynamic since it is optimizing in discrete time steps and the problem is static in every step. Many dynamic scheduling problems can be transformed to their respective static scheduling problem and solved using an iterative optimization method. Longer time intervals in the iterative method will make room for more stochastic events, which might disrupt the existing schedule.

In Section 2.1 a method for characterizing static scheduling problems is

presented. The method described is used to classify the problem in Section 2.3 after a description of the multitask cell in Section 2.2. Section 2.4 describes some previous work related to this topic. Section 2.5 describes complexity classes to make it clear how hard this problem is.

2.1 Classification of scheduling problems

It is common to use a three-variable notation, $\alpha|\beta|\gamma$, as the one described by Anderson et al. in [MS97] for describing machine scheduling problems. Each variable could be seen as a field that contains one or more variables. The α-variable denotes machine environment, β-variable defines the job characteristics and the γ-variable defines the optimality criterion. More details are given in every separate section.

2.1.1 Characterization of the machine environment

The α field is a two-variable field of the form $\alpha = \alpha_1\alpha_2$. The α_1-variable can be one or several of the variables in the set $\{\circ, P, Q, R, O, F, J, MPM\}$, where

- $\circ$: single machine
- P: identical parallel machines
- Q: uniform parallel machines
- R: unrelated parallel machines
- O: open shop problem
- F: flow shop problem
- J: job shop problem
- MPM: **M**ulti**P**urpose **M**achines

Not all combinations are possible. For example it is not possible to have both one and two machines at a time; however, it is possible to have a shop problem combined with multipurpose machines. The different variables are interpreted as follows. If $\alpha_1 \in \{\circ, P, Q, R\}$, every job consists of only one operation that can be processed on any of the available machines. If $\alpha_1 = \{\circ\}$, there is only one machine available. If $\alpha_1 = \{P\}$, where P refers to identical parallel machines, the processing time for every job is the same on every machine. If $\alpha_1 = \{Q\}$, where Q refers to uniform parallel machines, the processing time might differ between the machines. If $\alpha_1 = \{R\}$, where

R refers to unrelated parallel machines, the processing time on each machine is both job dependent and machine dependent.

Let J_j denote a job number j and O_{ji} denote operation number i of job j. (Note that J and O denoting the job shop and open shop problems are not the same as J_j and O_{ji} denoting a job and an operation.) If $\alpha_1 \in \{O, F, J\}$ the jobs consist of more than one operation, which is denoted as $J_j = \{O_{1j}, \ldots, O_{mj}\}$ for a job consisting of m operations. If $\alpha_1 = O$, where O refers to an open shop problem, every operation O_{ij} has to be processed on machine M_i but the order in which the operations are processed is irrelevant. When $\alpha = F$, where F refers to a flow shop problem, the order in which the operations is processed is relevant: operation O_{1j} has to be processed before operation O_{2j} and so on. The last problem is the job shop problem when $\alpha = J$. The job shop problem is similar to the flow shop problem except that jobs might differ. That is, the jobs can differ in number of operations, processing times and machines needed for processing.

If $\alpha_1 = MPM$ it means that at least one machine in the environment can process different types of operations.

α_2 denotes the number of machines, but is often left out if the number is arbitrary.

2.1.2 Characterization of the job characteristics

The second field in the three-field notation denotes processing relations and restrictions. Since not all problems have the same relations β is a subset of some common relations, $\beta \subseteq \{\beta_1, \beta_2, \beta_3, \beta_4\}$. The β-variables are interpreted as follows:

- $\beta_1 = r_j$, if the release dates are specified
- $\beta_2 = d_j$, if due dates are specified
- $\beta_3 = t_{jk}$, if there are setup times for the operations
- $\beta_4 = prec$, if there are precedence constraints between the jobs

Note that these are the most common relations and restrictions, not all possible relations and restrictions.

2.1.3 Optimality criteria

The γ-field in the three-variable notation denotes the optimality criterion that is used to minimize the cost and measure the results. Common criteria are:

- $\gamma = C_{max}$, which is to minimize the maximum completion time,
- $\gamma = \sum_{j \in \mathcal{J}} w_j C_j$, which is to minimize the total (weighted) finish time for all jobs j in the set $\mathcal{J}$ of jobs,
- $\gamma = L_{max}$, which is to minimize the maximum lateness.

There is no universal rule to determine which criterion should be used; the criteria differ depending on problem and purpose. There is always the possibility to compose other optimality criteria that might fit the problem better.

2.2 The multitask cell

When the multitask cell is mentioned in the report it refers to the whole production cell including all ten resources. Which are the different resources in the multitask cell?

- There are three workstations where the raw products are set up or torn down from their fixtures manually.
- There are five multitask machines that can perform drill, mill or lathe operations on the products.
- There is one automatic deburring machine where most of the deburring is done.
- There is one manual deburring station where all the manual deburring is done; not all deburring can be done with the automatic machine.

A product is set up on a fixture so it is securely fixed before the operating machine can process an operation on the product. Connected with the multitask cell is a stock containing tools that are used in different processes and stock places for the products. There are two stocker cranes in the multitask cell, one for transporting the products and one for transporting the tools to the resources. See Figure 2.1 for a graphical illustration of the multitask cell.

The system is delivered with a priority function that is used for scheduling the jobs in the multitask cell, which is presented in Section 4.1.

2.3 Classification of the multitask cell

To get a complete characterization according to the three-variable notation the following questions need to be answered:

1. What does the machine environment look like?

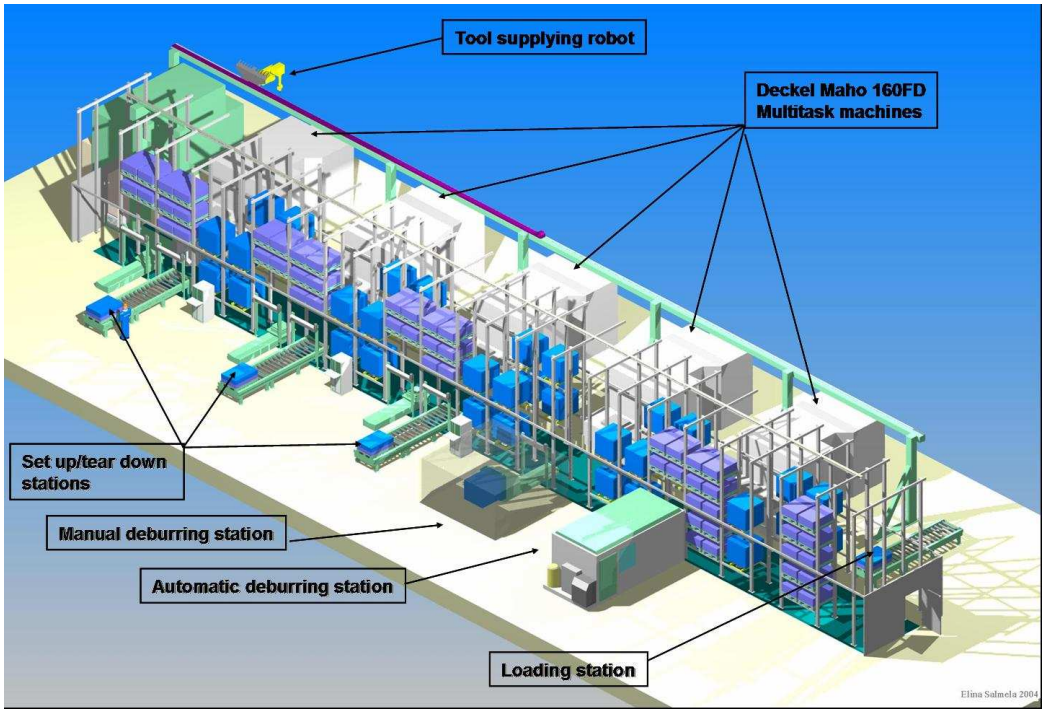


Figure 2.1: Graphical overview of the multitask cell.

2. Which are the parameters that specify a job?

3. What is the optimization criterion?

The machine environment in the multitask cell is a job shop environment since it can be defined as a set of jobs where the number of jobs, processing time of the operations and the route for the jobs through the multitask cell might differ between the jobs. The three set up and tear down stations and the five multitask machines make the multitask cell an MPM environment. Hence, $\alpha = \alpha_1\alpha_2 = JMPM10$, which means a job shop environment with ten resources out of which eight are multipurpose machines.

Which are the parameters that specify a job? Release dates, r_j, specify when the jobs will be available. Due dates, d_j, are soft constraints; jobs are allowed to be late but it will cost more. There are no precedence constraints between the jobs. There is no setup time for the jobs or operations, but it takes time to transport a job between two resources that need to be considered. This time is always the same, independent of the job, and no index is therefore needed; the parameter will be called waiting time and be denoted w. The β-field with release dates, due dates and waiting time is denoted as $\beta = \beta_1\beta_2\beta_3 = r_j d_j w$.

What is the optimization objective? For this study two different optimality criteria have been used. The reason is that the preferred optimality criterion made the optimization problem too difficult to solve. The preferred criterion is first of all to minimize the number of late hours, that is, $\min \sum_{j \in \mathcal{J}} L_j$ where L_j is the number of late hours for job j in the set $\mathcal{J}$ of jobs. Note that L_j is 0 if job j finishes on time. It is also desirable to minimize the total completion time to get the jobs scheduled as early as possible when they are on time. Hence, the preferred objective function is $\min \sum_{j \in \mathcal{J}} (C_j + L_j)$, and the preferred optimality criterion is written as $\gamma^1 = \sum_{j \in \mathcal{J}} (C_j + L_j)$. The consequence is that when a job is late the value of this objective function increases two cost units every time unit the job is late, compared to only one cost unit when it is on time. Therefore, the main purpose of γ^1 is to eliminate late jobs. The second optimality criterion used was to only minimize the total lateness, $\sum_{j \in \mathcal{J}} L_j$. This gives $\gamma^2 = \sum_{j \in \mathcal{J}} L_j$.

The optimality criteria for the problem were chosen depending on the focus of the study. Another focus may have resulted in a different objective function, but the environment and the parameters would remain the same. The three-variable notation is written as:

$$\alpha|\beta|\gamma = JMPM10|r_j d_j w_{ji}| \sum_{j \in \mathcal{J}} (C_j + L_j), \text{ and} \tag{2.1}$$

$$\alpha|\beta|\gamma = JMPM10|r_j d_j w_{ji}| \sum_{j \in \mathcal{J}} L_j. \tag{2.2}$$

2.4 Previous work

There are not that many books and articles discussing job shop problems with multipurpose machines of the same magnitude as the problem of scheduling the jobs in the multitask cell. A lot of work has been done regarding the job shop scheduling problem, but not that many are dealing with a job shop environment containing multipurpose machines. The majority of the books are theoretical studies of scheduling in general. It is difficult to find books with real examples taken from industry. The most relevant book found for this study was [Bruc01], although it does not apply to this study. The problem that is discussed analytically in [Bruc01] only consider two jobs. Other books that are dealing with scheduling problems are [MS97] and [KV02].

The regular job shop problem and the job shop problem with multipurpose machines differ a lot, which makes it hard to generalize methods that can solve job shop problems to be applicable to job shop problems with multipurpose machines. The main difference is that in a job shop problem only

the operation order on the machines needs to be decided; in a job shop problem with multipurpose machines the machine selection for each operation also needs to be decided.

2.5 Complexity

Before explaining the complexity of the problem there is a short introduction to the concept of NP-completeness and complexity classes.

2.5.1 NP-completeness

In computational complexity theory, a complexity class is a set of problems of related complexity. A decision problem Π is said to belong to the complexity class P if it is solvable in polynomial time. Time refers to the number of operations related to the input size n and a decision problem means a problem that has the answer *yes* or *no*. This means that a problem Π that is in the complexity class P and has input size n can be solved using less than cn^k operations, where c and k are constants. When a problem is classified as *NP* (nondeterministic polynomial) it means there is no known algorithm for solving it in polynomial time. However, with a solution to an NP-problem you can verify the solution in polynomial time. A problem is *NP-hard* if the algorithm for solving the problem can solve all problems in NP. This means that an NP-hard problem is *at least* as hard as any problem in NP. The problems that are *NP-complete* are those that are both in NP and are NP-hard, that is, the most difficult problems in NP. NP-complete problems can be verified in polynomial time and all NP-problems can be reduced to any NP-complete problem. A graph illustrating the relations between the complexity classes is shown in Figure 2.2.

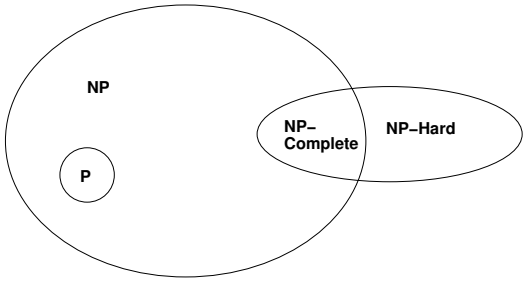


Figure 2.2: Relations between the classes *P*, *NP*, *NP-Hard* and *NP-complete*.

This means that *NP-complete* problems are usually much more difficult

to solve than problems in P. See [GJ79] for further reading about complexity classes and NP-completeness.

2.5.2 Complexity of the multitask cell

Scheduling regular job shop problems in general is among the hardest combinatorial problems. When adding the multipurpose machines to the job shop problem the complexity of the problem increases significantly. The multipurpose machines add the aspect of machine selection to the job shop problem, which is not an easy task. The MPM job shop problem with two machines and three machines is already NP-hard according to [Bruc01], which means that this problem is at least as hard as any NP-problem, if not harder still. Eight machines added and more jobs makes it even more complex. The conclusion is that the multitask problem is in the complexity NP-hard, which means that there is no known algorithm for solving the problem in polynomial time.

Chapter 3

Mathematical modeling

In Section 3.1 the fundamentals of (mixed) integer programming are presented. Section 3.2 will give an informal description of the multitask cell and in Section 3.3 a formal mathematical description is given. In Section 3.4 there is a small example illustrating the difference between the result from the existing priority function compared to an optimal solution in an extreme situation where a lot of decisions need to be made.

3.1 Mixed integer programming (MIP)

The multitask cell problem is a problem with both real valued and integer valued variables. The real valued variables represent starting time of the operations and the integer variables represent machine allocation and relations between jobs and operations. This section introduces the basics about *linear*, *integer* and *mixed integer* programming.

The optimization area devoted to solving *linear programs* is sometimes called *linear optimization* and refers to the problem of minimizing (or maximizing) a linear function over a convex polyhedron specified by linear and non-negativity constraints. A general linear problem with non-negativity constraints is written as

$$\begin{aligned}
&\text{minimize} && f(x) = c_1x_1 + \cdots + c_nx_n, && \text{(3.1a)}\\
&\text{subject to} && a_{11}x_1 + \cdots + a_{1n} \leq b_1, && \text{(3.1b)}\\
& && \vdots &&\\
& && a_{m1}x_1 + \cdots + a_{mn} \leq b_m, && \text{(3.1c)}\\
& && x_i \geq 0, \quad i = 1, \ldots, n. && \text{(3.1d)}
\end{aligned}$$

The vectors $x = (x_1, \ldots, x_n)^{\mathrm{T}}$ that satisfy the constraints (3.1b)–(3.1d) are called feasible solutions. A linear program is *unbounded* if there exists a

feasible solution x for every real number K such that $f(x) < K$. A linear program has an optimal solution if it has at least one feasible solution and it is not unbounded.

