Computational Strategies for Model Predictive Control on Switching Max-plus Linear Systems

T.D. Vos

Master of Science Thesis

Delft University of Technology

Delft Institute of Applied Mathematics
Delft Center for Systems & Control

Computational Strategies for Model Predictive Control on Switching Max-plus Linear Systems

Master of Science Thesis

For the degrees of Master of Science in Applied Mathematics and Systems & Control at Delft University of Technology

T.D. Vos

November 14, 2019

Faculties of Electrical Engineering, Mathematics & Computer Science (EEMCS) and Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

A switching max-plus linear model is a framework to describe the discrete dynamics of the timing of events. To influence these systems one can choose the routes of jobs and the orderings of operations as input for the system. In this thesis the techniques of model predictive control are used to find good input values. The problem of finding the optimal input is however $\mathcal{NP}$-hard, which means there is no guarantee to find the optimal in a reasonable amount of time. This is an issue for model predictive control on real applications of the switching max-plus model. In applications, on-line performance is used where there is limited time to compute the input values for control.

This thesis takes a look into methods to reduce the computational complexity of the MPC-SMPL problem. Alternative formulations such as reparameterization, model based-partitioning and the cutting plane method are developed and tested for the MPC-SMPL problem. To solve the MPC-SMPL problem 3 heuristics are designed and implemented for simulation. The heuristics are partition-based optimization, tabu search and simulated annealing. The goal is to find a strategy that obtains the best solution to the problem in a limited amount of time.

Preface

If you are reading this that means that I managed to finish my thesis. This work was my graduation project for a double degree for the Master of Science in Applied Mathematics and Systems & Control. I was really glad that I found a project where I could work on discrete mathematics as well as a control problem and all the challenges it brings. For me, it makes mathematics much more interesting when you are able to translate it to real problems, which is why I chose a double degree in the first place.

Although the fields of both Master degrees lie very close to each other in the subject of this thesis, a distinction between both fields is made in the report. Parts I, II and IV are designated as the thesis for Systems & Control and Parts I, III and IV are designated as the thesis for Applied Mathematics. Part V consists of the Appendices and contains mainly background information and details for the whole thesis.

I would like to thank my supervisors for guiding me on the project during the past fifteen months. Whenever I encountered a difficulty they were able to come up with many ideas I could further investigate. Along the project they remained critical, which is a very good thing to have by your side. Luckily, we did not only talk about my project during the meetings. During the coffee before the start of the meetings we talked about all sorts of things, which provided an informal and comfortable atmosphere.

I would also like to thank my supervisors and committee members for reading my thesis. Their feedback have helped me develop the report in which I tell the story of my research.

Supervisors & committee members

dr. ir. A.J.J. van den Boom
supervisor and chair of the committee from Systems & Control

dr. J.W. van der Woude
supervisor and chair of the committee from Applied Mathematics

A. Gupta
supervisor and committee member

prof. dr. ir. K.I. Aardal
committee member

Contents

Introduction **3**
Current state of the art . . . 3
The aim of this thesis . . . 4

Part I Theory & Modeling **6**

1 Max-Plus Algebra **7**
1.1 The max-plus eigenvalue problem . . . 9

2 Max-Plus Dynamical Systems **11**
2.1 Switching max-plus systems . . . 15
2.2 Routing, ordering and synchronization . . . 16
2.3 Scheduling problems . . . 19

3 Model Predictive Control for SMPL Systems **22**
3.1 Model predictive control . . . 22
3.2 Recasting the MPC problem as an MILP . . . 24
3.3 The MPC-SMPL problem specification . . . 25

4 Applications of SMPL systems **26**
4.1 The railway network . . . 26
4.2 The container terminal . . . 30

Part II Reducing complexity of the MPC-SMPL problem **34**

5 Reparameterization of Binary Variables **35**
5.1 Routing parameterization . . . 36
5.2 Ordering parameterization . . . 41
5.3 The Quine-McClusky method . . . 44
5.4 The implementation of reparameterizations . . . 47

6 Model-based Partitioning **50**
6.1 Partitioning . . . 51
6.2 Partition-based optimization . . . 53

Part III Bounds on the MPC-SMPL Problem **58**

7 Primal Bounds **59**
7.1 Greedy algorithms . . . 59
7.2 Tabu search . . . 63
7.3 Simulated annealing . . . 67

8 Dual Bounds **69**
8.1 Cuts for starting times . . . 69
8.2 Cutting on binary parameterizations . . . 77

Part IV Simulation & Results **80**

9 Reformulation Results of the MPC-SMPL Problem **83**
9.1 Results on reparameterization 83
9.2 Results on partitioning 87
9.3 Strength of the dual bound 88

10 Computational Results **90**
10.1 Algorithm comparison on general SMPL systems 90
10.2 Results on applications 92

11 Final Words **95**
11.1 Contributions 95
11.2 Conclusions 96
11.3 Recommendations 97

Part V Appendix **99**

A An MPC-SMPL Example **100**
A.1 The scenario and SMPL system model 100
A.2 Modeling with routing and ordering 102
A.3 Transformation of SMPL to MPC 104

B Linear Optimization **109**
B.1 Linear programming 109
B.2 Convexity, polyhedra & formulations 110
B.3 Complexity theory 114
B.4 The branch-and-bound approach 116
B.5 Improving the branch-and-bound algorithm 120

C Terminology **124**
C.1 Definitions 124
C.2 Dispatching rules 124

D Appendix for the Railway Network **126**
D.1 Variable routing for the railway network 127
D.2 Railway network partitions 132

E Appendix for Results **134**
E.1 Algorithm parameters 135

F MATLAB Files **137**

Mathematical notations **139**

List of Acronyms **141**

References **142**

Introduction

Modelling has been one of the most important aspects of solving real problems. A good model can help one understand a problem better, which helps finding better solutions. The max-plus algebra is a mathematical language that offers new possibilities in modelling. The use of the max operation allows some system dynamics to be modelled in a linear form. Some examples of applications are a railway network, or a six-legged robot modelled by Kersbergen [34].
To control a switching max-plus linear system this thesis will dive into the techniques of model predictive control (MPC). Model predictive control uses a system model to predict the behaviour of a system by its input over a finite time horizon. Finding a suitable input value for the model is translated to an optimization problem. The problem is then solved to find the best input values used for control, after which the problem moves the horizon in time and the process is repeated. The optimization process for model predictive control for switching systems involves solving a mixed-integer linear program (MILP). The problem is $\mathcal{NP}-$hard, which means that it can take a lot of computation time to solve large instances with currently known algorithms. Since model predictive control is often used on real time applications, it is important to solve the problem within a real time deadline in order to achieve the desired on-line performance. The problem has to be solved before the input is required. A small computation time on solving the MILP is therefore essential.
One can decrease the computational load by decreasing the number of time steps the controller looks into the future, but doing so will undermine the performance of the controller. This thesis will be dedicated to researching other methods to lower the computational load on controlling switching max-plus linear systems with model predictive control, such that controller can maintain a high prediction horizon.

Current state of the art

Recent findings in research of switcing max-plus linear systems mostly intend modelling purposes (van den Boom et al. [50]) or rewriting the problem into forms such as the extended linear complementarity problem (De Schutter & De Moor [17]). To actually solve the problem for MPC on SMPL systems some algorithms are suggested in De Schutter & van den Boom [19], but the detailed structure of the algorithms remains absent. The computational burden for the problem is recognized in De Schutter & van den Boom [19], but not much has been done to reduce the complexity of the problem by the structure of the SMPL system. Kersbergen [34] however did implement a distributed algorithm for the model on the railway network. The report also includes computational results on simulations, which showed promising results from the distributed solver.
Max-plus systems are closely related to scheduling problems as they are described in Pinedo [44]. Scheduling theory is a mathamatical approach that describes the structure and complexity deciding which job has to be performed by which processor and when it should be processed. Mathematicians such as Pinedo [44] have devoted much research to develop methods that improve the process of solving scheduling problems. Not much has been done to place switching max-plus linear system problems in existing scheduling problem classes, or vice versa. Scheduling theory and SMPL systems are treated as two different approaches in the literature. There is however a tremendous amount of available literature for scheduling problems, for which Pinedo [44] provides a solid basis.

Aim of this thesis

The max-plus linear system model in this thesis will be described in a general form. That means that any improvements can be applied to any application that are in the general form. There are many ways to improve on finding solutions for integer problems, but in practice the most effective methods rely on special characteristics of the problem itself. The switching max-plus linear system has a certain structure which might be possible to exploit. Such solutions are often used in integer programming. For example, GUROBI [29] is a solver for integer programs and solves such problems pretty well in general. The traveling salesman problem (Applegate [4]), is a special instance of the integer program, and could be solved by GUROBI. However, there is a special solver called CONCORDE [15] that exploits some special structures of the traveling salesmen problem such that it is faster in solving it than GUROBI in almost any case. Now since CONCORDE uses techniques that only apply to the traveling salesman problem, it cannot solve general integer programs. Likewise, this thesis will look for special characterizations in switching max-plus linear systems that might offer a decrease of the computation time of the controller.

To reduce the complexity of the problem of MPC on SMPL systems, this thesis will take a look on how to formulate the problem optimally. A better formulation of the problem might result in better computational results. This will be done by reparameterization of binary variables (section 5) and by finding cutting planes (section 8) for the MILP formulation.
The other approach in reducing the complexity of the problem in this thesis relies on a heuristic approach. A heuristic algorithm is an algorithm that is very fast but rarely computes the optimal solution. A heuristic algorithm has the potential to come very close to the optimal solution in a very short amount of time. In this thesis 3 algorithm will be designed, implemented and tested on controlling SMPL systems with model predictive control. Partition-based optimization (section 6) is the first heuristic and is similar to the distributed algorithm of Kersbergen [34]. The other heuristic methods are tabu search and simulated annealing, which are both local search approaches (Wolsey [54]).

The mentioned methods will be tested on general SMPL systems without application. In addition, 2 examples of applications (section 4) will be modelled and implemented for additional simulation. In the end, this thesis will serve as a strategy guide on how to solve the MILP arising from MPC on SMPL systems.

Part I
Theory & Modeling

The first part of this thesis consists mainly about the theory and background information required to model switching max-plus linear systems. In addition, the concepts of model predictive control and how it is applied to switching max-plus linear systems will be explained. The main problem specification of this thesis is then formulated for the model and control method. At the end of Part I there will be 2 applications of the SMPL system modelled into the switching max-plus linear system framework.

1 Max-Plus Algebra

The framework of max-plus systems uses the max-plus algebra to model certain timed event-based dynamics. This section will explain what the max-plus algebra is and how the analysis on the algebra follows. Some properties of the max-plus algebra will be derived that will be used later on for system analysis.

An algebra consists of a set of elements and two arithmetic operations, often denoted by a *summation* ($+$) and a *product* ($\star$). More information about algebras can be found in Baccelli et al. [7]. One can consider the *conventional algebra* (Friedberg et al. [22]) as the most common algebra, which is the algebra where the adding and multiplying of real numbers is defined as most people know it. However, In this thesis a different algebra is introduced: the *max-plus algebra* (Heidergott et al. [30]). The reason the max-plus algebra is sometimes chosen over the conventional algebra is that it allows a representation of non-linear synchronization operations of the conventional algebra in a linear way. Heidergott et al. [30] provides an overview of what the max-plus algebra is and what properties can be derived from the algebraic structure. The basic rules are expressed in the following way. Define $\epsilon := -\infty$, and xfor $a, b \in \mathbb{R}_\epsilon := \mathbb{R} \cup \{\epsilon\}$ define

$$a \oplus b := \max\{a, b\}, \qquad a \otimes b := a + b$$

It can be obtained that

$$a \oplus \epsilon = a, \qquad a \otimes \epsilon = \epsilon$$

The max plus algebra is defined by $\mathbb{R}_{\max} = (\mathbb{R}_\epsilon, \oplus, \otimes, \epsilon, 0)$.