A problem with integer constraints on the variables x_i is an *integer (linear) program*. If not all variables in a problem are restricted to integer values the problem is a *mixed integer (linear) program*. The multitask cell problem falls under the last category.

3.2 Model description

The following list describes what is needed in the multitask model:

- sets:
  1. one set to represent the jobs,
  2. one set of operations for each job,
  3. one set to represent the resources (machines),
- parameters:
  4. one parameter for each job to represent the release dates,
  5. one parameter for each job to represent the due dates,
  6. one parameter for each operation to represent where the operation can be processed,
  7. one parameter for each operation to represent the processing times,
  8. one parameter for the transportation time,
  9. one parameter for each resource to represent when the resource will be available,
- variables:
  10. one variable for each operation to represent the starting times,
  11. one variable for each operation to represent where the operation will be processed,
  12. one or more variables to represent the order in which the operations are processed on the resources,
- constraints:
  13. one constraint to secure that each operation is only processed on one resource,

14. one constraint to secure that a machine only processes one job at a time,
15. one constraint to secure that the release dates are not violated,
16. one constraint to secure that the operations within a job are processed in the right order,
17. one constraint to simulate that some resources are occupied at start up, and

- objective function:
  18. one objective function to optimize.

The following aspects have been left out due to the complexity of the problem and because they are not considered to limit the performance of the multitask cell.

- The number of fixtures: it is assumed that the fixture needed is always available.
- The staffing requirements: it is assumed that there is always staff available.
- The tool requirements: it is assumed that the stock is infinite and that the tools are transported from the stock to the resource instantaneously.
- The stock capacity: it is assumed that there is always room for the products in the stock of the multitask cell.

3.3 Mathematical description

The list on page 18 presents three distinct parts of the model; the *sets* of jobs and machines, the *parameters* and the *variables*. These are described using mathematical notation in their corresponding section.

3.3.1 Definitions of sets

$\mathcal{J}$ = the set of jobs j that is to be done
$\mathcal{N}_j$ = an ordered set of the n_j operations i that is a part of job j, $i \in \mathcal{N}_j = \{1, \ldots, n_j\}$
$\mathcal{K}$ = the set of machines that are available (3 set up/tear down stations, 5 multitask machines, 1 manual deburring station, 1 automatic deburring station), $(\mathcal{K} = 1, \ldots, 10)$.

These sets cover the *sets* item in the list on page 18. An *operation* is defined as a part of a job that requires a resource for some period of time. A resource can only process one operation at a time.

3.3.2 Definitions of parameters

$$a_{jik} = \begin{cases} 1, & \text{if operation } (j,i) \text{ can be} \\ & \text{processed on resource } k, \\ 0, & \text{otherwise,} \end{cases} \qquad j \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad k \in \mathcal{K},$$

p_{ji} = the processing time for operation i of job j,
r_j = the release date for job $j \in \mathcal{J}$,
d_j = the due date for job $j \in \mathcal{J}$,
w = the transportation time for a product inside the multitask cell,
M = a sufficiently large positive integer number,
s_k = the time when resource $k \in K$ will be available the first time.

The parameter M is needed to construct some of the constraints in Section 3.3.4. The parameters above complete the *parameters* item in the list on page 18.

3.3.3 Definitions of variables

x_{jik} = the starting time of operation (j,i) on machine k,

$$z_{jik} = \begin{cases} 1, & \text{if operation } (j,i) \text{ is} \\ & \text{allocated to machine } k, \quad j \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad k \in \mathcal{K}, \\ 0, & \text{otherwise,} \end{cases}$$

$$y_{jipqk} = \begin{cases} 1, & \text{if operation } (j,i) \text{ is being processed} \\ & \text{before } (p,q) \text{ on machine } k, \\ 0, & \text{otherwise,} \end{cases} \quad \left\{ \begin{array}{l} j,p \in \mathcal{J},\ i \in \mathcal{N}_j, \\ q \in \mathcal{N}_p, \\ (j,i) \neq (p,q), \\ k \in \mathcal{K}, \end{array} \right\}$$

c_j = cost for job j, which is defined as the lateness of the job.

These variables complete the *variables* item in the list on page 18. From the definition of the variables it is obvious that the problem is a mixed integer program; the x_{jik} are the real valued variables, and z_{jik} and y_{jipqk} are the integer (binary) variables.

3.3.4 Definitions of constraints

The constraints are defined using the previous parameters and variables. The first constraints ensure that an operation can only be processed on exactly one machine from the set of valid machines for that operation [equations (3.2)–(3.3)], and if two operations are on the same machine there has to be an ordering between the operations [equations (3.4)–(3.5)]:

$$\sum_{k \in \mathcal{K}} z_{jik} = 1, \quad j \in \mathcal{J}, \quad i \in \mathcal{N}_j, \tag{3.2}$$

$$z_{jik} \leq a_{jik}, \quad j \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad k \in \mathcal{K}, \tag{3.3}$$

$$y_{jipqk} + y_{pqjik} \leq a_{jik}, \quad \begin{cases} j,p \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad q \in \mathcal{N}_p, \\ k \in \mathcal{K}, \quad (j,i) \neq (p,q), \end{cases} \tag{3.4}$$

$$y_{jipqk} + y_{jipqk} + 1 \geq (z_{jik} + z_{pkq}), \quad \begin{cases} j,p \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad q \in \mathcal{N}_p \\ k \in \mathcal{K}, \quad (j,i) \neq (p,q). \end{cases} \tag{3.5}$$

The previous constraints force the solution to have an ordering of the machines, but the solution needs to have a valid ordering to make the starting times reasonable. The following constraints specify a valid ordering and require a starting time for every operation [equations (3.6)–3.8)]:

$$\begin{array}{r} M(1 - y_{jipqk}) + M(1 - z_{pqk}) + x_{pqk} \geq x_{jik} + p_{ji} - M(1 - z_{jik}), \\ j,p \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad q \in \mathcal{N}_p, \quad (j,i) \neq (p,q), \quad k \in \mathcal{K}, \end{array} \tag{3.6}$$

$$M \cdot y_{jipqk} + M(1 - z_{jik}) + x_{jik} \geq x_{pqk} + p_{pq} - M(1 - z_{pqk}), \quad j, p \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad q \in \mathcal{N}_p, \quad (j,i) \neq (p,q), \quad k \in \mathcal{K}, \tag{3.7}$$

$$M(1 - z_{j,i+1,k}) + x_{j,i+1,k} \geq x_{jik} + p_{ji} + w - M(1 - z_{jil}), \quad j \in \mathcal{J}, \quad i \in \mathcal{N}_j \setminus \{n_j\}, \quad k, l \in \mathcal{K}. \tag{3.8}$$

Equation (3.6) states that if operation (j,i) (operation i of job j) is to be processed before operation (p,q) on resource k ($y_{jipqk} = z_{jik} = z_{pqk} = 1$), the starting time ($x_{pqk}$) of operation (p,q) has to be after the finish time ($x_{jik} + p_{ji}$) of operation (j,i). Equation (3.7) states the same as equation (3.6) with the difference that (p,q) is scheduled before (j,i). Equation (3.8) states that the operations within the same job has to be processed in the right order.

The last types of constraints needed for the actual model are the non-negativity and integer (binary) constraints on the variables [equations (3.9)–(3.11)]:

$$z_{jik} \in \{0,1\}, \qquad j \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad k \in \mathcal{K}, \tag{3.9}$$

$$z_{jipqk} \in \{0,1\}, \qquad j, p \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad q \in \mathcal{N}_p, \quad k \in \mathcal{K}, \tag{3.10}$$

$$x_{jik} > 0, \qquad j \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad k \in \mathcal{K}. \tag{3.11}$$

To simulate that the resources are occupied at time zero one more constraint is required [equation (3.12)]:

$$x_{jik} - s_k z_{jik} \geq 0, \qquad j \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad k \in \mathcal{K}. \tag{3.12}$$

Equation (3.12) states that if operation (j,i) is processed on machine k, that is, $z_{jik} = 1$, then it has to start after the simulated operation on machine k is done. This constraint is not needed if all resources are available at time zero. (If $s_k = 0$ for all k the constraint will be redundant.)

3.3.5 Objective functions

Two objective functions have been used to complete the *objective function* item in the list on page 18. One function minimizes the total lateness [equation (3.13)], and one also strives to minimize the finish times of the early jobs [equation (3.14)]; they are discussed in Section 2.3. The objective functions are:

$$\text{minimize} \sum_{j \in \mathcal{J}} L_j, \tag{3.13}$$

$$\text{minimize} \sum_{j \in \mathcal{J}} (C_j + L_j), \tag{3.14}$$

where C_j is the completion time of job j and L_j is the lateness of job j, which is 0 if the job is on time. The variables C_j and L_j are (using the x, y and z variables and the due date parameters) formed as:

$$C_j = \sum_{k\in} x_{j,n_j,k} + p_{jn_j},$$

$$L_j = \begin{cases} C_j - d_j, & \text{if } C_j > d_j, \\ 0, & \text{otherwise,} \end{cases}$$

where d_j is the due date for job j and n_j is the last operation of job j.

3.4 A small example of the model

The following small and fictitious example illustrates the model and also serve as a further motivation for this project. The example is quite difficult since the difference between a "good" and an optimal solution is easier to see in a "stressed situation" with a heavy load and a lot of choices that needs to be done in order to make the schedule optimal. This is also the most interesting scenario since Volvo Aero wants to utilize the multitask cell as much as possible.

The example is scheduled using two different methods. The first method is based on the existing priority function where the operations are scheduled according to their priority number every time a resource is available. The second method generates an optimal schedule using the objective function in equation (3.13).

3.4.1 Input to the example

The input below can also be found as AMPL code in Appendix A.2, where it serves as example input code to the AMPL model. The machines in the example are two set up and tear down stations, two multitask machines, and two deburring machines. The input data for the jobs is specified in Table 3.1.

The parameters that simulate that there are existing jobs in the multitask cell at time 0 are also set and are found in Table 3.2. *Mx* in the table is used in the machine allocation table (Table 3.3) in the result section (Section 3.4.2) for the example; the number following *M* is used to denote which machine an operation is allocated to. The processing times for the different operations and where they can be processed are found in the AMPL code for the input in Appendix A.2.

Products	prdX	prdY
Quantity	2	3
Operations/product	11	7
Time when first arrives	2	2
Interval (hrs.)	15	15
Expected lead time (hrs.)	68.33	17.05

Table 3.1: Input data for the jobs in the example. An operation is defined as something that requires one of the ten resources.

Machine	**Machine is free at time**
Set up/tear down (M1)	7
Set up/tear down (M2)	6
Multitask (M3)	8
Multitask (M4)	10
Manual deburring (M5)	17
Automatic deburring (M6)	25

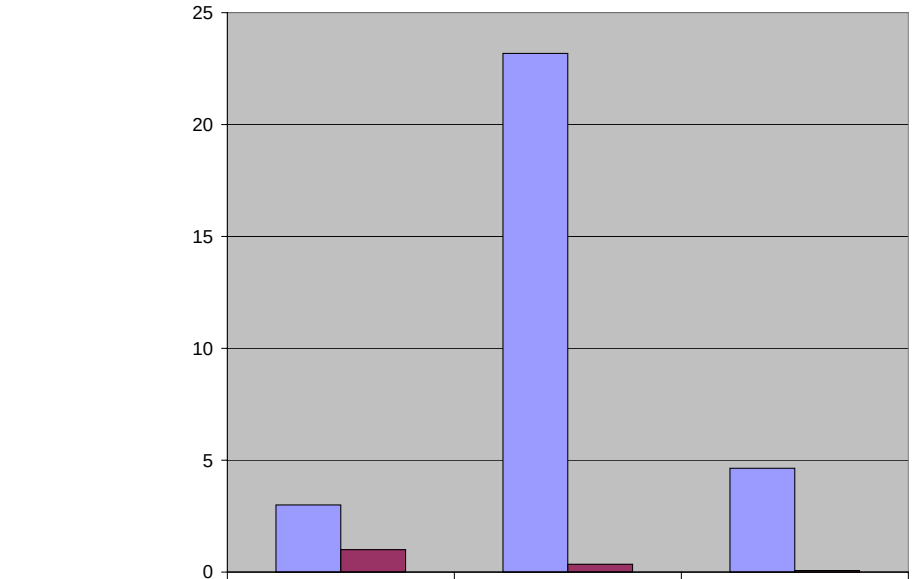
Table 3.2: The table shows when the machines are ready to process their first operation, assuming that the scheduling starts at time 0.

3.4.2 Result of the example

The most relevant data is presented in Figure 3.1. The lead times of the products and how the machines are utilized are not included since the optimization method is only focusing on minimizing the total lateness. In this example the total lateness were decreased with 98% and the number of late jobs were decreased with 66% in the optimal schedule compared to the existing priority function. This indicates that the existing priority function could yield a schedule far from optimal in a stressed environment.

How the operations of the jobs were allocated is shown in Table 3.3. The number in the table denotes the machine number, i.e. 1 stands for machine M1 and so on.

Figures 3.2–3.3 show when every operation should start but not on which machine. To get the complete schedule both Table 3.3 and the Figures 3.2–3.3 have to be considered.



	# Late jobs	Lateness	Average lateness
Existing priority function	3	23,18	4,64
Optimal solution	1	0,35	0,07

Figure 3.1: Performance of two schedules for the example.

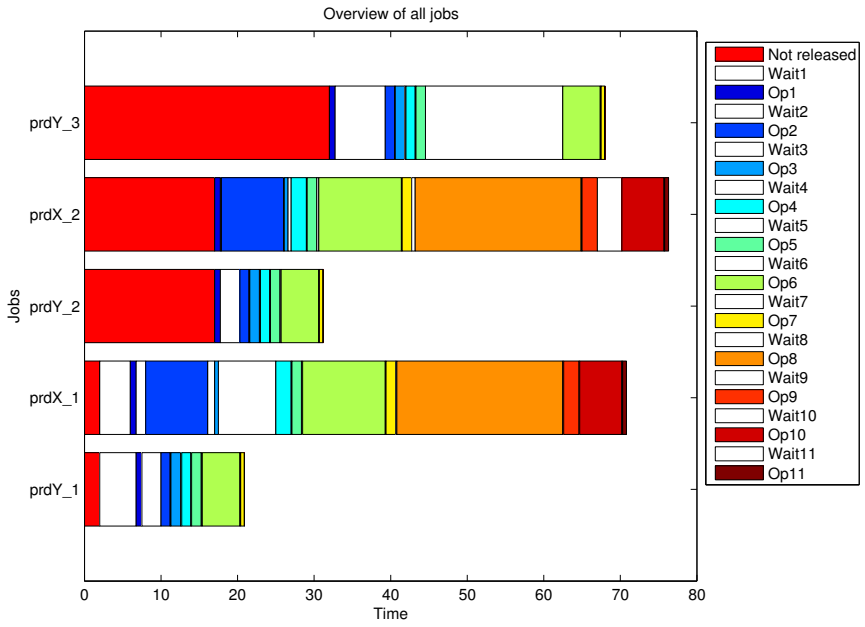


Figure 3.2: Gantt chart for the existing priority function.