Definition 1.1. *An algebra $(\mathcal{S}, +, \star, \boldsymbol{0}, \boldsymbol{1})$ is called a semiring if the following conditions hold for $a, b, c \in \mathcal{S}$:*

1. $\boldsymbol{0}, \boldsymbol{1} \in \mathcal{S}$ *and* $a \star \boldsymbol{0} = \boldsymbol{0} \star a = \boldsymbol{0}$
2. $(a + b) + c = a + (b + c)$ *(additive associativity)*
3. $a + b = b + a$ *(Additive commutativity) and* $a + \boldsymbol{0} = \boldsymbol{0} + a = a$
4. $(a \star b) \star c = a \star (b \star c)$ *(Multiplicative associativity) and* $\boldsymbol{1} \star a = a \star \boldsymbol{1} = a$
5. $a \star (b + c) = (a \star b) + (a \star c)$ *and* $(b + c) \star a = (b \star a) + (c \star a)$ *(left- and right distributivity)*

Notice that all five conditions of Definition 1.1 hold for the max-plus algebra $\mathbb{R}_{\max}$, and thus is $\mathbb{R}_{\max}$ a semiring.

The max-plus binary space (Bousfield [10]) is defined by $\mathbb{B}_\epsilon = \{0, \epsilon\}$. Binary variables are to say if something is *true* (0) or *false* (ϵ). The negation of a binary variable $v \in \mathbb{B}_\epsilon$ in the max-plus algebra is denoted by $\bar{v}$, so $\bar{0} = \epsilon$ and $\bar{\epsilon} = 0$.

Let $\underline{n}$ denote the set $\{1, 2, \ldots, n\}$ for $n \in \mathbb{N}_+ = \mathbb{N}\backslash\{0\}$. The set $\mathbb{R}_\epsilon^{n \times m}$ is the set of all n by m matrices with entries that take values of $\mathbb{R}_\epsilon$. Define the max-plus identity matrix as $E_n \in \mathbb{R}_\epsilon^{n \times n}$ that takes value 0 on the diagonal entries and has value ϵ everywhere else. The matrix $\mathcal{E}_{nm} \in \mathbb{R}_\epsilon^{n \times m}$ consists of the value ϵ on every entry. When it is clear from context what the dimensions are of E_n and $\mathcal{E}_{nm}$ the dimensions will be left out and E and respectively $\mathcal{E}$ will be written. When choosing A and B to be elements of $\mathbb{R}_\epsilon^{n \times m}$ define the max-plus matrix summation as in (1.1).

$$[A \oplus B]_{ij} := A_{ij} \oplus B_{ij}, \quad i \in \underline{n}, j \in \underline{m} \tag{1.1}$$

Here, the notation $[\cdot]_{ij}$ means the entry on the i-th row and j-th column. Now if $A \in \mathbb{R}_{\epsilon}^{n\times l}$ and $B \in \mathbb{R}_{\epsilon}^{l\times m}$ the max-plus matrix product can be defined by equation (1.2).

$$[A \otimes B]_{ij} := \bigoplus_{k=1}^{l} A_{ik} \otimes B_{kj}, \quad i \in \underline{n}, j \in \underline{m} \tag{1.2}$$

Note that it follows that $A \otimes B \in \mathbb{R}_{\epsilon}^{n\times m}$. Just as in conventional algebra, the multiplication is prioritized over addition in the absence of brackets. For example, $M \otimes N \oplus Q$ is equal to $(M \otimes N) \oplus Q$. Powers of matrices are also defined in max-plus algebra. For $A \in \mathbb{R}_{\epsilon}^{n\times n}$ define

$$A^{\otimes k} = \bigotimes_{i=1}^{k} A = A \otimes \cdots \otimes A, \qquad k \in \mathbb{N}$$

such that the matrix multiplication is performed exactly $k-1$ times in the max plus sense. The power series of A is defined by

$$A^{+} = \bigoplus_{k=1}^{\infty} A^{\otimes k}$$

Define $A^{\star} := E \oplus A^{+}$. One can also multiply a scalar $\lambda \in \mathbb{R}_{\epsilon}$ with a matrix $A \in \mathbb{R}_{\epsilon}^{n\times m}$. The result is given by

$$[\lambda \otimes A]_{i,j} = \lambda \otimes [A]_{i,j} \tag{1.3}$$

The max-plus Schur-product (also known as the max-plus Hadamard product [31]) is defined by

$$[A \odot B]_{i,j} = [A]_{i,j} \otimes [B]_{i,j}$$

Definition 1.2. *A directed graph or digraph is a pair $G = (V, D)$ where V is a finite set of vertices and $D \subseteq V \times V$ is a set of possible arcs (directed edges) (i, j) from vertex i to j. An arc can possibly be from a vertex to itself. The vertex set of G is sometimes denoted by $V(G)$ and the arc set of G by $D(G)$.*

A *path* of a graph G is a sequence of vertices $(n_1, n_2, \ldots, n_p)$ where (n_k, n_{k+1}) for $k = 1, \ldots, p-1$ are all arcs of D. A path of G is *simple* if it does not contain repeating vertices. A *circuit* of G is a path of G where its beginning vertex is equal to its last vertex. A circuit of G is *simple* if it does not contain a smaller circuit[1]. If G has a circuit it is *cyclic*, if not then G is acyclic.

The max plus algebra is closely related to spectral graph theory, which is also described in Heidergott et al. [30]. When $A \in \mathbb{R}_{\epsilon}^{n\times n}$ let $\mathcal{G}(A)$ be the graph with $V = \underline{n}$ and let there be an arc from vertex i to j precisely if $a_{ji} \neq \epsilon$. Graph $\mathcal{G}(A)$ also defines a weight function $w : D \to \mathbb{R}$ by $w((i, j)) = a_{ji}$ for $(i, j) \in D$. As defined in Heidergott et al. [30], $\mathcal{G}(A)$ is called the *communication graph* of A.
Also defined in Heidergott et al. [30], define a digraph $G = (V, D)$ to be *strongly connected* when for every vertex i it is possible to find a path to any vertex $j \in V$. A matrix $A \in \mathbb{R}_{\epsilon}^{n\times n}$ is said to be *irreducible* if $\mathcal{G}(A)$ is strongly connected. If A is not irreducible then A is *reducible*. Now $[A^{\otimes k}]_{ij}$ is the maximal weight of a path from j to i of length k in $\mathcal{G}(A)$. The proof of this claim

[1]Normally this would be called a *cycle* instead of simple circuit, but since the word cycle already has another frequently used meaning in this thesis it will be referred as a simple circuit.

is covered in the book of Heidergott et al. [30]. As a direct consequence one can see that $[A^+]_{ij}$ is the maximal weight of any path from vertex j to i.

$$\begin{bmatrix} \epsilon & \epsilon & \epsilon & \epsilon \\ 3 & \epsilon & \epsilon & 1 \\ \epsilon & 2 & 2 & \epsilon \\ \epsilon & 0 & \epsilon & \epsilon \end{bmatrix}$$

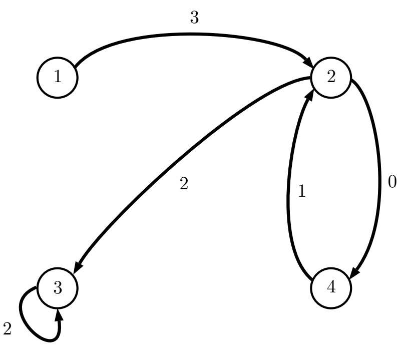


Figure 1.1: A max-plus matrix and its communication graph.

The following Lemma from Heidergott et al. [30] uses the properties of graphs to simplify the computation of A^+ for some matrices.

Lemma 1.3. *If the communication graph of $A \in \mathbb{R}_\epsilon^{n\times n}$ has no circuit C with $w(C) > 0$ then*

$$A^+ = \bigoplus_{k=1}^{n} A^{\otimes k} \tag{1.4}$$

Proof. This proof is similar to the version given in Heidergott et al. [30]. Any entry of $[A^{\otimes k}]_{ji}$ for $k > n$ is equal to the weight of a path p from i to j of length k (Heidergott et al. [30]). Since the length of p is larger than the number of vertices in $\mathcal{G}(A)$, p must contain at least one circuit, denote c. Now define $\tilde{p}$ as the path that is p without circuit c. Because $\mathcal{G}(A)$ does not contain any circuits of positive weight, it holds that $w(\tilde{p}) \geq w(p)$. Iteratively, it can be concluded that for any entry of $A^{\otimes k}$ for $k > n$ there is a larger or equal entry by the same indices of $A^{\otimes m}$ for some $m \leq n$. Hence, equation (1.4) is satisfied. □

In max-plus algebra, it can be requested to solve the equation $x = A \otimes x \oplus b$ for a given $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^m_{\max}$. The following theorem provides a solution to this problem if certain conditions are met.

Theorem 1.4. *Let $A \in \mathbb{R}_\epsilon^{n\times n}$ and $b \in \mathbb{R}_\epsilon^n$. If $\mathcal{G}(A)$ has no positive circuits then $x = A \otimes x \oplus b$ is uniquely solved by $x = A^\star \otimes b$.*

The proof is covered by Heidergott et al. [30]. Theorem 1.4 shows to be a useful property in the max-plus systems theory, introduced in section 2.

1.1 The max-plus eigenvalue problem

Eigenvalues and eigenvectors play an important role in the conventional linear algebra (Friedberg et al. [22]) and systems theory (Leigh [37]). It makes therefore sense to study the role of max-plus eigenvalues if one works on systems described by the max-plus algebra (section 2).

Definition 1.5. *Let $A \in \mathbb{R}_\epsilon^{n\times n}$. If there exists a scalar $\lambda \in \mathbb{R}_\epsilon$ and a vector $z \in \mathbb{R}_\epsilon^n$ with at least one finite element such that*

$$A \otimes z = \lambda \otimes z \tag{1.5}$$

then λ is called an eigenvalue of A and z is an eigenvector of A with resprect to eigenvalue λ.

In max-plus systems the value λ is often used to denote the *cycle-time*, which will be explained in section 2. From Heidergott et al. [30] it is known that if A is irreducible it has exactly one eigenvalue, denoted by $\lambda(A)$. The corresponding eigenvector is however not unique. Notice that if z is the corresponding eigenvector to λ, it holds for a constant $s \in \mathbb{R}$ that

$$\begin{aligned} A \otimes (z \otimes s) &= \begin{bmatrix} \max\{[A]_{11} + z_1 + s, \dots, [A]_{1n} + z_n + s\} \\ \vdots \\ \max\{[A]_{n1} + z_1 + s, \dots, [A]_{nn} + z_n + s\} \end{bmatrix} \\ &= (A \otimes z) \otimes s = (\lambda \otimes z) \otimes s = \lambda \otimes (z \otimes s) \end{aligned} \tag{1.6}$$

where the last step uses the multiplicative associativity. Hence, if z is an eigenvector of A, then $z \otimes s$ is also an eigenvector of A, just as in the conventional algebra (Friedberg et al. [22]). Other than in the conventional algebra, in the max-plus algebra there can exist multiple eigenvectors that are not proportional to each other. Take for example the matrix

$$A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

It can be verified from definition 1.5 that $\lambda = 2$ is an eigenvalue of A and $z = [1\ 0]^T$ is an associated eigenvector. However, by evaluationg equation (1.5) it can be verified that $z = [0\ 1]^T$ is also an eigenvector associated to eigenvalue $\lambda = 2$. Hence, eigenvectors in max-plus algebra are not unique and there can be multiple associate eigenvectors of an eigenvalue that are not proportional to each other.