Existing priority function												
		Operations										
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Products	prdY_1	2	4	1	4	2	4	2	-	-	-	-
	prdX_1	2	3	5	6	1	4	1	4	5	6	2
	prdY_2	2	4	2	4	2	3	2	-	-	-	-
	prdX_2	1	3	5	6	2	3	2	3	5	6	2
	prdY_3	2	4	2	3	1	4	1	-	-	-	-
Optimal												
		Operations										
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
Products	prdY_1	2	3	2	3	1	4	1	-	-	-	-
	prdX_1	1	3	5	6	2	4	1	4	5	6	1
	prdY_2	1	4	1	3	1	3	1	-	-	-	-
	prdX_2	1	4	5	6	2	3	2	3	5	6	1
	prdY_3	1	3	2	3	1	3	2	-	-	-	-

Table 3.3: The machine allocation for the different methods.

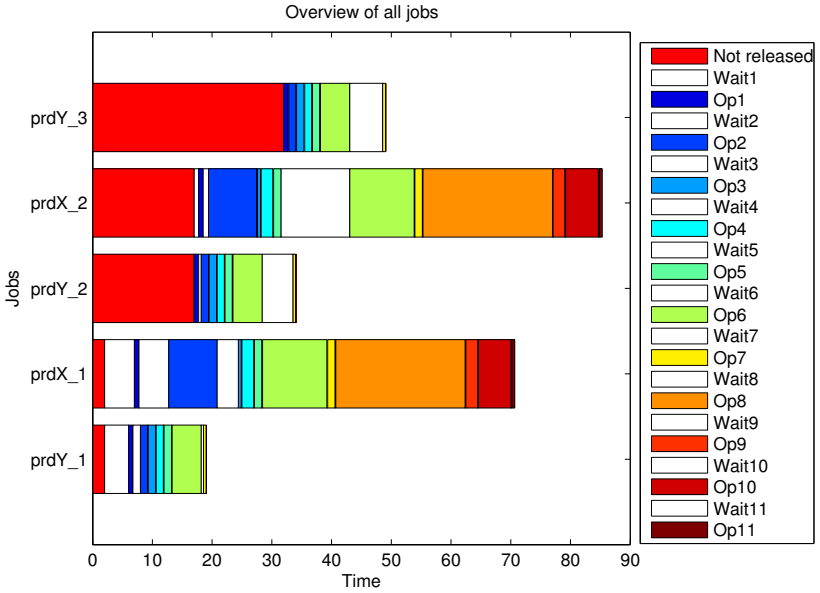


Figure 3.3: Gantt chart for the optimal schedule.

Chapter 4

Heuristics

For small-scale examples it is clearly possible to find optimal schedules. However, it would take too long to find an optimal solution when increasing the number of resources and jobs for such a strategy to work in reality, since the solution time increases exponentially with the size of the problem. It took approximately two hours to solve the small example described in Section 3.4 when using a computer dedicated for high performance computing and default values in CPLEX (the software tool used for optimization); solution time would increase significantly when the problem size is increased. The example in Section 3.4 contained a minimum number of resources for the multitask cell to still have the same properties as the full-scale multitask cell. If any of the set up/tear down stations or multitask machines were removed the multitask cell would loose the ability to operate on two different jobs that require the same type of resource at the same time. If any of the remaining resources were removed the multitask cell would no longer be able to carry out all operations. Therefore, it is necessary to consider heuristics to find a solution closer to optimum.

Before describing the different heuristics tested during the thesis work Section 4.1 describes how priority functions are used. Priority functions are described first since the existing solution goes under this group of functions and because most of the heuristics tested use a priority function to generate a feasible solution to start with. Sections 4.2–4.6 present the optimization based heuristics. The term "optimization based heuristics" refers to heuristics that in some way strives to "lay the puzzle" instead of just prioritizing the operations.

In all of the heuristics a priority function is used to generate a first feasible schedule from the input. Some advantages from this are that the result cannot be worse than the existing solution generated by the priority function, that there is always a feasible schedule to use in case of failure of the more

advanced heuristic, and that the heuristics might run faster since solutions that are worse than starting solution can be disregarded from the search.

4.1 Priority functions

Priority functions are easy to use since they are relatively easy to implement and understand, they are stable since they always succeed to generate a schedule, and they produce solutions quickly. The priority function delivered with the multitask cell is not in any way adapted to what the jobs look like at Volvo Aero. Together with the existing function three new functions that are modifications of the existing one, and four completely new ones, are presented below. One of the functions was modified to show how the jobs would be least effectively prioritized. This is rather useful information since it points out a working direction when designing priority functions. More functions were tested but the functions presented here are the ones that gave the best results and have been tested in the simulation environment at Volvo Aero (see Section 6.1).

The existing priority function consists of two different formulas, one that is used as long as the jobs are on time and one that is used for late jobs. The variable CR, which stands for ***C**ritical **R**atio*, is used to denote the relative priority number. This value is calculated for all the remaining operations within a job; the lowest CR is the job's overall CR-value. The idea is that the jobs are prioritized according to future bottleneck operations. The formulas for the critical ratio of the existing function are:

$$CR = \begin{cases} \frac{1+(d_j-t)(machines)}{1+TRPT}, & \text{when job } j \text{ is on time,} \\ \frac{1}{(1+(t-d_j)(machines))(1+TRPT)}, & \text{when job } j \text{ is late,} \end{cases} \tag{4.1}$$

where

$$\begin{aligned} t &= \text{current time,} \\ TRPT &= \text{total remaining processing time,} \\ machines &= \text{number of machines that can process the operation,} \\ d_j &= \text{due date for job } j. \end{aligned}$$

Since both formulas have the $TRPT$ term in the numerator jobs with a long lead time will get a lower CR number most of the time.

The purpose with the modifications was to make the functions take the lead times into account when calculating the priority numbers. The reason for this is that jobs with longer lead times probably are less sensitive to

disturbance resulting in delays, compared to jobs with shorter lead times. Also, jobs with shorter lead times do not affect the multitask cell as much as jobs with longer lead times. To make the CR-function take the lead times into account and prioritize jobs with shorter lead times should have a positive effect on the total lateness. The following functions and the existing one were tested to see the affect of incorporating the lead times:

$$CR' = CR \cdot (\textit{total process time}); \tag{4.2}$$

$$CR' = CR \cdot CR \cdot (\textit{total process time}); \tag{4.3}$$

$$CR' = \frac{CR}{\textit{total process time}}, \tag{4.4}$$

where CR' is the new value and CR is the old function value from the existing priority function. "*Total process time*" for a job is calculated using all operations, disregarding which resource that can process the operations, in the mathematical model. In the simulation environment at Volvo Aero only operations that can be processed on any of the machines, the multitask machines or the deburring machine, are used to calculated the "*total process time*". That different methods are used to calculate "*total process time*" should not make a difference since longer jobs have longer operations, and in both cases "*total process time*" could most likely be replaced with expected lead time without any significant difference. The function in equation (4.4) was tested to see what happens when jobs with longer lead times are prioritized even more than before. Another reason for the modification is that it opens up a window for short jobs to be processed before already late jobs to avoid the ones on time to be late too.

Except for the existing function and the modifications described above, a new type of function to calculate the CR-value was also implemented and tested. This function was tested in two different versions:

$$CR = d_j - (\textit{current time}) - (TRPT + (\textit{expected waiting time between operations})) \tag{4.5}$$

$$CR = d_j - (\textit{current time}) - (TRPT + (\textit{expected waiting time between operations}))) \cdot (\textit{total process time}) \tag{4.6}$$

Function (4.5) prioritizes the jobs according to the difference between available time and the time needed to finish the jobs. (Note that the CR-value is negative if the job cannot finish on time.) In function (4.6) the lead times are used as a weight to prioritize shorter jobs somewhat more than longer jobs.

The function 4.6 did not work as well as first expected because, when a job is late the term $(current\ time) - d_j$ is negative and jobs that are late will always be prioritized prior to jobs that are on time. Also, the function prioritizes jobs with longer lead times before jobs with shorter lead times when both are late because of the negativity and the following multiplication in the function.

When functions (4.5)–(4.6) were implemented at Volvo Aero there was a slight misunderstanding resulting in the following functions:

$$CR = d_j - (TRPT + (expected\ waiting\ time\ between\ operations)), \tag{4.7}$$

$$CR = d_j - (TRPT + (expected\ waiting\ time\ between\ operations)) \cdot (total\ process\ time). \tag{4.8}$$

These functions performed surprisingly well in the simulation environment at Volvo Aero, and was therefore tested further and included in the report.

4.2 First trial: Time window optimization

The term *time window optimization* refers to optimizing in discrete time steps, which was the first idea for a heuristic. To be able to optimize in discrete time steps it is necessary to have a counter that keeps track of the time and a parameter that specifies how long an interval should be. The *time window* is defined as the current time plus the interval parameter. When the scheduling is done only jobs that are released in or prior to the current time window are considered. All operations that are scheduled to start in the current time window are fixed and cannot be changed in the future iterations; the operations that are scheduled to start after the time window are not fixed and can be re-scheduled in future iterations. In every iteration a priority function creates a feasible schedule that serves as a lower bound to the optimizer to narrow down the search.

The time window optimization method is the one that resembles dynamic scheduling the most of the methods that was implemented, since it only schedules jobs that are released in a near future. What happens is that the method divides the overall dynamic scheduling problem into several small static scheduling problems that are only considering a fragment of the total number of jobs.

In Figure 4.1 the method is illustrated for a general problem with four jobs and five machines. (It is not the multitask cell.) Every box in the figure

represent an operation and the letters inside the box represent the machine that will process the operation. Filled boxes are the operations that have been fixed (start time of the operations are in or prior to the current time window). The interval parameter used in the example is ten units so the time windows are $[0-10)$, $[10-20)$ and $[20-30)$ and so on. In the first time window only *job* 1 and *job* 2 are available for scheduling. The first operation of both jobs and the second operation of the first job are scheduled to start in the first time window and is therefore fixed. In the second time window *job* 3 is also available for scheduling. This forces some changes to the schedule generated in the first iterations; operation three of *job* 2 is scheduled to be processed on a different machine, for example. The procedure is repeated until there are no jobs left to schedule. If no new jobs are released in a time window no re-scheduling is done since the last generated schedule is the best one still. Therefore, the time window $[20-30)$ is the last time window where any actual scheduling is done in the example.

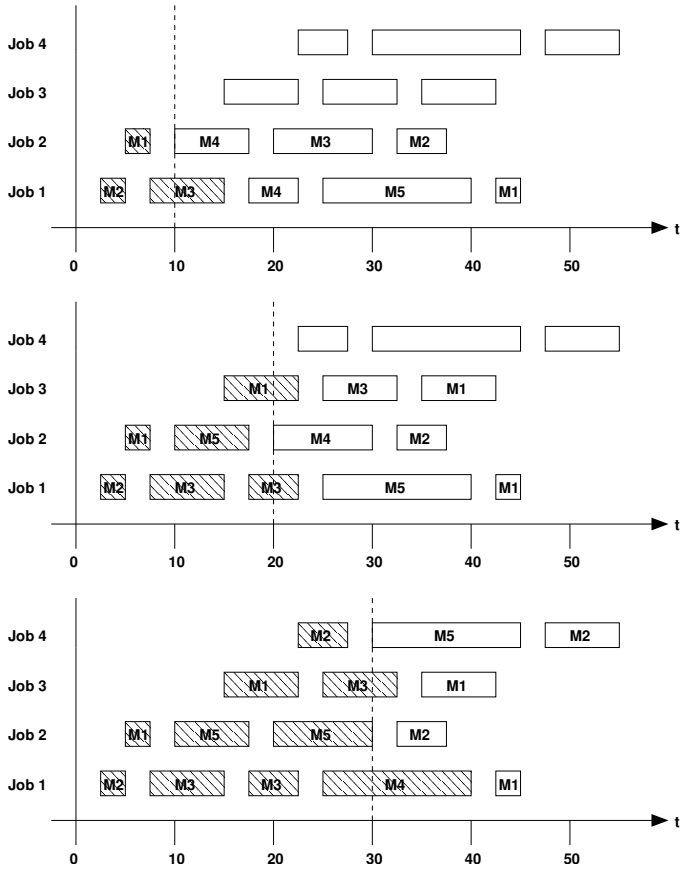


Figure 4.1: An illustration of the time window optimization method.

4.2.1 Additions to the model

Some additions to the mathematical model were needed to get the time window optimization method working. The additions to the model are a time parameter, T, that keeps track of the current time and an interval parameter, Δt, that defines how long the time windows are. Some changes to the constraints and objective functions are also necessary to make sure that only jobs that are released prior to time $T + \Delta t$ are scheduled in each iteration. The new modified constraints and objective functions are:

$$\sum_{k \in \mathcal{K}} z_{jik} = 1, \quad r_j < T + \Delta t, \quad j \in \mathcal{J}, \quad i \in \mathcal{N}_j, \tag{4.9}$$

$$z_{jik} \leq a_{jik}, \quad r_j < T + \Delta t, \quad j \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad k \in \mathcal{K}, \tag{4.10}$$

$$y_{jipqk} + y_{pqjik} \leq a_{jik}, \quad \begin{cases} r_j < T + \Delta t, \quad j, p \in \mathcal{J}, \quad i \in \mathcal{N}_j, \\ q \in \mathcal{N}_p, \quad k \in \mathcal{K}, \quad (j,i) \neq (p,q), \end{cases} \tag{4.11}$$

$$y_{jipqk} + y_{jipqk} + 1 \leq (z_{jik} + z_{pkq}), \quad \begin{cases} r_j < T + \Delta t, \quad j, p \in \mathcal{J}, \\ i \in \mathcal{N}_j, \quad q \in \mathcal{N}_p, \\ k \in \mathcal{K}, \quad (j,i) \neq (p,q), \end{cases} \tag{4.12}$$

$$\begin{aligned} M(1 - y_{jipqk}) + M(1 - z_{pqk}) + x_{pqk} \geq x_{jik} + p_{ji} - M(1 - z_{jik}), \\ j, p \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad q \in \mathcal{N}_p, \quad (j,i) \neq (p,q), \\ k \in \mathcal{K}, \quad r_j < T + \Delta t, \quad r_p < T + \Delta t, \end{aligned} \tag{4.13}$$

$$\begin{aligned} M \cdot y_{jipqk} + M(1 - z_{jik}) + x_{jik} \geq x_{pqk} + p_{pq} - M(1 - z_{pqk}), \\ j, p \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad q \in \mathcal{N}_p, \quad (j,i) \neq (p,q), \\ k \in \mathcal{K}, \quad r_j < T + \Delta t, \quad r_p < T + \Delta t, \end{aligned} \tag{4.14}$$

$$\begin{aligned} M(1 - z_{j,i+1,k}) + x_{jik} \geq x_{jik} + p_{ji} + w - M(1 - z_{jil}), \\ j \in \mathcal{J}, \quad i \in \mathcal{N}_j \setminus \{n_j\}, \quad k, l \in \mathcal{K}, \quad r_j < T + \Delta t, \end{aligned} \tag{4.15}$$

$$\text{minimize} \sum_{j \in \mathcal{J}} L_j, \qquad r_j < T + \Delta t, \tag{4.16}$$

$$\text{minimize} \sum_{j \in \mathcal{J}} (C_j + L_j), \qquad r_j < T + \Delta t. \tag{4.17}$$

In each of the iterations constraints of the form:

$$z_{jik} = 1, \qquad j \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad k \in \mathcal{K} \tag{4.18}$$

$$y_{jipqk} = 1, \qquad j, p \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad q \in \mathcal{N}_p, \quad (j,i) \neq (p,q), \tag{4.19}$$

$$x_{jik} = x_{jik}, \qquad j \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad k \in \mathcal{K}, \tag{4.20}$$

are generated. The constraints are only generated for the operations that are scheduled to start in the current time window, $x_{jik} < T + \Delta t$, and states that operation i of job j will be processed on machine k at time x_{jik}. These constraints can not be removed once they are generated.