Theorem 1.6 shows an important relation between the eigenvalue of an irreducible matrix and its communication graph.

Theorem 1.6. *Let $A \in \mathbb{R}_\epsilon^{n\times n}$ be an irreducible matrix. Let $\mathcal{C}(A)$ be the set of all circuits in $\mathcal{G}(A)$, and let $w(c)$ be the weight of a circuit c of $\mathcal{C}(A)$. Now A has a finite eigenvalue and*

$$\lambda(A) = \max_{c\in\mathcal{C}(A)} \frac{w(c)}{|c|}$$

where $|c|$ is the length or number of arcs of circuit c.

The proof is covered by Heidergott et al. [30]. For simplicity, when it is clear from which matrix an eigenvalue is used the eigenvalue will be denoted by λ instead of $\lambda(A)$.

This concludes the basics about max-plus algebra. The properties of the max-plus algebra are different from the conventional algebra in some ways, but also show some similarities. The next section will describe how the max-plus algebra is used for modelling purposes.

2 Max-Plus Dynamical Systems

In this section, the modeling framework of the max-plus system will be explained. All system models in this thesis are of a linear form. First, a max-plus linear system model will be explained. Afterwards it follows how one can extend the system to a switching max-plus linear system. There are three system characteristics that are often used in applications of switching max-plus linear systems that will be explained next. These characteristics are routing, ordering and synchronization. For each of them a method to implement them in the max-plus model will be discussed. Finally, this section explains what a scheduling problem is (Pinedo [2.3]), and how it can be related to max-plus models.

Classical systems theory (Leigh [37]) describes the behaviour of certain states, inputs and outputs. Their behaviour is commonly described over time. In max-plus systems this is normally not the case. Variables in applications of max-plus systems are often modeled such that they describe when a certain event takes place. So instead of depending on time, the variables describe certain time stamps of these events. The max-plus system model describes the timing of events of a *discrete event system* (DES) (more information can be found in Baccelli et al. [7] and Cassandras & Lafortune [13]); a mathematical system where a countable series of events occur in an order allowed by the system.

The discrete event systems in this report describe the timing of operations on processors. An *event* is a moment where a certain *operation* starts to take place. An operation can for example be the production of a product, a traveling train or the delivery of an item. A *processor* is the place where an operation is processed. Examples of processors are a machine that works on a product, a railway track on which trains arrive and depart or an address where an item arrives. A *job* is a series of operations that have to be performed in a specific order, each on a different processor. The set of jobs is usually denoted by $J = \{1, 2, \dots, n\}$ and the set of processors is usually denoted by $M = \{1, 2, \dots, m\}$. An index of a job is usually denoted by j, and the index of a processor by i. The operations are denoted by $e = (j, i)$, where each operation is the part of a job j being processed on a processor i. The time that triggers the operation is the event, denoted by $x(e)$ which are also referred as the *state variables* of the system. Let $i(e)$ denote the processor of operation e and $j(e)$ denote the job of operation e.

For max-plus linear systems, often k is used to denote a certain *operation-cycle* of a process. The times that events occur are usually computed per cycle. Every cycle has a *cycle time*, which denotes the time length of that cycle. For example, if one models a railway network and wants every departure time to repeat every hour, k represents the working hour. One can also define k as a batch of operations that has to be processed repeatedly. Whether cycle k is repeated a fixed number of times, or continuously should always be clear from the application or model problem. The simplest max-plus system is an *autonomous max-plus linear system* (autonomous MPL system). The system contains no input variables or switching mechanics. The general form of the autonomous max-plus linear system is given by the relation

$$x(k) = \bigoplus_{\mu=\hat{\mu}}^{\mu_{\max}} A_\mu \otimes x(k-\mu), \quad k \geq 0 \tag{2.1}$$

where $\hat{\mu} \in \{0, 1\}$, $x(-1), \dots, x(-\mu_{\max}) \in \mathbb{R}^n$ are initial values, $A_i \in \mathbb{R}^{n \times n}$ and $\mu_{\max}$ is the number of previous states that are taken into consideration to determine the next state. If $\hat{\mu} = 0$ the model is an *implicit* max-plus model and if $\hat{\mu} > 0$ the model is *explicit*. Take $\mu_{\max} = 1$, $\hat{\mu} = 1$

and $x(0)$ to be an eigenvector of A_1 in the max-plus sense, to recursively obtain solution (2.2).

$$x(k) = A_1^{\otimes k} \otimes x(0) = \lambda(A_1)^{\otimes k} \otimes x(0) \tag{2.2}$$

The entire behaviour of x can actually be uniquely determined.

Example 2.1. Suppose a product consists of 3 parts, made on 2 machines. The cycle time to create a single product is to be minimized. Assume the following:

- Part 1 and part 2 can only be created on machine 1. Part 3 can only be created on machine 2.
- Machine 1 can only start producing part 2 if there is a yet unused part 1 completed.
- Productions cannot be interrupted.
- Only full products are requested, there is no use in making single parts without the other parts. Machines can only start producing new parts when there is no spare of that same part available or when all parts are complete.
- Machines start as soon as the production of a part is available.
- It takes no time to combine the 3 parts to the final product.

Denote p_i as the amount of time it takes to produce part i. Lets assume $p_1 = 3, p_2 = 4, p_3 = 6$. This system can be modelled as an autonomous max-plus linear system. Let $x_i(k)$ be the time that one starts the production of part i for the k-th time. Then the starting time of producing part 1 in cycle k can be no earlier than the time where all parts of the previous cycle are finished. The same holds for the second and third part, but the second part also has to wait until the first part of the same cycle is completed. Hence the following equations for $x(k)$ can be obtained.

$$x_1(k) = \max\{x_1(k-1)+3, x_2(k-1)+4, x_3(k-1)+6\} \tag{2.3}$$
$$x_2(k) = \max\{x_1(k)+3, x_1(k-1)+3, x_2(k-1)+4, x_3(k-1)+6\} \tag{2.4}$$
$$x_3(k) = \max\{x_1(k-1)+3, x_2(k-1)+4, x_3(k-1)+6\} \tag{2.5}$$

Notice that from equations (2.3) and (2.4) it follows that

$$\begin{aligned} x_2(k) &= \max\{x_1(k)+3, \max\{x_1(k-1)+3, x_2(k-1)+4, x_3(k-1)+6\}\} \\ &= \max\{x_1(k)+3, x_1(k)\} = x_1(k)+3 \end{aligned} \tag{2.6}$$

From equations (2.3), (2.5) and (2.6) it can now be obtained that the system can be represented as follows.

$$\begin{aligned} x_1(k) &= \max\{x_1(k-1)+3, x_2(k-1)+4, x_3(k-1)+6\} \\ &= \max\{x_2(k-1), x_2(k-1)+4, x_3(k-1)+6\} \qquad (2.7) \\ &= \max\{x_2(k-1)+4, x_3(k-1)+6\} \\ x_2(k) &= x_1(k)+3 \qquad (2.8) \\ x_3(k) &= \max\{x_1(k-1)+3, x_2(k-1)+4, x_3(k-1)+6\} \\ &= \max\{x_2(k-1), x_2(k-1)+4, x_3(k-1)+6\} \qquad (2.9) \\ &= \max(x_2(k-1)+4, x_3(k-1)+6) \end{aligned}$$

Now to make the system explicit substitute equation (2.7) into (2.8) to obtain equation (2.10).

$$x_2(k) = \max\{x_1(k-1)+6, x_2(k-1)+7, x_3(k-1)+9\} \tag{2.10}$$

This can be written in the max-plus linear system explicit form, let $x(k) = [x_1(k)\ x_2(k)\ x_3(k)]^T$, then obtain

$$x(k) = A \otimes x(k-1) \tag{2.11}$$

where

$$A = \begin{bmatrix} \epsilon & 4 & 6 \\ 6 & 7 & 9 \\ \epsilon & 4 & 6 \end{bmatrix}$$

The communication graph $\mathcal{G}(A)$ is shown in figure 2.1.

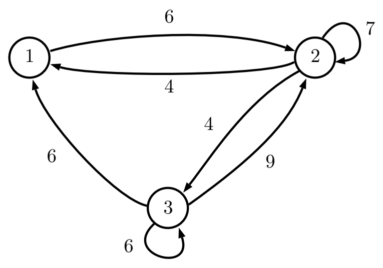


Figure 2.1: The communication graph of A from equation (2.11).

From figure 2.1 it can be obtained that $\mathcal{G}(A)$ has the following 5 simple circuits:

$$(1,2,3,1), (1,2,1), (2,3,2), (2,2), (3,3)$$

By evaluating the mean weight of these circuits it can directly be obtained that the circuit $(2,2)$ has the highest mean weight of value 7. Theorem 1.6 on page 10 now provides that $\lambda(A) = 7$. Note that this approach of finding the eigenvalue is too much work if A is a very large matrix because the number of circuits of $\mathcal{G}(A)$ may grow exponentially. An efficient algorithm to find the eigenvalue of a large matrix is described in Heidergott et al. [30].
Rewriting a system in explicit form by substitution is not always the most efficient way of determining the system matrices. One can also use Theorem 1.4 on page 9 to find the explicit representation from the implicit form. For example, equations (2.3)-(2.5) can directly be put into the implicit system equation (2.12).

$$x(k) = A_0 \otimes x(k) \oplus A_1 \otimes x(k-1), \tag{2.12}$$

$$A_0 = \begin{bmatrix} \epsilon & \epsilon & \epsilon \\ 3 & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon \end{bmatrix}, \quad A_1 = \begin{bmatrix} 3 & 4 & 6 \\ 3 & 4 & 6 \\ 3 & 4 & 6 \end{bmatrix}$$

Since $\mathcal{G}(A_0)$ does not contain any circuits (and thus no positive circuit) Theorem 1.4 can be applied to get an explicit solution for $x(k)$. Since $A_0^{\otimes s} = \mathcal{E}$ for $s \geq 2$ it can be directly obtained that

$$A_0^\star = E \oplus A_0 = \begin{bmatrix} 0 & \epsilon & \epsilon \\ 3 & 0 & \epsilon \\ \epsilon & \epsilon & 0 \end{bmatrix}$$

Applying Theorem 1.4 now results in the explicit equation of the form

$$x(k) = A_0^\star \otimes A_1 \otimes x(k) = \begin{bmatrix} \epsilon & 4 & 6 \\ 6 & 7 & 9 \\ \epsilon & 4 & 6 \end{bmatrix} \otimes x(k) =: A \otimes x(k-1)$$

Notice that $[0\ 3\ 0]^T$ is an eigenvector of A corresponding to eigenvalue 7. So the optimal production scheme for product k can be calculated by selecting $x(0) = [0\ 3\ 0]^T$ as the initial production scheme. The final results are illustrated in figure 2.2. The resulting production times can now be computed as in equation (2.13).

$$x(k) = A \otimes x(k-1) = A^{\otimes k} \otimes x(0) = 7^{\otimes k} \otimes \begin{bmatrix} 0 \\ 3 \\ 0 \end{bmatrix} = \begin{bmatrix} 7k \\ 7k+3 \\ 7k \end{bmatrix} \tag{2.13}$$

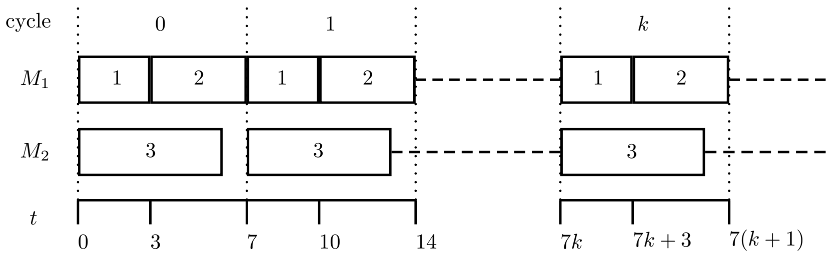


Figure 2.2: Optimal production scheme of example 2.1.