4.3 Second trial: Langrangian relaxation and subgradient optimization

To better explain how this method works there are first two sections describing the basics of Lagrangian theory (Section 4.3.1) and subgradient optimization (Section 4.3.2). In Section 4.3.3 the actual method using Lagrangian relaxation and subgradient optimization is presented.

4.3.1 Lagrangian theory

The Lagrangian approach is very useful for obtaining lower bounds for optimization problems. Out exposition of Lagrangian theory is taken from [Ber98]. Lagrangian relaxation is best explained with an example like the following optimization problem:

$$\begin{aligned}
&\text{minimize} && f(x) = a'x, && && (4.21a)\\
&\text{subject to} && b_s'x \leq e_s, && s = 1, \ldots, q && (4.21b)\\
&&& c_t'x \leq d_t, && t = 1, \ldots, r && (4.21c)\\
&&& x_i = \{0, 1\}, && i = 1, \ldots, n && (4.21d)
\end{aligned}$$

which has a linear cost function, linear side constraints, and binary constraints on the variables x_i. Lagrangian relaxation basically consists of two steps. First some constraints are removed to expand the region of feasible solutions making it easier to solve; secondly the corresponding terms are added to the objective function so the objective value will be affected in a negative way if the constraints are violated. If the constraint specified by equation (4.21c) is the complicating constraint in the illustrating example it makes the most sense to remove it from the original problem. The Lagrangian function is formed when the terms $\mu_t(a'x - d_t)$ are added to the cost function:

$$L(x, \mu) = a'x + \sum_{t=1}^{r} \mu_t(c_t'x - d_t), \tag{4.22}$$

where $\mu_t \geq 0$ are the Lagrange multipliers. A Lagrange multiplier can be seen as a unit penalty that is added to the objective function for each unit

that $c_t'x$ exceeds d_t. The set of remaining constraints can be formulated as:

$$S = \{x : x_i = \{0,1\}, \quad b_s'x \leq e_s, \quad i = 1,\ldots,n, \quad s = 1,\ldots,q\},$$

which still has to be considered. Suppose that f^* is the objective value of any optimal solution to (4.21). Next note that

$$\begin{aligned}
\min_{x\in S} L(x,\mu) &= \min_{x\in S}\left\{a'x + \sum_{t=1}^{r}\mu_t(c_t'x - d_t)\right\} && (4.23)\\
&\leq \min_{\{x\in S | c_t'x\leq d_t, t=1,\ldots,r\}}\left\{a'x + \sum_{t=1}^{r}\mu_t(c_t'x - d_t)\right\} && (4.24)\\
&\leq \min_{\{x\in S | c_t'x\leq d_t, t=1,\ldots,r\}} a'x && (4.25)\\
&= f^*. && (4.26)
\end{aligned}$$

If the dual function is defined as:

$$q(\mu) = \min_{x\in S} L(x,\mu), \tag{4.27}$$

it is clear that $q(\mu) \leq f^*$, for all $\mu \geq \mathbf{0}^n$, that is, Largrange relaxation provides lower bounds on f^*. The Lagrangian dual problem can be formulated as:

$$\begin{aligned}
&\text{maximize} \quad q(\mu), && (4.28a)\\
&\text{subject to} \quad \mu_t \geq 0, \qquad t = 1,\ldots,r., && (4.28b)
\end{aligned}$$

This yield the best lower bound to the optimal solution of the original problem that is possible to achieve with the Lagrangian function.

4.3.2 The subgradient method

The dual function q in (4.27) is, for a fixed $x \in S$, a linear function. Thus, because the set S is finite, the dual function q is the minimum of a finite number of linear functions of μ; this implies that q is piecewise linear. Because of the piecewise linear structure of the dual function (4.27) it is not differentiable in some points; hence steepest decent methods cannot be used. The subgradient method is a simple algorithm to minimize nondifferentiable concave functions. From [Roc70] (page 214) the characterization of a subgradient g to q at a given μ is as follows:

$$q(\nu) \leq q(\mu) + (\nu - \mu)'g, \quad \forall \nu \in \Re^n. \tag{4.29}$$

Let $x_\mu \in S$ minimize the Lagrangian $L(x, \mu)$ for a given value of μ. The vector at μ with elements

$$g_t(x_\mu) = c_t' x_\mu - d_t, \quad t = 1, \ldots, r,$$

is a subgradient to the dual function q defined in (4.27). To show this, note that for all $\nu \in \Re^n$,

$$\begin{aligned} q(\nu) &= \min_{x \in S} L(x, \nu) \\ &\leq L(x_\mu, \nu) \\ &= a' x_\mu + \nu' g(x_\mu) \\ &= a' x_\mu + \mu' g(x_\mu) + (\nu - \mu)' g(x_\mu) \\ &= q(\mu) + (\nu - \mu)' g(x_\mu). \end{aligned}$$

This shows that a vector that is formed by constraint function g at x_μ is a subgradient of q at μ.

The next step is to describe the actual subgradient method. The method consists of the iteration

$$\mu_t^{k+1} = \max\{0, \mu_t^k + s^k g_t^k\}, \quad t = 1, \ldots, r, \quad k = 1, \ldots, \tag{4.30}$$

where k is the iteration number, g_t^k is element number t of the subgradient g of q at μ^k, and $s^k > 0$ is the step size. The subgradient is easiest calculated through minimizing $L(x, \mu^k)$ at μ^k, let the minimizing x be x_{μ^k} and set

$$g^k = g(x_{\mu^k})$$

where $g(x)$ is the vector defined by the relaxed constraints. When using the subgradient method the dual cost q may not improve at all iterations. The convergence of the method is based on reducing the distance from the current solution to the optimal solution, if sufficiently small step sizes s^k are used. Figure 4.2 illustrates this fact.

To guarantee convergence it is common to select the following step size formula to update the step size s^k:

$$s^k = \frac{\alpha^k (q^k - q(\mu^k))}{||g^k||^2} \tag{4.31}$$

where q^k is an approximation to the optimal dual cost and $0 < \alpha^k < 2$. The parameter α in the step size update formula can be chosen in several different ways. For this thesis work two different methods for updating the α parameter have been tested. In the first method α^0 was set to a value in the

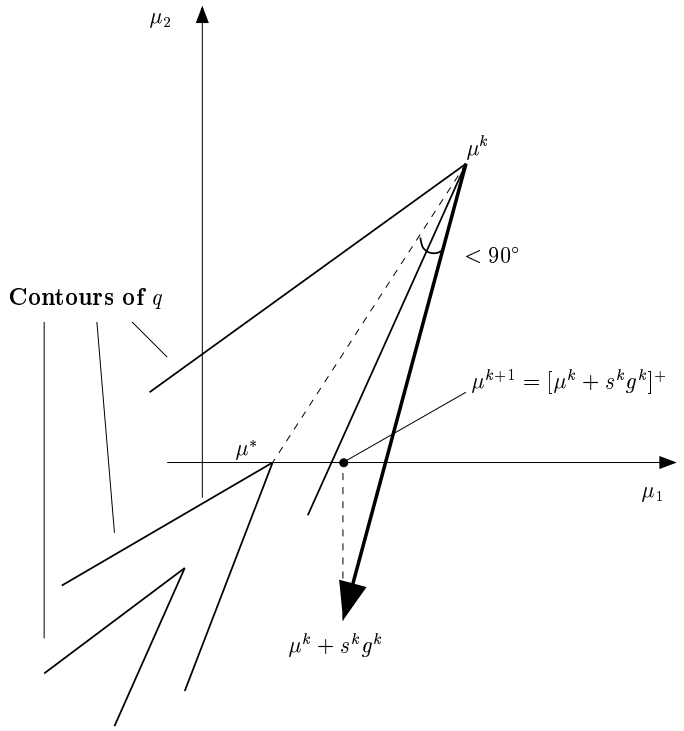


Figure 4.2: The figure illustrates when the dual cost might not improve in an iteration.

interval $(0, 2)$ and decreased with a factor of two every few iterations. (During the testing phase the number of iterations as well as the start value of α was altered to test different values.) In the second approach an adaptive formula was tested to speed up the convergence. The formula used for updating the α^k is:

$$\alpha^{k+1} = \begin{cases} \frac{1}{2}\alpha^k, & \text{if } \overline{q} - \underline{q} > 0.01 \cdot \underline{q}, \\ \frac{3}{2}\alpha^k, & \text{if } \overline{q} - \underline{q} < 0.001 \cdot \underline{q}, \\ \alpha^k, & \text{otherwise,} \end{cases} \qquad k = 1, 2, \ldots, \tag{4.32}$$

where

$$\underline{q} = \min_{r=k-p+1,\ldots,k} q(\mu^r) \quad \text{and} \quad \overline{q} = \max_{r=k-p+1,\ldots,k} q(\mu^r). \tag{4.33}$$

This means that in every iteration the best and worst lower bound during the last p iterations are compared and α is updated according to the formula. For more information about this updating method of α see [Jun00].

4.3.3 Lagrangian relaxation of the multitask model

The idea of this method was to generate dual solutions that can (hopefully) be transformed to feasible solutions rather fast. One way this transformation can be done is described in Section 4.4.

The constraints that were relaxed are the constraints in equations (3.6)–(3.7). The most complicated constraints were relaxed since the purpose of the relaxation was to speed up the optimization. Before redefining the new cost function a vector consisting of the relaxed constraints is defined:

$$g^1_{jipqk} = x_{ijk} + p_{ji} - M(1 - z_{jik}) - (M(1 - y_{jipqk}) + M(1 - z_{pqk}) + x_{pqk}) \\ j, p \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad q \in \mathcal{N}_p, \quad (j,i) \neq (p,q), \quad k \in \mathcal{K}; \tag{4.34}$$

$$g^2_{jipqk} = x_{pqk} + p_{pq} - M(1 - z_{pqk}) - (M \cdot y_{jipqk} + M(1 - z_{jik}) + x_{jik}) \\ j, p \in \mathcal{J}, \quad i \in \mathcal{N}_j, \quad q \in \mathcal{N}_p, \quad (j,i) \neq (p,q), \quad k \in \mathcal{K}. \tag{4.35}$$

Let g be the combined vector of g^1_{jipqk} and g^2_{jipqk}. Even though there are several indices it do not complicate the notation since they can be seen as one composed index for one vector. This gives the following dual cost function:

$$q(\mu) = \min L(x, \mu) = \min \sum_{j \in \mathcal{J}} L_j + \mu^{\mathrm{T}} g(x). \tag{4.36}$$

Note that L_j denotes *lateness* while $L(x, \mu)$ denotes *the Lagrange function.*

4.4 Third trial: Fixing variables

The idea of the fixing variables method is to transform the original problem to a new problem through fixing some of the variables. Before fixing any variables a feasible solution is needed, or some variables that are correctly set. Which variables are appropriate to fix? It does not make much sense to fix the time variables, x_{jik}, since if the time variables are fixed the schedule is also fixed. Fixing the order variables y_{jipqk} has the same result since they determine where and in which order the operations should be done. The most reasonable variables to fix are therefore the allocation variables z_{jik}. Fixing the allocation variables transform the problem from an MPM job shop problem to a regular job shop problem where all that is left to do is to decide the order of the operations on the machines. When fixing the z_{jik} it is impossible to guarantee the optimal schedule since the allocation might not be optimal.

As mentioned earlier this method has to be used in conjunction with some other method or priority function. The method consists of three steps:

1. Create a feasible allocation solution with a priority function, that is, a feasible z_{jik} solution.

2. Fix the allocation variables z_{jik}.

3. Optimize the new problem with the z_{jik} variables fixed.

4.5 Fourth trial: Local search

The idea behind local search was to start with an existing feasible solution and modify the solution in small steps towards a better one. The number of steps taken towards the final solution could either be decided by some pre-determined conditions being fulfilled or for some pre-defined number of iterations. In all iterations a new feasible solution (that might differ depending on the method used) is found. To describe these possible moves between feasible solutions a neighborhood structure $N : S \rightarrow 2^S$, where S is the set of feasible solutions, needs to be defined. The set $N(s)$ is the set of neighbors to a feasible solution s and where the "best" solution is searched for. To search for the best neighbor in every iteration might result in finding a local minimum instead of a global one.

The local search function is supposed to be used in conjunction with the method described in Section 4.4 and a modified version of the time window method (Section 4.2). Since the solution found with the method in Section 4.4 is optimal given the machine allocation, the only search available is to search for a swap between operations on different machines trying to create a better solution. The diagram in Figure 4.3 illustrates the flow of the method.

4.5.1 Neighborhood structure for the multitask cell

The definition of a good neighborhood structure is the most crucial part of local search methods. If the neighborhood is too small a good solution might be difficult to reach; if it is too large the execution time might be too long. Defining a good neighborhood structure took more effort than expected. The variables move in too many dimensions to make a simple move from one solution to another. The jobs do not have the same release and due dates, which makes it hard to find an easy way to know which operations that are possible to swap. Also, the operations are of various lengths which makes it hard to calculate the cost of swapping two jobs. (If you swap two operations

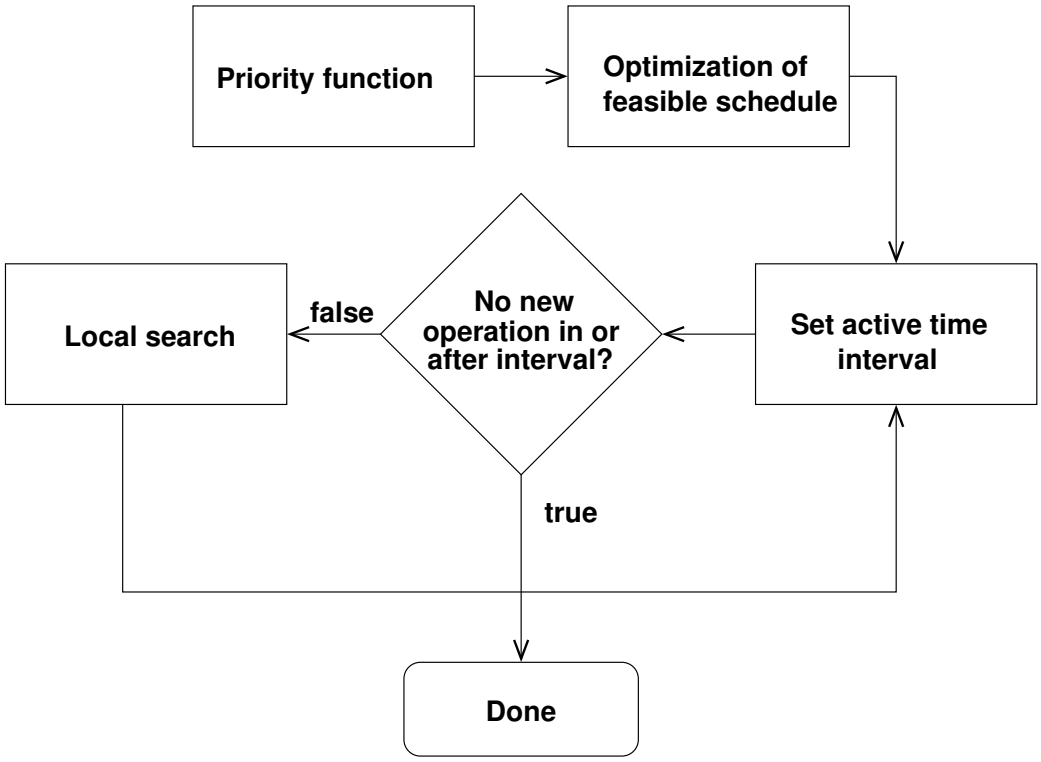


Figure 4.3: Diagram over the flow in the local search method.

you have to make a new optimization that might re-schedule operations on machines that were not one of the machines used in the swap.)

The neighborhood structure implemented and tested is defined as follows.

1. Find the longest operation that belongs to the latest job in the current time interval.

2. Find the corresponding operation for the earliest job on a different machine.

3. Swap and re-calculate the cost.

4. Keep track of the new cost and solution if better than previous one.

5. Repeat from 1 but exclude jobs that already have been considered.

The purpose with this strategy was to move hours from one job that is early to one that is late and hopefully make them both be on time.

4.6 Fifth trial: Relaxing and rounding

This was the last method implemented and the one on which the least time was spent. The idea was to relax the integrality constraints and make the problem a regular linear program, which is much easier to solve. The relaxed solution is not a feasible solution and it is in fact impossible to know if it will turn out to be a good solution after rounding. However, the relaxed solution creates a lower bound for the original problem. The method consists of three steps:

1. Optimize the relaxed problem.
2. Round the integer variable to a feasible solution.
3. Optimize the starting times.

In step two all integer variables are rounded to 0 or 1 creating a feasible machine allocation and order between the operations. This step is more difficult than it first appears since an operation i of job j can have its allocation variables z_{jik} larger than 0 for more than one machine k; the question is how to choose between the variables set for one operation. When implementing this method the largest z_{jik}, for a fixed j and i, was set to 1 and the other z_{jik} variables was set to 0. The next problem is to set the order between the jobs. Since the order variables are five dimensional (j, i, p, q, k) they are far from 1, which makes it difficult to set them in an optimal way. This is how the order variables were set:

- First set the allocation variables z_{jik} for all operations according to the previous description.
- For all operations (j, i) and (p, q) where $(j, i) \neq (p, q)$ on all machines, check if z_{jik} is set for both.
  - If $y_{jipqk} > y_{pqjik}$ set $y_{jipqk} = 1$ and $y_{pqjik} = 0$
  - If $y_{jipqk} < y_{pqjik}$ set $y_{jipqk} = 0$ and $y_{pqjik} = 1$
  - If the order variables are the same and larger than 0 the jobs with the earliest release date is scheduled first.

The order variables as well as the allocation variables were set because it would take too long to solve the problem with just the allocation variables set. When setting the order variables it seemed reasonable to compare the order variables operation by operation and set the variable with the highest value.

Chapter 5

Implementation

The choice of implementation language was obvious since there was only one modeling language available. All modeling has been implemented using the mathematical programming language *AMPL*. The optimization has been done with the optimization tool *CPLEX*, which is compatible with AMPL. To create Gantt charts and data from the results some modules were implemented using *Matlab*.

5.1 AMPL model

The implementation of the model was relatively straightforward since it is rather uncomplicated to implement a mathematical model in AMPL. To make it easier to input data to the model the job set J was split up into several sets. One of the sets has all the product types that are available, and for each product type there is one set telling how many jobs of that product there are. In mathematical notation this is written as:

$$h \in \mathcal{S}, \quad j \in \mathcal{J}_h, \quad i \in \mathcal{N}_h,$$

where $\mathcal{S}$ is the set of products, $\mathcal{J}_h$ is the set of jobs of product type h, and $\mathcal{N}_h$ is the set of operations for product type h. This simplifies the model a lot since the processing times only need to be specified one time per product instead of one time per job. The AMPL implementation of the model can be seen in Appendix A.1 and an example data input in Appendix A.2.

5.2 Implementing the functions

Implementing the priority functions was more complicated than implementing the model. One of the reasons is that it is hard to keep track of the

status of the machines and jobs in AMPL, which would have been easier in an object oriented programming language.

Even though AMPL is not an object oriented language the functions were divided into several parts to make it easier to test, debug and maintain. The different parts are:

- one part that finds the next available machine,
- one part that finds the next available operation to the machine found,
- one part containing the different priority functions,
- one part that sets the variables for an operation if a matching between a machine and an operation is found,
- one part that sets the time when the machine can be considered again if no matching is found,
- one part that sets the order variables,
- one part that creates the output for future calculation and presentation, and
- one part that controls the communication flow between the different parts.

Most of the implementation was straightforward; for example, if a matching between a machine and an operation is found, all that needs to be done is to set the variables that defines the machine allocation, the time when the operation should start and the time when the machine is available again. However, the implementation of the part that specifies when a machine can be considered again if no operation was found at the current time made the implementation more complicated. The first solution to this problem was quite naive: the new time for when a machine is available is set to the minimum of the minimum release date, and the minimum of the other machines times for when they are available, larger than the current time. This did not solve the problem since an operation can be available earlier if it belongs to a job that has recently finished one operation, and then the transportation time between two operations has to be taken into account. The final solution to this problem is as follows:

- let the current time be called t, and the current machine be called k;
- let the new variables $checked_k$ take the value 1 for every machine k that has been checked at the current time t;

- loop over all machines and find the smallest value of $(finish\ time) + (waiting\ time)(1 - checked_k)$ among the ones that are larger than t and let this be the new time that the machine can be considered, t';
- let t' take the value of the smallest release date if t' is larger than the smallest release date that is larger than t;
- let the new available time for the machine be t';
- set $checked_k = 1$.

The waiting time in step three is the time it takes before a job can be processed again after it has finished an operation.

5.3 Matlab module

The Matlab module was created to make it easier to interpret the results. The module can create Gantt charts and diagrams of the machine utilization. Example Gantt charts can be seen in Section 3.4. A Matlab function for creating Gantt charts was implemented since Matlab does not have a built in function for Gantt charts. The Gantt chart function was implemented using the built in box-plots and stacking the boxes on top of each other. There is one desirable feature that could not be implemented, which is to color code the bars according to the machines. Instead the bars are color coded according to the operation numbers, since there was no option available for specifying the color of every single bar. To be able to follow a job and see which machine it had been processed on it was necessary to produce an output file containing tables over machine allocations. This output file was later extended to contain more than just the machine allocation. A file created by the Matlab module contains:

- one table that summarizes to which machine each operation is allocated,
- one table that summarizes when every operation starts,
- one table that summarizes release date, due date, start time, finish time, total time in the multitask cell and how the total time differed from the expected lead time for each job,
- one table that summarizes how many of each product type that was processed, average time in the multitask cell for each product type, minimum and maximum times in the multitask cell for each product type and how that time differed from the expected lead time,

- one table that summarizes how many operations that has been processed on each machine, when the first operation started on every machine, when the last operation ended on every machine and how each machine was utilized between the first and last operation, and

- overall information like total number of late jobs, total lateness and relative number of late jobs (*total number of late jobs/total number of jobs*).

5.4 Implementation problems

There were several minor and one major problem during the implementation phase. Most of the minor problems were implementation problems while the major problem was caused by some error in the software.

The implementation errors were mostly index errors that arose from AMPL trying to access elements in arrays or matrices that did not exist. To fix those errors was rather easy; the problem was to find them. These problems were most common when dealing with the priority functions since when using CPLEX to schedule the jobs CPLEX takes care of the indexing as long as it is modelled correctly.

The hardest problem, which was never resolved, was found when working with the time window optimization method. An error message from AMPL indicated that there could be a numerical error. Also, the error message indicated that it might help changing the AMPL-parameters `$presolve_eps` or `$presolve_inteps`. To find the right values for these parameters was difficult and that is why only one relevant result was obtained from the time window method. The reason why this error appears in the time window method is probably because it is an iterative method. At every iteration some variables were rounded off to a (according to AMPL) suitable value and the rounded values made it impossible for CPLEX to find a feasible solution in the next iteration. This was probably the error where most time was spent without resolving the error. The attempt to resolve the error was put aside when the error was present in the model where the integrality constraints were relaxed, and it turned out that it would take too long using the time window method for larger input instances anyway.

There was a similar problem for the priority functions, but it was easier to resolve. AMPL tried to compare a variable to a fixed value (0 for example) and the result from the comparison claimed that they differed even though they theoretically should have the same value. This problem was easier to resolve because of the knowledge of the input data. The data specified did

only use values with at most two decimals and the incorrect value differed in at least the tenth decimal. To resolve this problem something smaller than 0.009 needed to be added to the right-hand side of $<$ and $\leq$ comparisons and to the left-hand side for $>$ and $\geq$ comparisons. The impact of this error was not as extensive as the previous one; a solution was still generated but some operations were scheduled later than they should have been. The error was not revealed at once since a solution was generated and the solution seemed to be correct.

Chapter 6

Results

The chapter are divided into different parts to reflect the sections in Chapter 4. The methods that were able to produce a somewhat useful schedule are compared to some priority functions and/or other heuristics. Before the actual results there is a section describing the testing environments.

6.1 Testing environment

To verify that the methods worked, only small input data was used during the implementation phase. Some of the heuristics never passed this step for different reasons. Being too slow or not being able to produce satisfying results were the main reasons. The priority functions were tested with relatively large instances in the mathematical model as well as in the simulation environment at Volvo Aero. Except from the fact that the simulator is able to use larger instances of input, is also takes the number of fixtures and tools etc. into account, which makes the result more realistic and reliable.

6.2 Priority functions

All the results for the priority functions were generated in the simulation environment at Volvo Aero to get more reliable results. The tests were simulated using three different scenarios:

1. No variation in the input for the details, that is, the first job of all product types started at the same time and the following details arrived in a fixed interval that is product type specific.

2. A small variation in the input for the first details of all product types

and the interval between the following details followed a normal distribution.

3. Same as 2 but with larger variations for all products as well as for the interval for the following details of each product type.

The product specific data is presented in Table 6.1 and the inputs are summarized in Table 6.2. The eight priority functions presented in Section 4.1 were tested in the simulation environment. The following list associates the names of the functions in the figures with their corresponding functions in Section 4.1:

prio 1 the existing priority function, function (4.1);

prio 2 the existing priority function with a slight modification, function (4.2);

prio 3 the existing priority function with a different modification, function (4.3);

prio 4 the priority function where the result is expected to be worse than with the existing function, function (4.4);

prio 5 the priority function that was based on a slight misunderstanding, function (4.7);

prio 6 same as **prio 5** but weighted, function (4.8);

prio 7 the priority function that was intended to be tested instead of **prio 5**, function 4.5;

prio 8 same as **prio 7** but weighted, function (4.6).

The different scenarios were run over one year in the simulation environment and the scenarios that had a input variation were run five times to get a good average. (If there is no variation in the input, all five years are exactly the same, that is why it is sufficient to run scenario 1 one time.) The multitask machines in the multitask cell were occupied approximately 87% of the time, which is considered to be a heavy load.

The results in Figures 6.1–6.3 show the total and average lateness and the total number of late jobs.

Products	prdQ	prdR	prdS
Operations / product	11	5	11
Interval (hrs.)	72.5	36	36
Expected lead time (hrs.)	68.3	14.5	61.2
Number of products / year	69	139	139

Products	prdT	prdU	prdV
Operations / product	3	5	7
Interval (hrs.)	44.7	44.7	44.7
Expected lead time (hrs.)	5.4	15.5	17.1
Number of products / year	112	112	112

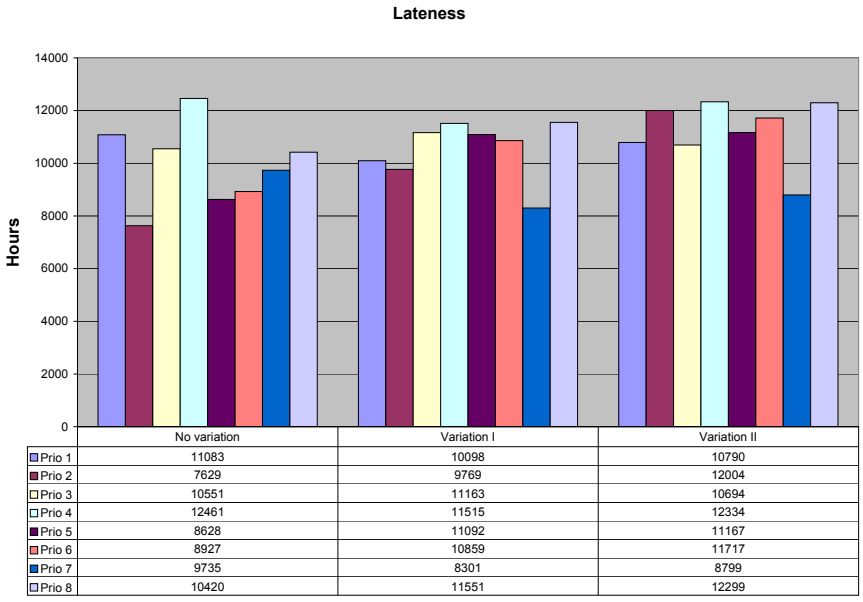
Products	prdW	prdX	prdY
Operations / product	11	4	3
Interval (hrs.)	44.7	44.7	143
Expected lead time (hrs.)	66.2	6.0	5.7
Number of products / year	112	112	35

Products	prdZ
Operations / product	13
Interval (hrs.)	13.1
Expected lead time (hrs.)	46.1
Number of products / year	381

Table 6.1: The product specific input for the tests that were simulated at Volvo Aero.

	First arrival		
	No variation	Variation I	Variation II
prdQ	0.5	0.5	55
prdR	0.5	5	15
prdS	0.5	10	30
prdT	0.5	15	0.5
prdU	0.5	20	9
prdV	0.5	25	18
prdW	0.5	30	27
prdX	0.5	35	36
prdY	0.5	40	0.5
prdZ	0.5	45	7

Table 6.2: The arrival dates for the first detail of each product.

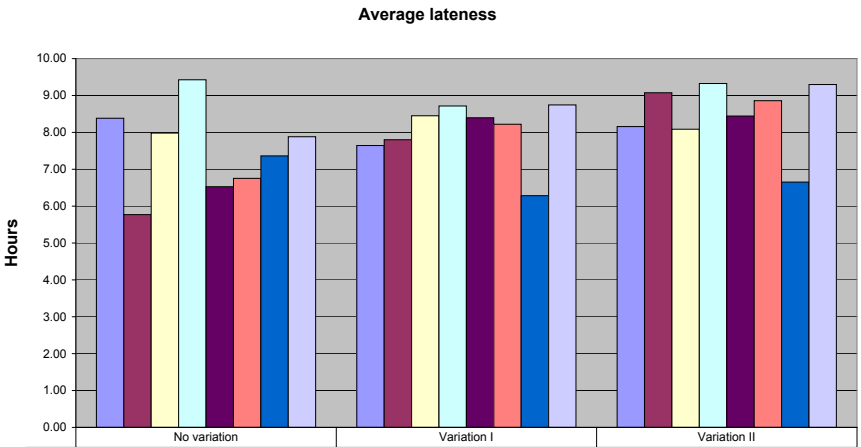


	No variation	Variation I	Variation II
Prio 1	11083	10098	10790
Prio 2	7629	9769	12004
Prio 3	10551	11163	10694
Prio 4	12461	11515	12334
Prio 5	8628	11092	11167
Prio 6	8927	10859	11717
Prio 7	9735	8301	8799
Prio 8	10420	11551	12299

Figure 6.1: Total lateness for the priority functions.