Systems theory goes hand-in-hand with control theory. Control theory emphasizes the concepts of steering non-autonomous systems through control variables, usually denoted by u. A *non-autonomous max-plus linear system* is described by state equation (2.14).

$$x(k) = \left(\bigoplus_{\mu=\hat{\mu}}^{\mu_{\max}} A_\mu \otimes x(k-\mu) \right) \oplus B \otimes u(k), \quad k \geq 0 \tag{2.14}$$

Here $B \in \mathbb{R}_\epsilon^{n \times m}$ and $\hat{\mu} \in \{0, 1\}$ determines whether the systems is implicit or explicit. Now $u(k)$ is the *input variable* of the k-th operation. In max-plus systems, u often is a reference signal, where it denotes the earliest starting time that operation can begin processing.
Likewise, a max-plus system can also define an *output variable*. The meaning of this variable is totally up to the interpretation of the real system. Usually, in real applications, the output is a signal produced by a sensor. The general output of a max-plus system is defined by the following equation.

$$y(k) = \bigoplus_{\mu=\hat{\mu}}^{\mu_{\max}} C_\mu \otimes x(k-\mu), \quad k \geq 0 \tag{2.15}$$

Here, C_μ is a matrix of $\mathbb{R}_\epsilon^{h \times n}$, where h is the number of outputs. For this report, only the case where the output fully depends on the state variables is considered. In some systems, the output also depends on the input via a *feed-through* term. This is the direct influence of the input on the output. In this report that term will be neglected for simplicity.

2.1 Switching max-plus systems

The structure of the MPL system is fixed over time. Sometimes an application requires that the dynamics of the MPL take different forms in different scenarios. The different scenarios are called the *modes* of the system. This can be modeled by introducing additional discrete dynamics. Therefore the switching max-plus linear (SMPL) system model is introduced as in van den Boom et al. [50], defined by

$$x(k) = \left(\bigoplus_{\mu=\hat{\mu}}^{\mu_{\max}} A_\mu(\phi(k), k) \otimes x(k-\mu) \right) \oplus B(\phi(k), k) \otimes u(k), \quad k \geq 0 \tag{2.16}$$

Again, $\hat{\mu}$ determines whether the systems is implicit or explicit. The function ϕ describes the switching rules that depend on the previous state, previous switching rule, input and operation-cycle. Formally it is a function of the form

$$\phi(k) = \varphi(x(k-1), \phi(k-1), u(k), v(k)) \tag{2.17}$$

Here, $v(k)$ is the input on the switching dynamics. Distinction of u and v allows the model to separate inputs that affect what type of dynamics are used from the influence of the input on the selected dynamics. Usually v is a binary variable in the max-plus sense. The reason that the system matrices also depend on k in addition to $\phi(k)$ in (2.16) is that the matrix parameters of the same mode might still differ in the sense that an operation might take longer in a different cycle.

Lemma 2.2. (As in van den Boom et al. [50]) *Let an SMPL linear system be described by equation* (2.16) *with* $\hat{\mu} = 0$ *such that the system is implicit. If* $\mathcal{G}(A_0(\phi(k)))$ *has no positive circuits for every* $\phi(k)$ *then the description*

$$x(k) = A_0^\star(\phi(k), k) \otimes \left(\bigoplus_{\mu=1}^{\mu_{\max}} A_\mu(\phi(k), k) \otimes x(k-\mu) \oplus B(\phi(k), k) \otimes u(k) \right), \quad k \geq 0 \tag{2.18}$$

is an explicit form of this SMPL system.

The proof is given by Baccelli et al. [7]. Explicit forms of system descriptions make the calculation of a next operation-cycle from the input easier, so it is usually preferred to have a SMPL system in the explicit form.

Besides input and state variables, most system models also contain an output variable. In SMPL systems the output y usually denotes the timing of end products. What y is depends completely on the model. The general model for the output signal is defined as

$$y(k) = \bigoplus_{\mu=0}^{\sigma_{\max}} C_\mu(\phi(k), k) \otimes x(k-\mu), \quad k \geq 0 \tag{2.19}$$

Some system models also let $y(k)$ depend on $u(k)$, but for simplicity this is neglected in equation (2.19). An example of a SMPL system including output is given in Appendix A.1 page 100.

Sometimes, a model allows events of a certain operation-cycle to happen after the next cycle has already begun. It should be clear from the SMPL model if this is allowed or not. In the case that it is possible, one can replace the counter from $\hat{\mu}$ to $\mu_{\max}$ with a subset of indices which are allowed to be negative. The SMPL system equation becomes as described in equation (2.20).

$$x(k) = \left(\bigoplus_{\mu \in I_k} A_\mu(\phi(k), k) \otimes x(k-\mu) \right) \oplus B(\phi(k), k) \otimes u(k), \quad k \geq 0 \tag{2.20}$$

Here, I_k is a subset of $\mathbb{Z}$. It It is important that the model in equation (2.20) allows a solution to exist.

2.2 Routing, ordering and synchronization

There are different ways to model discrete event systems in the SMPL system framework, a model does not have to be unique. There are some general ideas of modelling some system properties in the paper of Van den Boom et al. [50]. The paper defines a simple way to model the routing, ordering and synchronization of events. These properties are mainly used in scheduling models (Pinedo [44]). Recall that the vector $v(k)$ is the vector containing all binary control variables for switching mechanics of the SMPL system.

The *route* of a job j is an ordered series of processors. The operations of j occur in order of the route, and the job is completed when its route is completed. There can be more than one route, the length of a route can vary and some routes can have coinciding processors. In the example of Appendix A.1 on page 100 the possible routes are $1 \to 2$ and $1 \to 3$. Let L_j be the set of possible routes for job j, and let $R_j \subseteq M$ be the set of processors that can possibly be in a route of j. It does not make sense to model state variables $x_{j,i}(k)$ if processor i is not in R_j since the operation is in that case not present in a route of job j.
Let $M_j(l)$ be the set of processors that route $l \in L_j$ traverses. Considering the routing set R_j there are two cases. In the case of *fixed processors* every route of job j traverses through the same set of processors. This means that for every $l \in L_j$ the set $M_j(l)$ is always equal to R_j. In the *variable processors* case, a route does not contain fixed processors. In this case it is possible that a route for job j uses a certain processor, while another route for j does not uses that processor. Notice that example A.1 is a case with variable processors.

Suppose there are $|R_j|$ operations of job j that have to be processed each according to a route. Let $p_{j,i}(k)$ be its time it spends on processor i in cycle k. Let processor order $2 \to 3 \to 1$ be a possible route for this job. It can be determined that the linear inequality

$$\begin{bmatrix} x_{j,1}(k) \\ x_{j,2}(k) \\ x_{j,3}(k) \end{bmatrix} \geq \begin{bmatrix} \epsilon & \epsilon & p_{j,3}(k) \\ \epsilon & \epsilon & \epsilon \\ \epsilon & p_{j,2}(k) & \epsilon \end{bmatrix} \otimes \begin{bmatrix} x_{j,1}(k) \\ x_{j,2}(k) \\ x_{j,3}(k) \end{bmatrix} \tag{2.21}$$

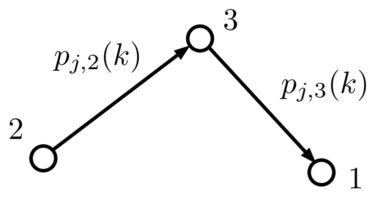


Figure 2.3: The route of system equation (2.21).

must hold if this route is selected. Denote $x_j(k) \geq A^r_{j,l}(k) \otimes x_j(k)$ as a simplification for equation (2.21), where l denotes the current route and x_j is the vector containing the events of job j. Notice that it is important that $A^r_{j,l}$ is chosen such that the route it provides is feasible. It cannot

have more than $m-1$ elements different from ϵ and there can be no cycles in the route.
Now let $v_{j,l}(k) \in \mathbb{B}_{\max}$ be a binary variable that determines if this route is chosen. This is different from example A.1 where there is only one switching variable for two routes. The difference in number of switching variables and how many are at least required will be discussed in section 5. Since there can be multiple routes, let each route correspond to a binary variable $v_{j,l}(k)$ which is equal to 0 if job j will follow route l in cycle k and it is ϵ otherwise. Note that there is exactly one route l such that $v_{j,l}(k) = 0$ and for any other routes $\hat{l}$ it has to hold that $v_{j,\hat{l}}(k) = \epsilon$. Other parameterizations with less variables are discussed in section 5. Now define the system routing matrix of job j as in equation (2.22).

$$A^r_j(v(k),k) := \bigoplus_{l \in L_j} \Big(v_{j,l}(k) \otimes A^r_{j,l}(k) \Big) \tag{2.22}$$

The constraint $x_j(k) \geq A^r_j(v(k),k) \otimes x_j(k)$ now contains the constraint of equation (2.21) only for the selected route. The general constraint for routes of all jobs can now easily be determined, and is given by the inequality

$$\begin{aligned} x(k) = \begin{bmatrix} x_1(k) \\ x_2(k) \\ \vdots \\ x_n(k) \end{bmatrix} &\geq \begin{bmatrix} A^r_1(v(k),k) & \mathcal{E} & \cdots & \mathcal{E} \\ \mathcal{E} & A^r_2(v(k),k) & \cdots & \mathcal{E} \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{E} & \mathcal{E} & \cdots & A^r_n(v(k),k) \end{bmatrix} \otimes \begin{bmatrix} x_1(k) \\ x_2(k) \\ \vdots \\ x_n(k) \end{bmatrix} \\ &= A^r(v(k),k) \otimes \begin{bmatrix} x_1(k) \\ x_2(k) \\ \vdots \\ x_n(k) \end{bmatrix} \end{aligned} \tag{2.23}$$