	No variation	Variation I	Variation II
Prio 1	1110	875	898
Prio 2	961	885	921
Prio 3	1084	894	892
Prio 4	1141	911	925
Prio 5	1018	892	892
Prio 6	995	880	889
Prio 7	768	697	701
Prio 8	1024	907	913

Figure 6.2: The number of late jobs for the priority functions. The total number of products during this time is 1323.



	No variation	Variation I	Variation II
Prio 1	8.38	7.64	8.16
Prio 2	5.77	7.80	9.08
Prio 3	7.98	8.45	8.09
Prio 4	9.43	8.72	9.33
Prio 5	6.53	8.40	8.44
Prio 6	6.75	8.22	8.86
Prio 7	7.36	6.28	6.65
Prio 8	7.88	8.74	9.30

Figure 6.3: The average lateness for the priority functions.

6.3 First trial: Time window optimization

The time window optimization method was one of the methods that managed to produce a suboptimal schedule from a decent size of input data. Why the schedule is only suboptimal is mainly because of the iterative procedure. To speed up the optimization even more, a variable for adjusting the tolerance on the gap between the best integer solution found so far, and of the best possible solution remaining was set to allow a 20% difference between the two. The consequence of this is that the each of the solutions in every iteration might have an objective value that is 20% higher than the optimal value. The input data used is specified in Table 6.3. The lead time for product type `prdY` was set to 40 hours for the first product and to 60 hours for the remaining products to obtain more stress on the system. The optimized result could most likely be better than it is. This is mainly because:

- a 20% margin from the true optimal was allowed in all iterations;
- the objective function only minimizes the number of late hours, not the total number of hours; and
- since the method is an iterative method it is impossible to reach the absolute optimum. (The solution generated in one iteration might not provide the optimal solution in the long run.)

Figure 6.4 compares the result for the time window optimization method with some other methods for the single example that the time window optimization method managed to solve.

The time window optimization method also produced the error that was not resolved (see Section 5.4). This is why it has not been possible to run (larger) tests with the method. Except for that, the method did speed up the optimization part, but not enough. Creating a schedule from the input in Table 6.3 took several days, even though a priority function was used in all iterations to create an initial solution.

Products	prdS	prdT	prdU
Quantity	2	2	2
Operations / product	11	5	11
Time for arrivals (hrs. from 0)	0, 64	10, 94	5, 89
Expected lead time (hrs.)	62	13	54

Products	prdV	prdW	prdX
Quantity	3	3	3
Operations / product	5	5	
Time for arrivals (hrs. from 0)	0, 54, 110	5, 64, 120	20, 79, 125
Expected lead time (hrs.)	5	14	12

Products	prdY	prdZ
Quantity	3	2
Operations / product	11	4
Time for arrivals (hrs. from 0)	0, 39, 95	35, 119
Expected lead time (hrs.)	40 (60)	5

Table 6.3: Input for the example that was successfully optimized using the time window optimization method.

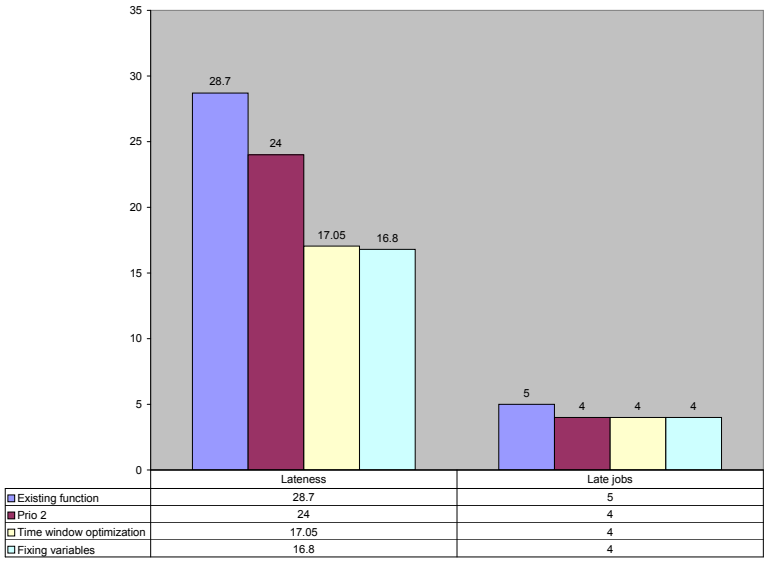


Figure 6.4: Result for the time window optimization method compared to the existing priority function, prio 2 [equation (4.2)] and the fixing variables method described in Section 4.4.

6.4 Second trial: Lagrangian relaxation

The Lagrangian relaxation method was probably the method where most time was spent, and where the least success was obtained. The tests for the Lagrangian relaxation method were made with a small example that only contained five jobs. The results never ended up in an actual schedule that gave any indication that the method might give a better schedule than the priority functions produced. Some random testing was done where the method in Section 4.4 was used to create a feasible solution at different iteration steps, but none of the solutions were better than the solution from the existing priority function. The time for optimizing the relaxed problem was also too long to be satisfying.

The tests with the adaptive step size rule did not improve the result significantly compared to the regular subgradient method.

6.5 Third trial: Fixing variables

The fixing variables method was the heuristic approach where most success was made, if a priority function was used to generate the first guess. Before the optimization part all operations for every job were allocated to a machine. The results presented here were tested with all the priority functions, except functions (4.7) and (4.4), in allocation algorithm. In a real environment the number of jobs that are allocated before the optimization part determines how good the schedule will be and how long it will take to generate it. The tests that were run used the input from Table 6.4. The results are shown in Figures 6.5–6.6, where the results are compared to the existing priority function, as well as to the priority function that had the best result. The functions used to generate the allocation for the method are equations (4.1), (4.2), (4.3), (4.5), (4.6) and (4.8), starting on page 28.

The method was also tested with the input from the small example in Section 3.4; the results for the example are shown in Figure 6.7 where the fixing variables method is compared to the optimal solution and the solution from the existing priority function.

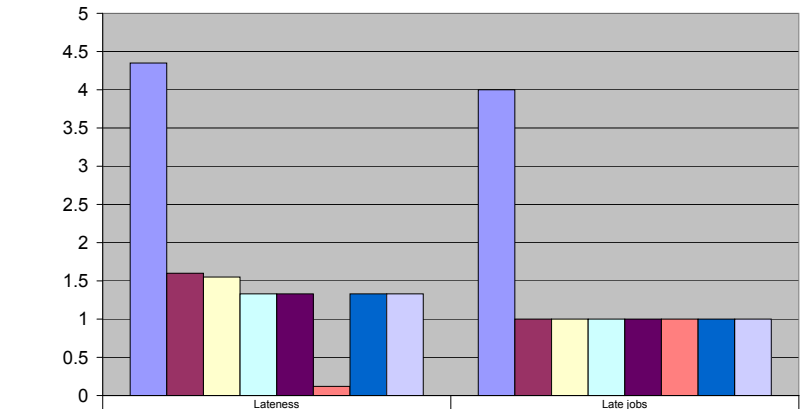
Products	prdQ	prdR	prdS
Quantity	3	3	3
Operations / product	11	5	11
First arrival (hrs. from 0)	0	10	5
Interval (hrs.)	48	48	48
Expected lead time (hrs.)	68.33	14.49	61.15

Products	prdT	prdU	prdV
Quantity	4	4	4
Operations / product	3	5	7
First arrival (hrs. from 0)	0	5	8
Interval (hrs.)	40	40	40
Expected lead time (hrs.)	5.35	15.5	17.05

Products	prdW	prdX	prdY
Quantity	4	3	2
Operations / product	11	4	3
First arrival (hrs. from 0)	0	12	7
Interval (hrs.)	40	48	30
Expected lead time (hrs.)	66.19	5.95	5.65

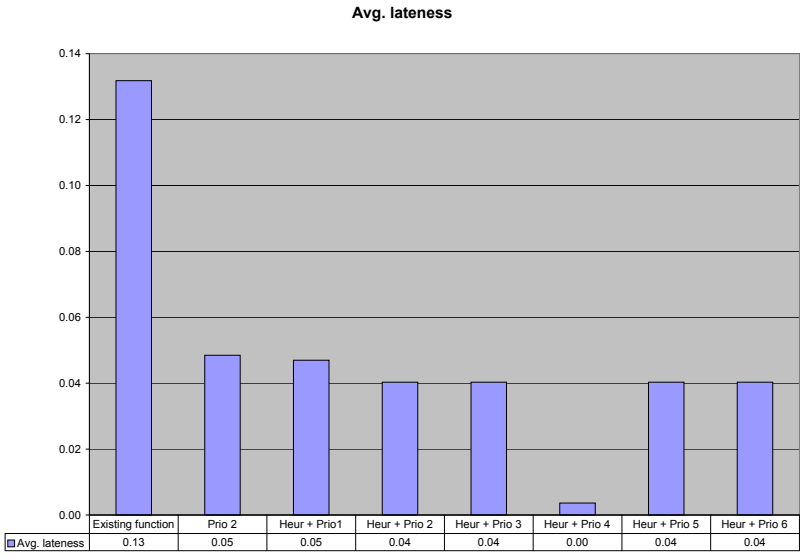
Products	prdZ
Quantity	3
Operations / product	13
First arrival (hrs. from 0)	15
Interval (hrs.)	48
Expected lead time (hrs.)	46.11

Table 6.4: The input for the test run for the fixing variables method.



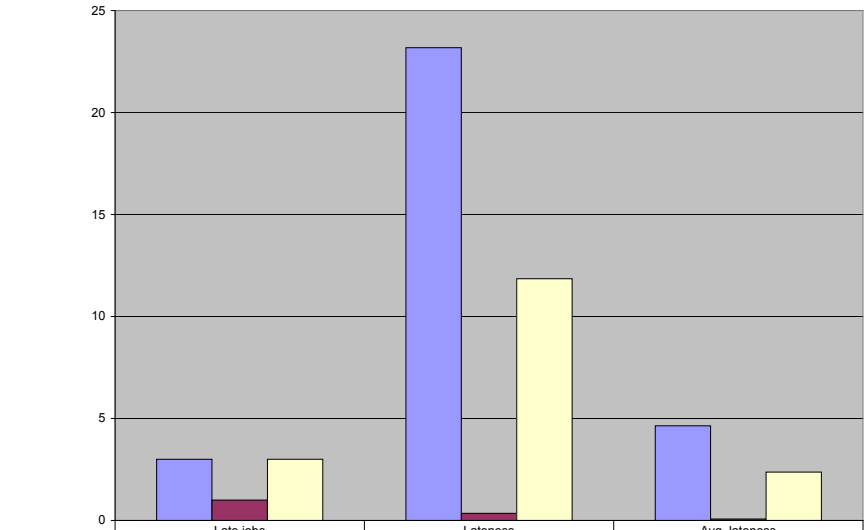
	Lateness	Late jobs
Existing function	4.35	4
Prio 2	1.6	1
Heur + Prio1	1.55	1
Heur + Prio 2	1.33	1
Heur + Prio 3	1.33	1
Heur + Prio 4	0.12	1
Heur + Prio 5	1.33	1
Heur + Prio 6	1.33	1

Figure 6.5: The total lateness and the number of late jobs for the existing priority function and the best priority function compared to the fixing variables method used with six priority functions.



	Existing function	Prio 2	Heur + Prio1	Heur + Prio 2	Heur + Prio 3	Heur + Prio 4	Heur + Prio 5	Heur + Prio 6
Avg. lateness	0.13	0.05	0.05	0.04	0.04	0.00	0.04	0.04

Figure 6.6: The average lateness for the existing priority function and the best priority function compared to the fixing variables method used with six priority functions.



	Late jobs	Lateness	Avg. lateness
Existing function	3	23.18	4.64
Optimal	1	0.35	0.07
Heuristic	3	11.85	2.37

Figure 6.7: Results for the small example from Section 3.4 with the fixing variables method included.

6.6 Fourth and fifth trial: Local search and relaxing and rounding

It turned out that the local search method did not work as well as expected when it was implemented. The are two main reasons. First of all, the implementation was not a good local search method since it will most likely find a local minimum if it improves the start schedule at all. Secondly, it took too long to optimize after a swap had been made since the schedule after the swap was infeasible, which made it harder for CPLEX to optimize. In the tests that were done with the implemented version the resulting schedule after every swap was worse than the starting schedule.

The relaxing and rounding method was an attempt to see if an easily implemented heuristic might give a good result. The idea was pretty naive and it turned out that the method did not work in practice. After rounding the integer variables it was impossible to generate a feasible schedule. (The integer variables could state that operation 4 of job X should start before operation 6 of job Y and operation 10 of job Y should start before operation 2 of job X, which is a Catch-22 and impossible to realize.)

Chapter 7

Discussion and conclusions

This chapter is divided into three parts. First, the problem in general and the different methods together with the result is discussed in Section 7.1, Section 7.2 discuss further studies, and Section 7.3 presents the conclusions.

7.1 Discussion

Schdeuling job shop problems with multipurpose machines is among the toughest combinatorial problems within optimization. The heuristics presented and tested in this report did not seem to work as well as anticipated. Only one heuristic managed to produce good results in a reasonable amount of time. If a method failed to generate a feasible schedule in a reasonable amount of time for the input that was used to test the usability of the methods, there was no point to continue. (If a method cannot optimize a small example with only five jobs and a decreased number of resources, how good will it be for 20 or more jobs and all the included resources.) This was the case for the Lagrangian method; the main ideas behind the Lagrangian method was to generate several solutions faster and then make it feasible, but when this failed the method proved to be unusable.

The time window method was not designed to be a method that would succeed in creating good schedules fast. It was implemented to get a working model of the multitask cell and as a basis for all the other methods. However, there was one usable solution that the time window method managed to produce (see Section 6.3) and that indicates the difference between a decent schedule and a good one.

The result from the fixing variables method in Section 4.4 was much better than expected. The fact that the result from a priority function compared to the solution from the fixing variables method differed that much

was surprising since they are using the same machine allocation. The main advantage of using the fixing variables method was that it succeeded to generate a schedule in a decent amount of time. In the results for the fixing variables method in Section 6.5 the difference between the fixing variables heuristic and the best priority function did not differ significantly. However, choosing the fixing variables method instead of a priority function will always create a schedule that is at least as good as the schedule from the priority function. Also, the difference between the priority function and heuristic will most likely increase when the load increases.

As mentioned in the result section for the local search method (Section 6.6) it did not work as well as expected. The reason why CPLEX does not manage to produce a feasible schedule after a swap is because CPLEX needs a feasible solution to create an upper bound to be able to prune the branch and bound tree. The idea to use CPLEX to create a new schedule came from the fixing variables method, which seemed effective in fixing the allocation variables and optimize. I did not consider the fact that a feasible solution was used before the optimization part for the fixing variables method.

The tests on the priority function proved to be very successful. The results showed that if larger jobs were prioritized prior to smaller jobs, the resulting schedule will be worse than if smaller jobs has higher priority. In Figures 6.1 and 6.2 (pages 50 and 51) **Prio 4** produced the worst result, which is the function that prioritize larger jobs more than smaller jobs. This gives an indication that prioritizing smaller job more than larger jobs might give a positive effect. Prioritizing a larger job before a smaller job causes the bottleneck operation for the larger job to be the bottleneck operation for the smaller job as well, which is not good since the processing time is most likely larger for the larger job. **Prio 7** proved to be the best priority function tested at eliminating the bottlenecks for the smaller jobs since it gave the best result. Other than that, **Prio 7** differs from the other functions because it does not prioritize late jobs differently from jobs that are on time. In all the other cases the base value (before weighted) is either between 0 and 1 or negative. Having a function that prioritizes late jobs more than jobs on time might cause some jobs that are on time to be late.

There is not much to say about the relaxation and rounding method in Section 4.6, except that it did not work well. The method for rounding could be improved to generate feasible schedules, but there was not enough time to explore such algorithms, and even so it could still not guarantee optimality.

7.2 Further studies and recommendations

Before implementing any optimizing heuristic it may be wise to wait for the result from the *Optimist project*, which is a Ph. D. project at the University of Skövde. However, I suggest Volvo Aero to consider if the existing priority function is the best one for the multitask cell. To change priority function should be rather easy and the result could differ significantly. This should probably be studied further before implementing the new function. There are many factors to regard, like the expected lead time and the arriving schedule. These numerous factors affect the result of a priority function. The focus of this thesis has continuously been to minimize the total lateness since I believe measuring the total lateness shows how well the system works under pressure.

It has come to my knowledge that Volvo Aero might purchase the *XpressMP* optimization tool for a Ph. D. project, which can be used in conjunction with *Microsoft Excel*. This might make it more interesting to try to implement the method from Section 4.4, the best heuristic tested, and also extend the model since not everything was included in the model.

Another improvement that could be made is to have several different priority functions and every time a resource is available use the one that generates the best schedule for say the next 20 hours. If this is something that is considered I think it is important to have a good objective function to measure how good the resulting schedule for the different functions are.

The multitask cell is a stochastic environment, which makes it nearly impossible to construct an optimized schedule. Some of the stochastic variables are: personnel, tools, fixture, processing times and machine break downs, to mention a few. All of them were left out in the model since they were difficult to model and would make it even harder to optimize. Before implementing anything in the real environment some testing needs to be done on a model that can handle most of these stochastic variables.

One thing that is interesting to see is how the solutions from the priority functions would change if the lead times for the product types are changed as well as the arriving schedule. Changing these parameters are probably two of the areas where there are some possibilities to improve the utilization of the machines even more. The result from the tests of the priority functions also indicates that the current expected lead times might be too short. Even though the load in the tests of the priority functions is high, the number of late products and the total lateness seems to be too high.

Volvo Aero might want to consider a study on how to set the due dates as well. One idea I have is to have two due dates where the first is set somewhat earlier depending on the job and one a little bit later. The first due date is

used when calculating the CR-number and the second one is the time when you want the product to be finished. This could decrease the number of late jobs but it creates a new question: how should these two due dates be set?

7.3 Conclusions

I knew that this project would be a tough one, but it proved out to be even harder than expected. I am a bit disappointed that I did not manage to provide more optimized schedules on larger data sets that I did. However, I think that the results provided by this study can motivate further studies regarding the multitask cell.

Because of the stochastic environment of the multitask cell, I believe that the best way to schedule the multitask cell is to use a good priority function. An optimization method can be used, but in that case it should only schedule jobs that will be on time to the multitask cell and not jobs that might be late on the way to the multitask cell. The advantage with using an optimization method is that you get a guarantee on the schedule that you cannot get from a priority function.

To make an optimal schedule for the multitask cell is (almost) impossible. The best heuristic tested in this thesis was the use of a priority function to generate a feasible schedule, and use the machine allocation and optimize the order in which the operations on the resources are processed. (That is, to transform the job shop MPM problem to a regular job shop problem.) When the priority function creates a feasible schedule it makes it possible for CPLEX, which was used during the thesis, to prune the branch and bound tree and find a solution quite fast.

Also, the expected lead times need to be studied further. From the result of the priority functions there are approximately 90% late jobs, which is probably not acceptable. Changing the expected lead times will affect the priority function considerably since the expected lead times are an important part of all the functions tested.

Bibliography

[AEP05] N. Andréasson, A. Evgrafov, and M. Patriksson, *An Introduction to Continuous Optimization*, Studentlitteratur, Lund, 2005.

[MS97] E. J. Anderson, C. A. Glass, C. N. Potts, Machine scheduling, E. Aarts, J. K. Lenstra, eds., *Local Search in Combinatorial Optimization*, John Wiley & Sons Ltd., Chichester, 1997, pp. 361-414.

[Bruc01] P. Brucker, *Scheduling Algorithms*, 3rd ed., Springer-Verlag, Berlin, 2001.

[Roc70] R. T. Rockafellar, *Convex Analysis*, Princeton University Press, Princeton, NJ, 1970.

[GMS98] M. Goossens, F. Mittelbach, A. Samarin, *The LaTeX Companion*, Addison-Wesley, Longman, Inc., Reading, Mass, 1998.

[Ber98] D. P. Bertsekas, *Network optimization: Continous and Discrete Models*, Athena Scientific, Belmont, Mass, 1998.

[CCPS98] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, A. Schrijver, *Combinatorial Optimization*, John Wiley & Sons, Inc., New York, NY, 1998.

[KV02] B. Korte, J. Vygen, *Combinatorial Optimization*, 2nd ed., Springer-Verlag, New York, NY, 2002.

[GJ79] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, 1979.

[FGK05] R. Fourer, D. M. Gay, B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, 2nd ed., Duxbury

Press, Brooks/ColeThomson Publishing Company, Pacific Grove, CA, 2002.

[Jun00] Carolin Juneberg, *Large-scale 0-1 minimization problems solved using dual subgradient schemes, ergodic sequences and core problems*, Master's thesis, Chalmers University of Technology, Gothenburg, 2000.

[wikiNP-H] *NP-hard*, http://en.wikipedia.org/wiki/NP-hard, 2005-12-12.

[wikiNP-C] *NP-complete*, http://en.wikipedia.org/wiki/NP-complete, 2005-12-12.

[mathNP-C] E. W. Weisstein. *NP-Complete Problem*, From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/NP-CompleteProblem.html, 2005-12-12.

[ILOG] *ILOG AMPL CPLEX System Version 8.1 - User's Guide*, 2002.

Chapter 8

Glossary

AMPL - Short for "A Mathematical Programming Language", a high level programming language, developed at *Bell Laboratories*, for describing and solving mathematical problems.

CPLEX - CPLEX is an optimization software package that is compatible with AMPL and is sold by *ILOG*.

fixture - A fixture is an appliance on which a product needs to be securely fixed before processed.

flow shop problem - A manufacturing scheduling problem dealing with multiple machines and jobs, where every job looks the same and there is an ordering of the operations for the jobs (all jobs has to go through all machines).

Gantt chart - A Gantt chart is used to show activities as they occur over time.

heuristic - A heuristic is an algorithm that generate solutions without guarantee of optimality.

job - A job is a set of connected operations.

job shop problem - A manufacturing scheduling problem dealing with multiple machines and jobs, where there is an ordering of the operations for the all jobs.

lateness - The lateness for a job is the number of hours (time units) a job is late; if it is on time the lateness is 0.

Matlab - Matlab is a numerical computing and programming language created by *The MathWorks*

multipurpose machine - (MPM) machines that are able to process several different types of operations.

multitask cell - The production cell at Volvo Aero that is studied in this project.

multitask machine - The names of the MPM machines in the multitask cell.

open shop problem - A manufacturing scheduling problem dealing with multiple machines and jobs, where there is no ordering for the operations for the jobs.

operation - An operation, for this report, is defined as something that requires a resource's full attention in the multitask cell for some period of time.

polyhedron - A polyhedron is defined as the solution set to a finite number of linear inequalities, that is, for $i = 1, \ldots, m$, $a_i \in \Re^n$, and $b_i \in \Re$, a polyhedron has the form

$$\{x \in \Re^n : a_i^T x \leq b_i, i = 1, \ldots, m\}.$$

resource - A resource is defined as a workstation or machine in the multitask cell.