Before implementing the routing constraints in the SMPL system, first the system matrices for ordering and synchronization are defined, similar to the matrices defined in van den Boom et al. [50].
With multiple jobs each traversing their own route, it can occur that multiple operations end up on the same processor simultaneously. To make sure the model will not allow operations to be processed simultaneously on the same processor ordering constraints are added to the system. Define the square matrix $P_\mu(v(k),k)$ with a row and column for each operation e. Recall that $i(e)$ is the processor of operation e and $j(e)$ is the job of operation e. The matrix is now defined by

$$[P_\mu(v(k),k)]_{e_1,e_2} = \tag{2.24}$$

$$\begin{cases} 0 & \text{if } i(e_1) = i(e_2) \text{ and jobs } j(e_1) \text{ in cycle } k, j(e_2) \text{ in cycle } k-\mu \text{ take a route containing } i(e_1) \\ \epsilon & \text{else} \end{cases}$$

Notice that P_μ is dependent on $v(k)$ and k because jobs may or may not traverse a specific processor in a different route, or be different in an alternative cycle. However, the dependency on the routing variables $v(k)$ and cycle k drop out in the case of fixed processors because the set of processors for job j is always the same. In this case it can be seen that this matrix is equal to

$$[P_\mu]_{e_1,e_2} = \begin{cases} 0 & \text{if } i(e_1) = i(e_2) \\ \epsilon & \text{else} \end{cases}$$

Now to define the order of operations that spend time on a shared processor define the matrix $Q_\mu(v(k), k)$ as in equation (2.25).

$$[Q_\mu(v(k), k)]_{e_1,e_2} = \begin{cases} 0 \text{ if } i(e_1) = i(e_2) \text{ and } j(e_1) \text{ in cycle } k \text{ is scheduled after } j(e_2) \text{ in cycle } k-\mu \\ \epsilon \text{ else} \end{cases} \tag{2.25}$$

Usually the binary vector $v(k)$ contains control variables to alter the value of Q_μ such that the order of operations can be controlled. The final matrix required is $H_\mu(k)$ and denotes the time between events such that the operations are not processed simultaneously.

$$[H_\mu(k)]_{e_1,e_2} = \begin{cases} h_{\mu,e_1,e_2}(k) & \text{if } R_{j(e_1)} \cap R_{j(e_2)} \neq \emptyset \text{ and } i(e_1) = i(e_2) \\ \epsilon & \text{else} \end{cases}$$

Here, $h_{\mu,e_1,e_2}(k)$ denotes the separation time between operations e_1 in cycle k and e_2 in cycle $k-\mu$. The system ordering matrix can now be determined, and is given by equation (2.26).

$$A^o_\mu(v(k), k) = P_\mu(v(k), k) \odot Q_\mu(v(k), k) \odot H_\mu(k) \tag{2.26}$$

The ordering constraints are now given by

$$x(k) \geq A^o_\mu(v(k), k) \otimes x(k-\mu), \quad \text{for } \mu = \hat{\mu}, \ldots, \mu_{\max}$$

The last type of system inequality is the synchronization of events. Synchronization means that some events can only occur after some other events have occured. Define

$$[A^s_\mu(v(k), k)]_{e_1,e_2} = \begin{cases} 0 \text{ if operation } e_2 \text{ in cycle } k-\mu \text{ has to start before operation } e_1 \text{ in cycle } k \\ \epsilon \text{ else} \end{cases}$$

The inequality

$$x(k) \geq A^s_\mu(v(k), k) \otimes x(k-\mu), \quad \text{for } \mu = \hat{\mu}, \ldots, \mu_{\max}$$

now follows as the synchronization constraint. All constraints can now be added together by max-plus operations and the new system will be defined by equations (2.27) and (2.28).

$$x(k) \geq A_\mu(v(k), k) \otimes x(k-\mu) \tag{2.27}$$

$$A_\mu(v(k), k) = A^r(v(k), k) \oplus A^o_\mu(v(k), k) \oplus A^s_\mu(v(k), k) \tag{2.28}$$

Notice that the inequality sign in equation (2.27) should be replaced with an equality sign whenever it is given that any operation starts as soon as the processor is available. The system of equation (2.27) can be extended to a non-autonomous form (in the sense of continuous variables) in the same way as in section 2.1. An example of a SMPL system model constructed via routing and ordering is shown in Appendix A.2 page 102.

When a max-plus linear system is given and $A_\mu(\phi(k), k) \in \mathbb{R}^{n\times n}_\epsilon$ the *order* of this system is n. When modeling max-plus linear systems, the question arises whether a certain model is of minimal order. This report does not consider minimal realization theory of max-plus systems, but it is worth mentioning that for any real implementations one should always be able to argue the use of a model order. For more information on the minimal realization one can read De Schutter & De Moor [18].

2.3 Scheduling problems

Max-plus systems are closely related to scheduling problems. Scheduling is a mathematical approach of deciding, when and what job should be processed by what processor. In this thesis different scheduling classes will be compared to classes of max-plus systems. The motivational purpose is to apply any knowledge of scheduling problems to controlling max-plus systems. Scheduling theory has been analyzed by many mathematicians to optimize decision making in certain applications. Researchers have not only classified scheduling problems and their complexity (Appendix B.3 page 114), they have also developed algorithms to solve classes of scheduling problems. Pinedo [44] describes an overview on basics and the progress made in scheduling problems.

Formally, a class of scheduling problems as described in Pinedo [44] is a triplet denoted by $\alpha|\beta|\gamma$. Here, α is the processing environment, β denotes the set of limiting constraints and γ describes the objective function. Some examples of these three classifications are shown in this section.

Processing environment

- Identical parallel processing (Pm). There are m processors. For each processor the processing time of job j is p_j.
- Parallel processing with variable speeds (Qm). There are m processors, processor i works with speed v_i. The processing time of job j performed by processor i is p_j/v_i.
- Unrelated parallel processors (Rm). There are m processors. Processing job j by processor i takes $p_{i,j}$ time.
- Flow shop (Fm). All jobs should be processed on all m processors, in a specific single route. For every processor the first job in is the first job out of the processor.
- Job shop (Jm). Each job has its own route on a subset of the m processors.
- Open shop (Om). Each job has to be processed on a subset of m processors. The route for each job can be determined under the constraints.

The notations Pm, Qm, Rm, Fm, Jm and Om are also referred as the classes of scheduling problems. In Pm, Qm and Rm the jobs are single and independent operations. The processing environment of SMPL systems depends on the routing structure of the problem. In case any job has only a single route, the system is a job shop (Jm) environment. In the case there are multiple routes for jobs, the system becomes more of a restricted open shop (Om) environment[2]. Since usually not every route is feasible, it is not exactly the same as an open shop environment. In most applications the SMPL model has a few routes, which makes the processing environment a mixture between an open shop and job shop environment. It is therefore useful to look at the literature on both environments to use for control.

Constraints

- Release dates (or a timetable reference) ($r_{j,i}$). An operation comes available at time r_j and cannot be processed earlier.
- Deadlines ($\bar{d}_j$). Every job must be completed before its deadline. If this is not the case the schedule is infeasible.

[2]Max-plus algebra and scheduling as in Pinedo [44] are two different approaches for describing discrete event timing. The link between the two approaches has been established by the writer of this thesis for the purpose of getting more insights from the literature.

- Preemption (*prmp*). It is possible to interrupt a job that is being processed and work on it later.
- Precedence constraints (*prec*). Each job j has a set of predecessor jobs that have to be completed in order to start job j.
- Processor eligibility restrictions (M_j). Not every job can be processed by every processor. M_j is the set of processors that can do job j.

The constraints highly depend on the application of the SMPL system. The synchronization constraints translate to adding precedence constraints (*prec*) in the scheduling form. If any reference or timetable is supposed to be tracked, ($r_{j,i}$) should be included in the scheduling form of the problem. For this thesis, no problems considering preemption or deadlines are considered.

In this thesis the objective function of the scheduling problem is always considered to be a minimization problem.

Objective function

- Makespan ($\mathcal{C}_{\max}$). Complete all jobs as soon as possible. So minimize $\max_j\{\mathcal{C}_j\}$ where $\mathcal{C}_j$ is the completion time of job j.
- Maximum lateness ($\mathcal{L}_{\max}$). Every job has a *due date* d_j and lateness defined as $\mathcal{L}_j := \mathcal{C}_j - d_j$. The objective is to minimize $\max_j\{\mathcal{L}_j\}$.
- Maximum tardiness ($\mathcal{T}_{\max}$). Define $\mathcal{T}_j = \max(\mathcal{L}_j, 0)$. The objective function to minimize is the maximum of all $\mathcal{T}_j$ over all jobs j.
- Weighted sum of completion times ($\sum_j w_j\mathcal{C}_j$). Weights w_j determine the importance of each job.
- Weighted tardiness ($\sum_j w_j\mathcal{T}_j$).
- Weighted unit penalty ($\sum_j w_j\mathcal{U}_j$). Define $\mathcal{U}_j = 1$ if $\mathcal{C}_j > d_j$ and $\mathcal{U}_j = 0$ otherwise.
- Convex function (f). Minimize a convex function f over completion times, lateness, tardiness and unit penalties.

The objective function in SMPL systems depends on the control problem, which will be eleborated on in section 3. The control objective can be pretty much anything in SMPL systems.
Note that α and γ can only be assigned one description but β can take multiple constraints. It can be verified that the problem of deciding the best schedule for one cycle of the system in example 2.1 belongs to the class $Pm|M_j, prec|\mathcal{C}_{\max}$. There are identical parallel processors, hence Pm is the processing environment. However, not every processor can execute every job so the system has processor eligibility restrictions (M_j). In addition to the constraints, there are precedence constraints (*prec*) since jobs 2 and 3 can only be executed after job 1 is completed. The overall goal is to minimize $y(k)$, which is the makespan ($\mathcal{C}_{\max}$) of the system.

Pinedo [44] provides an overview of scheduling problems and their complexity. Scheduling problems have been discussed in a lot of academic researches, and there is many knowledge of many scheduling classes as a result. This thesis will compare the mathematical modeling of scheduling with the SMPL systems. If the general SMPL control problem (presented later on in section 3) can be casted into a scheduling class it follows that the knowledge of scheduling can be applied to controlling the SMPL systems.

In SMPL systems, the output of the system is usually some form of completion times of jobs. For example, in a production scheme model, the system output is usually the times that completed products leave the system. Now let the processors of job j be given by $i_j(1), \ldots, i_j(m_j)$ and let $x(k) = [x_{1,i_1(1)}(k) \ \cdots \ x_{1,i_1(m_1)}(k) \ \cdots \ x_{n,i_n(1)}(k) \ \cdots \ x_{n,i_n(m_n)}(k)]^T$ be the vector of the starting times all operations of n jobs. If the output of the system is defined as a makespan $(y(k) = \mathcal{C}_{\max}(k))$, the matrix $C_0(\phi(k), k)$ from equation (2.19) becomes independent of the switching mechanic, so denote $C_0(k)$ for the rest of this section. It must be defined as

$$C_0(k) = [p_{1,i_1(1)}(k) \ \cdots \ p_{1,i_1(m_1)}(k) \ \cdots \ p_{n,i_n(1)}(k) \ \cdots \ p_{n,i_n(m_n)}(k)]$$

Here, it is the case that $\phi(k) = k$ and $\sigma_{\max} = 0$. Notice that it now holds that

$$\begin{aligned} y(k) = C_0(k) \otimes x(k) &= \bigoplus_{j=1}^{n} \bigoplus_{m=1}^{m_j} (p_{j,i_j(m)} \otimes x_{j,i_j(m)}) \\ &= \max_{j \in J} \max_{i \in R_j} (p_{j,i}(k) + x_{j,i}(k)) = \mathcal{C}_{\max}(k) \end{aligned}$$

Likewise, if the output is a vector of completion times, take

$$C_0(k) = \begin{bmatrix} p_{1,i_1(1)}(k) & \cdots & p_{1,i_1(m_1)}(k) & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & 0 & \ddots & 0 & \ddots & 0 \\ 0 & \ddots & \ddots & 0 & p_{n,i_n(1)}(k) & \cdots & p_{n,i_n(m_n)}(k) \end{bmatrix}$$

to obtain

$$y(k) = C_0(k) \otimes x(k) = \begin{bmatrix} \bigoplus_{m=1}^{m_1} (p_{1,i_1(m)} \otimes x_{1,i_1(m)}) \\ \vdots \\ \bigoplus_{m=1}^{m_n} (p_{n,i_n(m)} \otimes x_{n,i_n(m)}) \end{bmatrix} = \begin{bmatrix} \mathcal{C}_1(k) \\ \vdots \\ \mathcal{C}_n(k) \end{bmatrix}$$

Of course, the output can also be the a vector containing the completion times as well as the makespan. In this case, define

$$C_0(k) = \begin{bmatrix} p_{1,i_1(1)}(k) & \cdots & p_{1,i_1(m_1)}(k) & 0 & \cdots & \cdots & 0 \\ 0 & \ddots & 0 & \ddots & 0 & \ddots & 0 \\ 0 & \ddots & \ddots & 0 & p_{n,i_n(1)}(k) & \cdots & p_{n,i_n(m_n)}(k) \\ p_{1,i_1(1)}(k) & \cdots & p_{1,i_1(m_1)}(k) & \cdots & p_{n,i_n(1)}(k) & \cdots & p_{n,i_n(m_n)}(k) \end{bmatrix}$$

such that

$$y(k) = C_0(k) \otimes x(k) = \begin{bmatrix} \mathcal{C}_1(k) \\ \vdots \\ \mathcal{C}_n(k) \\ \mathcal{C}_{\max}(k) \end{bmatrix}$$

defines the correct output.

In section 2 it is described how the max-plus algebra can used to model the timing of events. The SMPL system model switches between modes such that different routing and ordering can take place in the system. SMPL systems are closely related to scheduling problems (Pinedo [44]), which means insights from the literature for scheduling problems can be useful for SMPL systems. The next section will describe a control method for SMPL systems.

3 Model Predictive Control for SMPL Systems

The main objective of this section is to specify the control problem for the SMPL system. The control method uses the concept of model predictive control. First, an objective function will be given that will be used for the optimization model. Minimizing the objective function over the inputs of the system can then be transformed to an *mixed-integer linear program* (MILP). The definition of the MILP can be found in Wolsey [54] or Appendix B.1 page 110. The MILP offers the possibility to model as well the continuous and discrete input values of the SMPL model. The solution to the MILP is then used to compute the input values of the SMPL system.

The non-autonomous SMPL system described in equation (2.16) on page 15 has input signals available to control the system. In order to determine how the system is going to be controlled an objective has to be established. The determination of the input laws should depend on what the control objective is. The control objective can be pretty much anything, common examples are achieving stability, robustness, reference tracking or disturbance rejection (Leigh [37]). All can be measured in several ways. The paper of Van den Boom et al. [50] provides a generalized cost function, but for this thesis a slightly simpler version is considered.
The optimization model uses binary variables in the conventional sense, a conversion is needed to apply these binary variables. Let $v \in \mathbb{B}_\epsilon$ be a binary, variable in the max-plus sense. Then

$$v^b = \begin{cases} 0 & \text{if } v = \epsilon \\ 1 & \text{if } v = 0 \end{cases} \tag{3.1}$$

is the associated binary variable in the conventional sense. Numerically, the number ϵ cannot be used. In calculations, max-plus binary variables are therefore replaced by the approximation $v \approx \beta(1 - v^b)$ where $\beta \ll 0$ is a negative number that is very large in absolute value. The parameter β should be larger than the maximum time between two events of variable ordering such that the feasibility of the model does not change after the transformation. If beta is chosen too large, numerical issues can occur in system simulation. Choose β therefore always equal to minus an upperboud on the maximum time between two events.

3.1 Model predictive control

Described in Garcia et al. [23], *Model predictive control* (MPC) is a well known method for controlling systems of any form. It uses the *receding horizon principle*; The system behaviour is determined over a prediction horizon of N_p cycles. If t is the time where new control inputs are required, the controller looks at the behaviour of the system from time t to $t + \lambda N_p$, where λ is the time length of a cycle. An objective function is established to qualify the behaviour of the system over the prediction horizon. This objective function is minimized to find the optimal control values. From the optimal values, only the first few control variables required at time t will then be used for control. Then t moves ahead into the future to the time when next control variables are required and the process is repeated.

Remark 3.1. As mentioned in van den Boom & De Schutter [49], MPC in SMPL systems is defferent from conventional MPC because the events are related to a variable time. The MPC controller is defined to take all variables in consideration in a specified time range. However, in SMPL systems, the state variables are timings, and it is likely that there are time variables that can be either in or out of this time range. It is therefore important for MPC on SMPL systems that one establishes a rule that determines which variables are taken into account for the optimization. For timetable reference systems, one could choose to take all the variables for events supposed to occur before $t + \lambda N_p$ according to the timetable. In the absence of a timetable one can for example model the variables of events that can occur at the earliest possible time before $t + \lambda N_p$.

The behaviour of the MPC controller highly depends on the choice of the objective function. The objective function explained in this thesis will be similar to the MPC controller explained in De Schutter & van den Boom [19] and van den Boom et al. [50] to a certain extend. The objective function is denoted by $J_{N_p}(k)$, where k is the current cycle. The generalization of the objective function in this thesis used is given by equation (3.2).

$$J_{N_p}(k) = \sum_{\tau=0}^{N_p-1} \left(\sum_{i=1}^{h} \delta_{i,\tau} y_i(k+\tau) \right. \tag{3.2}$$
$$\left. + \sum_{j \in J} \sum_{i \in R_j} \left(\kappa_{j,i,\tau} x_{j,i}(k+\tau) + \alpha_{j,i,\tau} \max(x_{j,i}(k+\tau) - d_{j,i}(k+\tau), 0) \right) \right)$$

Here, $\delta_{i,\tau}, \kappa_{j,i,\tau}, \alpha_{j,i,\tau}$ are scalars that determine the relative weight between certain variables. The variable $d_{j,i}(k+\tau)$ is the due date of an operation (j,i) in cycle $k+\tau$. This event should occur at least at time $d_{j,i}(k+\tau)$ otherwise the objective function is penalized with value $\alpha_{j,i,\tau}(x_{j,i}(k+\tau) - d_{j,i}(k+\tau))$. The output of max-plus systems is usually defined by the system designer, and can be chosen to be the system makespan for example. For this argument of generalization the vector y is used in the overall cost function.

For optimal behaviour, one should choose N_p as large as possible. However, large N_p comes with large computational complexity (Appendix B.3 page 114) which is not desired. In lots of real applications the next optimal input has to be computed before it is used in order to achieve *on-line performance*. On-line performance is the working of the controller in real time, as described in Garcia et al. [23]. The prediction horizon is therefore decreased to a number small enough that ensures that the optimization process is solved in time. This will result however in worse performance, since the model predictive controller takes less future events into account. In this thesis the main discussion will be focused around the reduction of the computational time of the optimization process for MPC on SMPL systems. If any advantages can be found that actually decrease the computational load on the optimization problem, a larger prediction horizon N_p can be chosen for on-line performance which should result in better performance of the model predictive controller in practice.
Optimizing the objective function J_{N_p} does not go without constraints. All state and input variables have to be chosen such that the system matrices of equation (2.16) still describe the relations between the variables. The SMPL system model can therefore be transformed into the optimization constraints, such that the optimal solution will satisfy the system dynamics.

In practice, the processing time of certain operations times will not be fixed in time. Van den Boom & De Schutter [49] explain how the true state of the system differs from the estimated state of the system in MPC for SMPL systems. The problem is that an event can occur or not occur while was expected otherwise. This may cause the computed system output to be different from the real system output. For example, if x is the time a train departs from station T, it must first arrive at station T. When it is heading for station T the schedule for which the train departs at that station is already determined, but it might happen that the train arrives later than expected and can thus not depart at the computed time x. The time an event is planned to happen can therefore differ in time. So in practice it is common to let the SMPL system matrices and variables of equation (2.16) depend on time t, such that it denotes the planned system at time t. This is a small extension of the problem of controlling SMPL systems, it is essential for practical uses but does not change the theory behind the computational process of the MPC controller. In this thesis the time variable will therefor be left out of the system representations.

3.2 Recasting the MPC problem as an MILP

To solve the MPC optimization problem the problem is transformed into the form of a *mixed-integer linear program* (MILP). Section B provides an overview of what the general MILP is and how it can be solved. The idea is to write the SMPL system control problem in the form of problem (B.3)-(B.4) on page 110. The objective is to transform the SMPL model (2.16) to the linear constraints (B.4) and the MPC objective function (3.2) to the MILP objective function (B.3).

The objective function to be minimized is $J_{N_p}(k)$ as defined in equation (3.2) and can be written in linear form. Note that the objective function is linear in the states and output of the system. A trick can be done for the part of the objective function that describes the tardiness of all events, such it becomes a linear function. Let $\mathcal{T}_{j,i}(k)$ be the tardiness of the event triggering operation (j,i) in cycle k. Let $x_{j,i}(k+\tau) - \mathcal{T}_{j,i}(k+\tau) \leq d_{j,i}(k+\tau)$ and $\mathcal{T}_{j,i}(k+\tau) \geq 0$ be additional constraints for all $j \in J$, $i \in R_j$ and $\tau \in \{0, \ldots, N_p - 1\}$ such that the minimization of $J_{N_p}(k)$ will force $\mathcal{T}_{j,i}(k+\tau) = \max(x_{j,i}(k+\tau) - d_{j,i}(k+\tau), 0)$. When assigning proper orders to the used vectors, the linear objective function now becomes

$$J_{N_p} = \delta^T y + \kappa^T x + \alpha^T \mathcal{T} \tag{3.3}$$

Now if $\alpha \geq 0$, all the elements of $\alpha^T \mathcal{T}$ are positive since $\mathcal{T} \geq 0$. This means that if $x_{j,i}(k+\tau) \leq d_{j,i}(k+\tau)$ (an event occurs before or at its due date) that $\mathcal{T}_{j,i}(k+\tau)$ is forced to be 0 by minimization of $J_{N_p}(k)$. The example from Appendix A.2 is transformed to an MILP is Appendix A.3.

The next step is to convert the SMPL system equation (2.16) into linear constraints. This can be done using the binary variable transformation of equation (3.1). The constraints all consist of max operations, which can be transformed into linear form with the same steps performed when transforming the tardiness $\mathcal{T}_{j,i}(k)$. Notice that the equality nature of the constraint can be implemented by adding the constraint with the $\leq$ sign as well as the $\geq$ sign.

Remark that in this form all starting times for events will be as soon as possible. Whether this is supposed to be the case depends on the application. In certain scheduling problems it sometimes is better to wait a few time units before an event before triggering an event in order to optimize the objective function. In max-plus systems, this can be implemented by letting the event variables $x(k)$ be larger or equal to right-hand side of the given system equation, instead of denoting the equality sign. In the MILP this changes only by removing one direction of the system inequality, so it can still be written as an MILP. Whether it is allowed to postpone operations while they could have started should therefore always be clear from the model problem description.