Appendix A

Computer code

A.1 The AMPL-model

```
###
### A AMPL-model for the Multi-Task problem at Volvo
   Aero Corp. at äTrollhttan.
### This was done as a part of a master thesis at
   Chalmers Univ. of Tech.
### If you want to optimize a static schedule just set
    the currentTime
### parameter to a sufficient large number.
###
### author: Tomas Jansson
###


### SETS ###
# The set of products
set PRDS ordered;

# The set of machines
set MACHINES ordered;


### PARAMETERS ###
# Need this number to set up the constraints
param largeNumber default 0;
```

```
# Max number of operation for one product.
param maxOps >= 1 integer;

# Max number of one product wanted.
param maxPrd >= 1 integer;

# The number of operations for every product
param nrOpsPerPrd {PRDS} > 0 integer;

# Defines how many of each product that is wanted
param nrOfPrd {PRDS} >= 0 integer;

# The releas dates for the jobs.
param releaseDates {h in PRDS,i in 1..nrOfPrd[h
   ]} >= 0;

# The due dates for the jobs.
param dueDates {h in PRDS,i in 1..nrOfPrd[h]} >= 0;

# The operation times for each product
param opTimes {h in PRDS,i in 1..nrOpsPerPrd[h]} >= 0;

# validMachine[i,j,k] defines if the machine k is
# valid for operation j of job i
param validMachine {h in PRDS,i in 1..nrOpsPerPrd[h],
   MACHINES} binary;

# A "fake" parameter that fakes that the machines are
   busy at startup.
param fakeOp {MACHINES};

# The time a product has to wait before it can be
   processed on a new resource.
param waitTime >= 0 default 0.1;

# Controls wich of the oneJperM1 that should be in the
    Lagrange duality problem,
# a 1 says that the constraint shoud be in the problem
param ctrlConst1 {h in PRDS,i in 1..nrOfPrd[h], j in
    1..nrOpsPerPrd[h],
                o in PRDS,p in 1..nrOfPrd[o], q in 1..
```

```
                    nrOpsPerPrd[o],k in MACHINES}
                    binary default 1;

# Controls wich of the oneJperM2 that should be in the
    Lagrange duality problem,
# a 1 says that the constraint shoud be in the problem
param ctrlConst2 {h in PRDS,i in 1..nrOfPrd[h], j in
    1..nrOpsPerPrd[h],
                o in PRDS,p in 1..nrOfPrd[o], q in 1..
                    nrOpsPerPrd[o],k in MACHINES}
                    binary default 1;


param constCount default 0;     # Keeps count in how
   many constraints that has beeing added in total.

### PARAMETERS USED IN THE PRIORITY FCN ###
# This parameter control the main loop
param mainDone default 0;

param timeStep default 10;

# The current time.
param currentTime >= 0;

# Temporary time parameters that is used for
param temp1Time >= 0;

# priority calculation.
param temp2Time >= 0;

# The available machine
param availMachine symbolic in MACHINES;

# The product with the lowest priority number
param prioPrd symbolic in PRDS;

# The job with the lowest priority number
param prioJob;

# The operation with the lowest priority number
```

```
param prioOp;

# The currently lowest priority number.
param minPrio;

# Used when calculating the priority number.
param currentPrio;

# Helps to iterate over the machines.
param machineChecked {MACHINES} default 0;

# The time when a machine is done with the current
   operation.
param machineDone {MACHINES} >= 0;

# The priority number for each job.
param prioNr {PRDS, 1..maxPrd} default 100000;

# Indicates how far each job has come.
param opsDone {PRDS,1..maxPrd} default 0;

# A loop variable.
param done default 0;

# This is used to keep track of when the previos
   operation is done
# when calculating the priority numbers.
param prevDone;


### VARIABLES ###
# opStart[i,j,k] defines the start time of
# operation j of job i on machine k (if
# operation j of job i is is performed on
# machine k).
var opStart {PRDS, 1..maxPrd,1..maxOps,MACHINES} >= 0
   default 0;

# a copy of opStart, that stores the best value so far
var opStart_opt {PRDS, 1..maxPrd,1..maxOps,MACHINES
   } >= 0 default 0;
```

```
# machineAlloc[i,j,k] is one if
# operation j of job i is performed on
# machine k, zero otherwise.
var machineAlloc {PRDS, 1..maxPrd,1..maxOps,MACHINES}
   binary;

# a copy of machineAlloc, that stores the best value
   so far
var machineAlloc_opt {PRDS, 1..maxPrd,1..maxOps,
   MACHINES} binary;

# order[i,j,p,q,k] is one if
# operation j of job i is
# scheduled before operation q
# of job p on machine k.
var order {PRDS, 1..maxPrd,1..maxOps,
        PRDS, 1..maxPrd,1..maxOps,MACHINES} default 0
           binary;

# copy of order
var order_opt {PRDS, 1..maxPrd,1..maxOps,
        PRDS, 1..maxPrd,1..maxOps,MACHINES} default 0
           binary;

var costVar {h in PRDS, i in 1..maxPrd} >= 0;

# copy of costVar
var costVar_opt {h in PRDS, i in 1..maxPrd} >= 0;




### PARAMETERS AND VARIABLES FOR THE LDS (LANGRANGIAN
   DUAL SUBPROBLEM) ###

param my1 {h in PRDS,i in 1..nrOfPrd[h],j in 1..
   nrOpsPerPrd[h],
        o in PRDS,p in 1..nrOfPrd[o],q in 1..
           nrOpsPerPrd[o],k in MACHINES:
        not(h = o and i = p and j = q)} >= 0 default
```

```
            0;

param  my2 {h in PRDS,i in 1..nrOfPrd[h],j in 1..
   nrOpsPerPrd[h],
         o in PRDS,p in 1..nrOfPrd[o],q in 1..
            nrOpsPerPrd[o],k in MACHINES:
         not(h = o and i = p and j = q)} >= 0 default
            0;

param my1_2 {h in PRDS,i in 1..nrOfPrd[h],j in 1..
   nrOpsPerPrd[h],
         o in PRDS,p in 1..nrOfPrd[o],q in 1..
            nrOpsPerPrd[o],k in MACHINES:
         not(h = o and i = p and j = q)} >= 0 default
            0;

param  my2_2 {h in PRDS,i in 1..nrOfPrd[h],j in 1..
   nrOpsPerPrd[h],
         o in PRDS,p in 1..nrOfPrd[o],q in 1..
            nrOpsPerPrd[o],k in MACHINES:
         not(h = o and i = p and j = q)} >= 0 default
            0;


param upperBound default 0;

# An approximation parameter that is used when
   calculating the step-size.
param approx default 0;

param currentDualSol default 0;

# The solution from the previous iteration in the
   lagrangian iteration.
param oldDualSol default 0;

# The maximum solution we have seen durint the last
   progressCtr iterations
param maxProgressSol;

# The minimum solution we have seen durint the last
```

```
   progressCtr iterations
param minProgressSol;

param progressDiff;

# How many iterations we should make before we
# deciding if we should change the stepWeight
   parameter
param progressCtr default 10;

# A "weight" that is used when calculating the step-
   size.
param stepWeight default 1.5;

# This keeps count on if we're
param stepWeightCounter default 0;

# The size which we at least want to improve our
# solution with in every iteration (Lagrangian)
param speed default 0.001;

# When is it time to change the stepWeight?
param maxStepWeightCounter = 10;

param lagrangianCounterMax default 100;

# This is used when checking if a dual solution is
   feasible to the primal problem.
param isFeasible default 0;

var const1Var {h in PRDS, i in 1..nrOfPrd[h], j in 1..
   nrOpsPerPrd[h],
        o in PRDS, p in 1..nrOfPrd[o], q in 1..
           nrOpsPerPrd[o], k in MACHINES:
        not(h = o and i = p and j = q)} =
        (if (ctrlConst1[h,i,j,o,p,q,k] == 0) then 0
           else ((opStart[h,i,j,k]+opTimes[h,j]-
           largeNumber*(1-machineAlloc[h,i,j,k])) -
        (largeNumber*(2-order[h,i,j,o,p,q,k]-
           machineAlloc[o,p,q,k])+opStart[o,p,q,k])));
```

```
var const2Var {h in PRDS, i in 1..nrOfPrd[h], j in 1..
   nrOpsPerPrd[h],
        o in PRDS, p in 1..nrOfPrd[o], q in 1..
           nrOpsPerPrd[o], k in MACHINES:
        not(h = o and i = p and j = q)} =
        (if (ctrlConst2[h,i,j,o,p,q,k] == 0) then 0
           else ((opStart[o,p,q,k]+opTimes[o,q]-
           largeNumber*(1-machineAlloc[o,p,q,k])) -
        (largeNumber*(1+order[h,i,j,o,p,q,k]-
           machineAlloc[h,i,j,k])+opStart[h,i,j,k])));

var temp1Var {h in PRDS, i in 1..nrOfPrd[h], j in 1..
   nrOpsPerPrd[h],
        o in PRDS, p in 1..nrOfPrd[o], q in 1..
           nrOpsPerPrd[o], k in MACHINES:
        not(h = o and i = p and j = q)} = const1Var[h,
           i,j,o,p,q,k]^2;

var temp2Var {h in PRDS, i in 1..nrOfPrd[h], j in 1..
   nrOpsPerPrd[h],
        o in PRDS, p in 1..nrOfPrd[o], q in 1..
           nrOpsPerPrd[o], k in MACHINES:
        not(h = o and i = p and j = q)} = const2Var[h,
           i,j,o,p,q,k]^2;

# Need one stepsize parameter for each relaxed
   constraint.
var stepSize1 = stepWeight*(approx-currentDualSol)/
        sum{h in PRDS, i in 1..nrOfPrd[h], j in 1..
           nrOpsPerPrd[h],
        o in PRDS, p in 1..nrOfPrd[o], q in 1..
           nrOpsPerPrd[o], k in MACHINES:
        not(h = o and i = p and j = q)} (temp1Var[h,i,
           j,o,p,q,k]+temp2Var[h,i,j,o,p,q,k]);

var stepSize;

var max1Var {h in PRDS, i in 1..nrOfPrd[h], j in 1..
   nrOpsPerPrd[h],
        o in PRDS, p in 1..nrOfPrd[o], q in 1..
           nrOpsPerPrd[o], k in MACHINES:
```

```
        not(h = o and i = p and j = q)} = max(0,my1[h,
           i,j,o,p,q,k]+stepSize*const1Var[h,i,j,o,p,q
           ,k]);

var max2Var {h in PRDS, i in 1..nrOfPrd[h], j in 1..
   nrOpsPerPrd[h],
        o in PRDS, p in 1..nrOfPrd[o], q in 1..
           nrOpsPerPrd[o], k in MACHINES:
        not(h = o and i = p and j = q)} = max(0,my2[h,
           i,j,o,p,q,k]+stepSize*const2Var[h,i,j,o,p,q
           ,k]);


### OBJECTIVE ###
# The main objective function strives to keep costVar
   as low as possible.
minimize totCost:
        sum{h in PRDS, i in 1..nrOfPrd[h]:releaseDates
           [h,i] < currentTime+timeStep} costVar[h,i];

# The Langrangian dual
minimize dualCost:
        sum{h in PRDS, i in 1..nrOfPrd[h]:releaseDates
           [h,i] < currentTime+timeStep}
        (costVar[h,i] +
        sum{j in 1..nrOpsPerPrd[h], o in PRDS, p in
            1..nrOfPrd[o],
        q in 1..nrOpsPerPrd[o], k in MACHINES:
        releaseDates[o,p] < currentTime+timeStep and
           not(h==o and i==p and j==q)}
        (my1[h,i,j,o,p,q,k]*const1Var[h,i,j,o,p,q,k] +
        (my2[h,i,j,o,p,q,k]*const2Var[h,i,j,o,p,q,k]))
           );


### CONSTRAINTS ###
# The fake operation has to be ready before the
   machine
#can process a real operation.
subject to fakeOps{h in PRDS, i in 1..nrOfPrd[h], j in
    1..nrOpsPerPrd[h],
```

```
        k in MACHINES:
                releaseDates[h,i] < currentTime+
                   timeStep}:
        opStart[h,i,j,k] - fakeOp[k]*machineAlloc[h,i,
           j,k] >= 0;


# Creates the values for the costVar variables,
# and they are defined as the time that the jobs
   exceeds their due dates.
subject to costConstraint {h in PRDS,i in 1..nrOfPrd[h
   ],k in MACHINES:
                           releaseDates[h,i] <
                              currentTime+timeStep}:
        opStart[h,i,nrOpsPerPrd[h],k] + opTimes[h,
           nrOpsPerPrd[h]] - costVar[h,i] <=
        dueDates[h,i];


# The first operation can't start before its release
   date.
subject to illegalStart{h in PRDS, i in 1..nrOfPrd[h
   ], k in MACHINES:
                         releaseDates[h,i] <
                            currentTime+timeStep}:
        opStart[h,i,1,k] >= releaseDates[h,i]*
           machineAlloc[h,i,1,k];


# Removes "illegal" values in opStart.
# (Had some problems with values that were set to ~
   largeNumber)
subject to badOps{h in PRDS, i in 1..nrOfPrd[h], j in
    1..nrOpsPerPrd[h],
        k in MACHINES:
                  releaseDates[h,i] < currentTime+
                     timeStep}:
        opStart[h,i,j,k] - largeNumber*machineAlloc[h,
           i,j,k] <= 0;
```

```
# A operation can only be done on one machine
subject to oneOponeM {h in PRDS,i in 1..nrOfPrd[h], j
   in 1..nrOpsPerPrd[h]:
                     releaseDates[h,i] < currentTime+
                        timeStep}:
        sum {k in MACHINES} machineAlloc[h,i,j,k] = 1;


# The operation has to be done on a valid machine
subject to validity1 {h in PRDS,i in 1..nrOfPrd[h], j
   in 1..nrOpsPerPrd[h],
        k in MACHINES:
                     releaseDates[h,i] < currentTime+
                        timeStep}:
        machineAlloc[h,i,j,k] <= validMachine[h,j,k];


# Only one of the operations [h,i,j] and [o,p,q] can
   have their order variable set
subject to validity4 {h in PRDS,i in 1..nrOfPrd[h], j
   in 1..nrOpsPerPrd[h],
                o in PRDS,p in 1..nrOfPrd[o], q in 1..
                   nrOpsPerPrd[o],k in MACHINES:
                releaseDates[h,i] < currentTime+
                   timeStep and
                releaseDates[o,p] < currentTime+
                   timeStep}:
        (if (h==o and i==p and j==q) then 0 else
        (order[h,i,j,o,p,q,k] + order[o,p,q,h,i,j,k]))
            <= machineAlloc[h,i,j,k];


# Force one of the order variable to be set.
subject to forceOrder1 {h in PRDS,i in 1..nrOfPrd[h],
   j in 1..nrOpsPerPrd[h],
                o in PRDS,p in 1..nrOfPrd[o], q in 1..
                   nrOpsPerPrd[o],k in MACHINES:
                releaseDates[h,i] < currentTime+
                   timeStep and
                releaseDates[o,p] < currentTime+
                   timeStep}:
```

```
        (if (h==o and i==p and j==q) then 0 else
        machineAlloc[h,i,j,k]+machineAlloc[o,p,q,k] -
        (order[h,i,j,o,p,q,k]+order[o,p,q,h,i,j,k]+1))
            <= 0;


# Force the operations in one job to be correct
subject to forceOrder2 {h in PRDS,i in 1..nrOfPrd[h],
   j in 1..nrOpsPerPrd[h],
        j_2 in 1..nrOpsPerPrd[h], k in MACHINES:
        releaseDates[h,i] < currentTime+timeStep}: if
            (j_2 < j) then largeNumber else
        order[h,i,j,h,i,j_2,k] >= order[h,i,j_2,h,i,j,
           k];



# Make sure that there is only one job allocated at a
   machine at any given time
# Relaxed to the dual function
subject to oneJperM1 {h in PRDS,i in 1..nrOfPrd[h], j
   in 1..nrOpsPerPrd[h],
                o in PRDS,p in 1..nrOfPrd[o], q in 1..
                   nrOpsPerPrd[o],k in MACHINES:
                releaseDates[h,i] < currentTime+
                   timeStep and
                releaseDates[o,p] < currentTime+
                   timeStep}:
        (if ((h==o and i==p and j==q) or ctrlConst1[h,
           i,j,o,p,q,k] == 1) then 0 else
        (opStart[h,i,j,k] + opTimes[h,j] - largeNumber
            * (1-machineAlloc[h,i,j,k]) -
        (largeNumber*(2 - order[h,i,j,o,p,q,k] -
           machineAlloc[o,p,q,k]) +
        opStart[o,p,q,k]))) <= 0;


# Relaxed to the dual function
subject to oneJperM2 {h in PRDS,i in 1..nrOfPrd[h], j
   in 1..nrOpsPerPrd[h],
                o in PRDS,p in 1..nrOfPrd[o], q in 1..
```

```
                  nrOpsPerPrd[o],k in MACHINES:
               releaseDates[h,i] < currentTime+
                  timeStep and
               releaseDates[o,p] < currentTime+
                  timeStep}:
         (if ((h==o and i==p and j==q) or ctrlConst2[h,
            i,j,o,p,q,k] == 1)then 0 else
         (opStart[o,p,q,k] + opTimes[o,q] - largeNumber
            *(1 - machineAlloc[o,p,q,k]) -
         (largeNumber*(1 + order[h,i,j,o,p,q,k] -
            machineAlloc[h,i,j,k]) +
         opStart[h,i,j,k]))) <= 0;

# A copy that doesn't take the ctrlConst1 variable in
   account
subject to oneJperM1_2 {h in PRDS,i in 1..nrOfPrd[h],
   j in 1..nrOpsPerPrd[h],
               o in PRDS,p in 1..nrOfPrd[o], q in 1..
                  nrOpsPerPrd[o],k in MACHINES:
               releaseDates[h,i] < currentTime+
                  timeStep and
               releaseDates[o,p] < currentTime+
                  timeStep}:
         if (h==o and i==p and j==q) then 0 else
         opStart[h,i,j,k] + opTimes[h,j] - largeNumber
            *(1 - machineAlloc[h,i,j,k]) -
         (largeNumber*(2 - order[h,i,j,o,p,q,k] -
            machineAlloc[o,p,q,k]) +
         opStart[o,p,q,k]) <= 0;

# Same as above
subject to oneJperM2_2 {h in PRDS,i in 1..nrOfPrd[h],
   j in 1..nrOpsPerPrd[h],
               o in PRDS,p in 1..nrOfPrd[o], q in 1..
                  nrOpsPerPrd[o],k in MACHINES:
               releaseDates[h,i] < currentTime+
                  timeStep and
               releaseDates[o,p] < currentTime+
                  timeStep}:
         if (h==o and i==p and j==q) then 0 else
         opStart[o,p,q,k] + opTimes[o,q] - largeNumber
```

```
            *(1 - machineAlloc[o,p,q,k]) -
         (largeNumber*(1 + order[h,i,j,o,p,q,k] -
            machineAlloc[h,i,j,k]) +
         opStart[h,i,j,k]) <= 0;


# The operation within a job have to be done in the
   right order
subject to jobOrder {h in PRDS,i in 1..nrOfPrd[h], j
   in 1..nrOpsPerPrd[h]-1,
                k in MACHINES,l in MACHINES:
                   releaseDates[h,i] < currentTime+
                   timeStep}:
         largeNumber*(1-machineAlloc[h,i,j+1,k])+
            opStart[h,i,j+1,k] >=
         opStart[h,i,j,l]+opTimes[h,j]+waitTime-
            largeNumber*(1-machineAlloc[h,i,j,l]);


### PROBLEM DEFINTION ###
problem primalProb: opStart, machineAlloc, order,
   costVar, totCost,
         fakeOps, costConstraint, illegalStart, badOps
            , oneOponeM,
         validity1, validity4, oneJperM1_2, oneJperM2_2
            , jobOrder, forceOrder2;

problem lagrangianDual2: opStart, machineAlloc, order
   , costVar, const1Var,
         const2Var, dualCost, fakeOps, costConstraint,
            illegalStart, badOps,
         oneOponeM, validity1, validity4, jobOrder,
            forceOrder2,forceOrder1;;
```

A.2 Example input

```
###
### The data for the Multi-Task AMPL-model.
###
### author: Tomas Jansson
###
```

```
# {M1,..M3} = <set up stations>, {M4,...M8} = <
   multitask machines>, {M9} =
# <manuell gradningsstation> and {M10} = <robot
   gradning>.
set MACHINES = M1,M3,M4,M8,M9,M10;

param waitTime = 0.1;

param maxPrd:= 3;

param maxOps:= 11;

param largeNumber := 300;


param: PRDS: nrOfPrd:=
        #list as follows:
        #prd1 <the ammount of prd1>
        #prdLast <the ammount of prdlast>;
        prdX    2
        prdY    3;


param nrOpsPerPrd:=
        #list as follows:
        #prd1 <the number of operation for prd1>
        #prdLast <the number of operation for prdLast
           >;
        prdX    11
        prdY    7;

param interval :=
        prdX    15
        prdY    15;

param firstArrival :=
        prdX    2
        prdY    2;
```

```
param expLeadTime:=
        prdX    68.33
        prdY    17.05;

param fakeOp
        M1              7
        M3              6
        M4              8
        M8              10
        M9              17
        M10     25;



# A tear down followed by a set up is considered to be
    a combined operation

# Each column is the operation number for the product
   and the rows are the machines.
# 1 in the matrices indicates that the machine is
   valid for that operation.
param validMachine
[prdX,*,*] (tr)
:       1       2       3       4       5       6
          7       8       9       10      11:=
M1      1       0       0       0       1       0
          1       0       0       0       1
M3      1       0       0       0       1       0
          1       0       0       0       1
M4      0       1       0       0       0       1
          0       1       0       0       0
M8      0       1       0       0       0       1
          0       1       0       0       0
M9      0       0       1       0       0       0
          0       0       1       0       0
M10     0       0       0       1       0       0
          0       0       0       1       0
[prdY,*,*] (tr)
:       1       2       3       4       5       6
          7       8       9       10      11:=
M1      1       0       1       0       1       0
```

```
          1       .       .       .       .
M3        1       0       1       0       1       0
          1       .       .       .       .
M4        0       1       0       1       0       1
          0       .       .       .       .
M8        0       1       0       1       0       1
          0       .       .       .       .
M9        0       0       0       0       0       0
          0       .       .       .       .
M10       0       0       0       0       0       0
          0       .       .       .       .;

# Each column is the operation number for the product
   and the rows are the machines.
# The values in the matrices tells the operation time
   for the specific operation.
# If the value is 0 it means that the operation number
    doesn't exist.
param opTimes
:               1               2               3
                  4               5               6
                  7               8               9
                  10              11:=
prdX     0.75     8.1      0.5     2               1.25
        10.8     1.25     21.68   2               5.5
         0.5
prdY     0.75     1.2      1.25    1.2     1.25    4.9
         0.5      .                .               .
                  .;
```