This is one way to model to model the MPC problem on SMPL systems into an MILP. A more detailed explanation and an extension of the objective function (3.2) can be found in De Schutter & van den Boom [19].

Notice that when transforming the MPC control problem for SMPL systems one always ends up with matrix A from formulation (B.4) having only entries 0 and ± 1. This follows from the nature of the $\otimes$ operation in system equation (2.16).

3.3 The MPC-SMPL problem specification

The core of this thesis lies in solving the MPC problem for the general SMPL model framework. Sections 2 & 3 are summarized into a single problem specification, namely the MPC-SMPL problem.

Definition 3.2. (Main problem specification) *Given are*

- *A set J of n jobs and a set M of m processors.*
- *A set of routes L_j for each job $j \in J$.*
- *A prediction horizon N_p, a current cycle k and a number $\mu_{\max} \in \mathbb{N}_+$ that denotes the maximum difference in cycles for which operations can interchange order.*
- *Matrices $P(k), \dots, P(k+N_p-1) \in \mathbb{R}_+^{n\times m}$, with entries $p_{j,i}(k)$ denoting the processing time of operation (j,i) in cycle k. If i is not a processor of all routes $l \in L_j$ for a given $j \in J$, then operation (j,i) is non-existent and $p_{j,i}(k) = 0$.*
- *A specified structure of h outputs denoted by $y(k)$ that depends on the events $x(k)$ in cycle k.*
- *Release dates or also referred as a timetable references $r_{j,i}(k+\tau)$ for each event $x_{j,i}(k+\tau)$ and $\tau = 0, \dots, N_p-1$. They can be left out by setting $r_{j,i}(k+\tau) = -\infty$.*
- *Due dates $d_{j,i}(k) \in \mathbb{R}$ for each event $x_{j,i}(k+\tau)$ and $\tau = 0, \dots, N_p-1$. Due dates can be neglected by setting $d_{j,i}(k+\tau) = \infty$.*
- *A synchronization set S of pairs of events.*
- *Positive weights δ, κ and α.*

The MPC-SMPL problem is to find a route for each job $j \in J$ and an ordering for each processors $i \in M$ such that the resulting feasible schedule of all events between time t and $t+\lambda N_p$ minimizes the objective function (3.2)*.*

Notice that the problem defined by definition 3.2 specifies only a part of the cyclic process. The problem is repeatedly solved in on-line performance. The MPC controller only schedules the first few events by the solution of an instance of the MPC-SMPL problem and then the problem shifts in time to new instance of the MPC-SMPL problem.
Whenever a feasible routing and ordering is chosen, the values of the starting times and output should roll-out naturally by the system model. The question arises whether every combination of routes and orderings is feasible. Example 3.3 shows that this is not the case.

Example 3.3. Suppose $J = M = \{1, 2\}$ and a single cycle. The selected routes are $l_1 = (1, 2)$ and $l_2 = (2, 1)$. Now decide to order job 2 before job 1 on processor 1 and job 1 before job 2 on processor 2. It is now the case that every operation waits for another operation to complete, i.e. the combined order of operations (denoted (j,i)) is $(2,2) \to (2,1) \to (1,1) \to (1,2) \to (2,2)$. Hence, the given combination of routing and ordering is infeasible.

The general MPC-SMPL problem is $\mathcal{NP}$-hard (Appendix B.3 or Arora & Barak [5]), which can be shown by a reducing the $Om||f$ to the MPC-SMPL problem. The problem $Om||f$ is $\mathcal{NP}$-hard, which was shown by Williamson et al. [53]. Notice that the MPC-SMPL problem is a generalization of multiple scheduling classes, of which not all are $\mathcal{NP}$-hard. For example, the $O2||\mathcal{C}_{\max}$ problem is also a special instance of the MPC-SMPL problem and can actually be solved in polynomial time, as was shown by Gonzalez & Sahni [27].

4 Applications of SMPL systems

To get more of a feel of an example of a real SMPL system this section models two applications into the SMPL system model. The first application is a part of the dutch railway network, wherte the arrival and departure times of trains are modelled. Most of the modeling is inspired by Kersbergen [34]. The second application is a container terminal. Here, a number of containers have to be moved efficiently to their destination.

4.1 The railway network

One of the applications of the SMPL system is the railway network. Kersbergen [34] showed how the Dutch railway system can be modelled into the SMPL system framework. In this thesis, not the entire dutch railway network but only a part will be modelled for simulation. The part that will be implemented is the flow of trains between Amsterdam and The Hague (also known as Den Haag in Dutch). To simplify the simulation, only the trains in the direction of The Hague are modelled. A schematic overview of this part of the railway network can be found in figure D.1 in Appendix D on page 126.
The structure of the network is provided by Sporenplan [48], it contains the interconnections of all the tracks. The actual railway network will be modelled in this section. The train timetable for modeling was the timetable applied by Nederlandse Spoorwegen [42] in june 2019. The timetable can be found on the website of Nederlandse Spoorwegen [42].
Each 30 minutes (which is the cycle time λ), there is a flow of 17 trains on this part of the network: 5 intercity trains (IC), 2 intercity direct trains (ICD) and 10 sprinter trains (SP). The 17 trains are given in table 4.1 along their first departure and last arrival time in minutes past the start of the cycle.

type	departure	destination	times		ID
IC	Amsterdam Centraal	Den Haag Centraal	19	72	1
	Amsterdam Centraal	Den Haag HS	04	57	2
	Amsterdam Zuid	Den Haag HS	35	72	3
	Amsterdam Zuid	Den Haag Centraal	19	56	4
	Den Haag Centraal	Den Haag HS	47	53	17
ICD	Amsterdam Centraal	Schiphol	08	23	5
	Amsterdam Centraal	Schiphol	22	35	6
SP	Amsterdam Zuid	Hoofddorp	11	23	7
	Amsterdam Zuid	Hoofddorp	29	41	8
	Amsterdam Centraal	Haarlem	11	30	9
	Amsterdam Centraal	Haarlem	26	45	10
	Amsterdam Centraal	Den Haag Centraal	12	82	11
	Haarlem	Den Haag Centraal	25	67	12
	Amsterdam Centraal	Hoofddorp	00	12	13
	Amsterdam Sloterdijk	Leiden Centraal	04	37	14
	Den Haag Centraal	Den Haag HS	67	70	15
	Den Haag Centraal	Den Haag HS	53	56	16

Table 4.1: The time table of trains from Amsterdam to the Hague.

The model implemented in MATLAB for simulation in this thesis is more detailed than as in Kersbergen [34]. The model is extended with the ability of trains to pick an alternative platform of departure within a train station. In SMPL systems, this translates as giving trains alternative routes to follow. Note that the sequence of train stations will never change, only the platforms where the train will stop can sometimes change. Since a train can only do one platform per train station the SMPL system is of the variable processors case, since a train cannot stop at multiple platforms within a single station. The platforms and processor indices can be found in table D.1 in Appendix D on page 127.
The other decision mechanic is the choice of ordering trains. Orderings of trains can only be interchanged between a few specific train stations. The tracks where trains can interchange order lie between the following pairs of train stations.

- Amsterdam Centraal & Amsterdam Sloterdijk
- Schiphol & Hoofddorp
- Halfweg-Zwanenburg & Haarlem Spaarnwoude
- Haarlem & Heemstede-Aerdenhout
- Hillegom & Voorhout
- Voorhout & Leiden Centraal
- Sassenheim & Leiden Centraal
- Den Haag Mariahoeve & Den Haag Laan van NOI
- Den Haag Laan van NOI & Den Haag HS

Note that if two tracks come together the order of trains that come from the same track before the merging must be the same as the order between those trains after the merging.
The events of the system are trains arriving, leaving or passing station platforms. The main difference of the railway model from the regular SMPL system defined in 3.3 is that ordering and routing use different processing times as operations. When a train leaves a platform it takes some time before a new train can enter that platform, but the time it takes to get the departed train to the next platform is another processing time. In the MILP this translates to different processing times in routing and ordering constraints. There are also different processing times between arrival and departure. The minimum processing times for the time a train stops on a platform is given in 4.2.

Train type	IC	ICD	SP
Stopping time (s)	45	60	30

Table 4.2: Minimum stopping times per type of train.

The stopping times may of course be longer than expected due to passenger hold ups. It could be that way more people have to switch trains than expected which can cause a delay.
In the railway model, the jobs are the trains, and the processors are station platforms. The events are placed into 3 categories for the railway network. The events are times of arrival $a_{j,i}(k)$, departure $d_{j,i}(k)$ and passing a platform $t_{j,i}(k)$ if a train does not stop at that station. The reason the passing times are implemented is because trains cannot overtake other trains on every part of the railway network. By formulating the ordering variables correctly, the passing of train stations

occurs in the same order until it is possible to overtake. Therefore, the location of the overtake matters for the total delay of the trains.
The first important constraints are the traveling time constraints of the trains. The traveling time depends on if the train has to stop on the current or next station, that is why it is different per train. The constraints depend on the train, but there are 4 different traveling time constraints, given in (4.1-4.4). The value $p_{j,i_1,i_2}(k)$ denotes the travel time of train j from platform i_1 to i_2.

$$a_{j,i_2}(k) \geq d_{j,i_1}(k) + p_{j,i_1,i_2}(k) \tag{4.1}$$
$$a_{j,i_2}(k) \geq t_{j,i_1}(k) + p_{j,i_1,i_2}(k) \tag{4.2}$$
$$t_{j,i_2}(k) \geq d_{j,i_1}(k) + p_{j,i_1,i_2}(k) \tag{4.3}$$
$$t_{j,i_2}(k) \geq t_{j,i_1}(k) + p_{j,i_1,i_2}(k) \tag{4.4}$$

Other important constraints are the stopping times of the trains. As shown in table 4.2 there are 3 different processing times for this constraint. If the platform i is an obligated platform for train j, equation (4.5) gives the according stopping time constraint.

$$d_{j,i}(k) \geq a_{j,i}(k) + p_{j,i}(k) \tag{4.5}$$

The use of notation $p_{j,i}(k)$ instead of p_j is used such that different delays can be simulated for different platforms and cycles. At some train stations, it is possible for a train to stop at an alternative platform. In this case, a function of binary routing variables needs to be added to the right-hand-side of (4.5) to ensure the constraints only apply when the corresponding route is selected. To do this for the stations where this is the case, one can implement the constraints of Appendix D.1. The ordering of trains on the railway network is constructed on every platform. The processing times for train ordering depend on whether the trains do depart, arrive or pass the platform. The possible processing times are given in table 4.3.

type of events	headway time (s)
departure $\rightarrow$ arrival	90
departure $\rightarrow$ pass	75
pass $\rightarrow$ arrival	75
pass $\rightarrow$ pass	60

Table 4.3: The headway times for ordering in the railway network.

Let the processing times for ordering be denoted by $p_{j_1,j_2,i}(k)$, where j_1, j_2 are the ordered trains and i is the platform. Suppose ordering variable $v^o_{j_1,j_2,i}(k)$ determines whether train j_1 is before train j_2 on platform i in cycle k. If both trains stop at the platform, the ordering constraints are

$$a_{j_2,i}(k) \geq d_{j_1,i}(k) + p_{j_1,j_2,i}(k) + \beta(1 - v^o_{j_1,j_2,i}(k)) \tag{4.6}$$
$$a_{j_1,i}(k) \geq d_{j_2,i}(k) + p_{j_2,j_1,i}(k) + \beta v^o_{j_1,j_2,i}(k) \tag{4.7}$$

Now if only one train stops at platform i and the other only passes i, say only train j_1 stops, the constraints are

$$a_{j_2,i}(k) \geq t_{j_1,i}(k) + p_{j_1,j_2,i}(k) + \beta(1 - v^o_{j_1,j_2,i}(k)) \tag{4.8}$$
$$t_{j_1,i}(k) \geq d_{j_2,i}(k) + p_{j_2,j_1,i}(k) + \beta v^o_{j_1,j_2,i}(k) \tag{4.9}$$

If neither of the trains stop but only pass station i the constraints become as in (4.10) and (4.11).

$$t_{j_2,i}(k) \geq t_{j_1,i}(k) + p_{j_1,j_2,i}(k) + \beta(1 - v^o_{j_1,j_2,i}(k)) \tag{4.10}$$

$$t_{j_1,i}(k) \geq t_{j_2,i}(k) + p_{j_2,j_1,i}(k) + \beta v^o_{j_1,j_2,i}(k) \tag{4.11}$$

Due to the possibility for some trains to sometimes choose an alternative platform the railway model is of the variable processor case. This means that constraints (4.6)-(4.11) are only sufficient if there is no alternative to platform i. If train j does not visit platform i, the events $d_{j,i}, a_{j,i}$ can be neglected and are supposed to take their reference value. To get the ordering constraints correct for processors with variable routing, some terms have to be added to constraints (4.6)-(4.11) such that the β term is zero only if both trains take a route using i. The added terms are similar to the used terms in Appendix D.1, but now take both the part for j_1 and j_2.
In the model, trains cannot interchange order everywhere. In addition, when two tracks merge into one track the order of one incoming track must be the same as the the order after the merging for two trains coming from the same direction. This can be simply implemented by setting

$$v^o_{j_1,j_2,i_1}(k) = v^o_{j_1,j_2,i_2}(k) \tag{4.12}$$

where trains have no opportunity to interchange between i_1 and i_2.
On the network between Amsterdam and The Hague the trains travel according to a reference schedule. The reference schedule denotes the earliest time for a train to depart from a platform or arrive at a platform. Implementation is simply by setting

$$d_{j,i}(k) \geq r^d_{j,i}(k) \quad \forall j \in J, i \in R_j \tag{4.13}$$

$$a_{j,i}(k) \geq r^a_{j,i}(k) \quad \forall j \in J, i \in R_j \tag{4.14}$$

where $r_{j,i}(k)$ is the reference of train j on platform i. Now the objective function of the railway network is to minimize the sum of delays. The delays are implemented by setting due dates equal to the reference and then defining the delays (or tardiness) $\mathcal{T}_{j,i}(k)$ as usual.

$$\mathcal{T}^d_{j,i}(k) \geq 0, \mathcal{T}^a_{j,i}(k) \geq 0 \tag{4.15}$$

$$\mathcal{T}^d_{j,i}(k) \geq d_{j,i}(k) - r^d_{j,i}(k) \tag{4.16}$$

$$\mathcal{T}^a_{j,i}(k) \geq a_{j,i}(k) - r^a_{j,i}(k) \tag{4.17}$$

The objective function is given by

$$J_{N_p}(k) = \sum_{\tau=0}^{Np-1} \sum_{j \in J} \sum_{i \text{ is stop of } j} \mathcal{T}_{j,i}(k+\tau) \tag{4.18}$$

In order to make it easier for passengers to reach their destination, some trains wait for each other such that the passengers have the option to switch trains. In the model this is implemented as synchronization constraints. The trains that wait for each other are

- Trains 1 & 3 on Leiden Centraal
- Trains 2 & 4 on Leiden Centraal
- Trains 2 & 16 on Den Haag HS
- Trains 3 & 15 on Den Haag HS

Therefore, constraints (4.19)-(4.22) are added.

$$d_{1,i_1}(k) = d_{3,i_2}(k) \quad i_1, i_2 \in \{27, 28, 29, 30\},\ i_1 \neq i_2. \tag{4.19}$$
$$d_{2,i_1}(k) = d_{4,i_2}(k) \quad i_1, i_2 \in \{27, 28, 29, 30\},\ i_1 \neq i_2. \tag{4.20}$$
$$d_{2,42}(k) = d_{16,41}(k) \tag{4.21}$$
$$d_{3,42}(k) = d_{15,41}(k) \tag{4.22}$$

Notice that synchronized trains cannot stop at the same platform at Leiden Centraal.

4.2 The container terminal

Another application of the SMPL system model is the container terminal. In a container terminal a lot of containers have to be transported to their next destination. Containers are initially present in a stack and their final destination is a certain container ship. Each container can be be moved by an *automatic guided vehicles* (AGV). First, they have to be loaded from the stack to the AGV by a *stack crane*. The AGV then has to move to the right *quay crane* that will unload the container from the AGV and load it on a *container ship*. A schematic drawing of the situation is shown in figure 4.1.

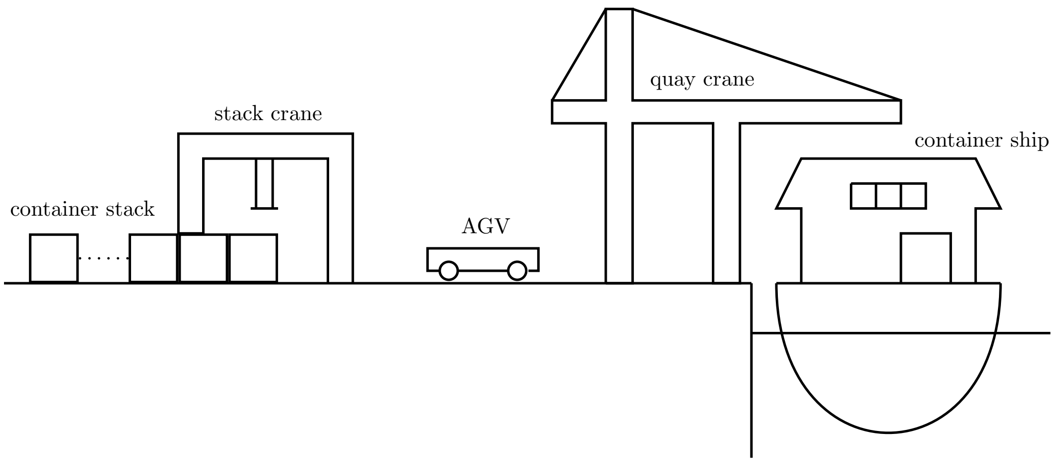


Figure 4.1: A schematic drawing of the container terminal.

In the real container terminal, there are multiple quay cranes, AGV's and stack cranes. The number of quay cranes is denoted by N_q, the number of AGV's by N_a and the number of stack cranes by N_s. Each cycle k a set of containers becomes available in a stacked order from the stack cranes. The containers of each cycle have to be distributed over $N_c \leq N_q$ ships via the quay cranes, where each container has a fixed destination.
The objective of the system is to minimize the sum of *transfer times* of each ship in each cycle. The transfer time of a ship is defined to be the time between the moment the first container of the cycle k becomes available and the time all the required containers are loaded onto the ship.

There are a number of free decisions that can be made to optimize the system. Firstly, every container can be transported by any AGV, but must be transported by exactly one. Each AGV may also choose an order in which it transports its assigned containers. Lastly, the order in which quay cranes unload the containers from the AGVs is also variable. The order in which the containers are loaded from the stacks is however fixed, because they are stacked in a certain order.

In terms of the SMPL system model, the containers do correspond to jobs. The processors are the stack cranes, AGVs and quay cranes. The system is of the form of variable processors because every container can be loaded to its ship via every AGV. The number of routes for any job is therefore equal to N_a with $L_j = \{(1), \dots, (N_a)\}$. Ordering of jobs takes place on the AGVs, the ordering on the stack cranes is fixed. From which stack crane a container is loaded onto an AGV is also fixed.
Define J to be the set of containers that has to be transferred each cycle by AGVs $a_1, \dots, a_{N_a}$. Now define $x(k)^s_{j,i}$ to be the starting time that container j in cycle k is loaded from its stack crane on AGV a_i if a_i is picking up container j. If a_i is not the assigned AGV to pick up container j define $x^s(k)_{j,i} = 0$. Let $x(k)^q_{j,i}$ be the time the quay crane starts unloading container j in cycle k from AGV a_i if that AGV is moving container j. If container j is not moved by AGV a_i then $x(k)^q_{j,i} = 0$. Now let $v_{j,i}(k)$ be the routing variable that is equal to 1 if container j in cycle k is transferred by AGV a_i and $v_{j,i}(k) = 0$ otherwise. The routing constraints are now given by

$$x^q_{j,i}(k) \geq x^s_{j,i}(k) + p^s + p_{s_j,q_j} + \beta(1 - v_{j,i}(k)),\ \forall j \in J, i = 1, \dots, N_a, k \geq 0 \tag{4.23}$$

where p^s is the loading time at a stack crane and p_{s_j,q_j} is the time for an AGV to travel from the stack crane where j is stacked to the quay crane destination of j. After dropping of a container, the AGV should decide which container is next. Define $z_{j_1,j_2,i}(k) = 1$ if containers j_1 and j_2 in cycle k are both transferred by a_i and container j_1 is transferred before j_2. The ordering constraints for the container terminal are now given by

$$x^s_{j_2,i}(k) \geq x^q_{j_1,i}(k) + p^q + p_{s_{j_2},q_{j_1}} + \beta(1 - z_{j_1,j_2,i}(k)) + \beta(2 - v_{j_1,i}(k) - v_{j_2,i}(k)) \tag{4.24}$$
$$x^s_{j_1,i}(k) \geq x^q_{j_2,i}(k) + p^q + p_{s_{j_1},q_{j_2}} + \beta z_{j_1,j_2,i}(k) + \beta(2 - v_{j_1,i}(k) - v_{j_2,i}(k)) \tag{4.25}$$
$$\forall j_1, j_2 \in J, j_1 \neq j_2, i = 1, \dots, N_a, k \geq 0$$

where p^q is the unloading time of a container at a quay crane. Now the ordering on the stack cranes is fixed. Suppose from stack crane h containers are coming from the order $j^h_1, j^h_2, \dots, j^h_{n_h}$. With k_j being the cycle of container j this results into the following constraints.

$$\begin{aligned} x^s_{j^h_2,i_1}(k_{j^h_2}) &\geq x^s_{j^h_1,i_2}(k_{j^h_1}) + p^s \\ &\vdots \\ x^s_{j^h_{n_h},i_1}(k_{j^h_{n_h}}) &\geq x^s_{j^h_{n_h-1},i_2}(k_{j^h_{n_h-1}}) + p^s \end{aligned} \tag{4.26}$$

$$\forall i_1, i_2 = 1, \dots, N_a, h = 1, \dots, N_s$$

Now the ordering for loading onto the ships via the quay cranes can be determined. Let $z^q_{j_1,j_2,i}(k) \in \{0,1\}$ be the ordering variable that determines the ordering of containers j_1 and j_2 in cycle k on quay crane q_i. Let $J_c(k)$ be the set of containers that have destination $c \in \{1, \dots, N_q\}$ in cycle k.

$$x^q_{j_2,i_2} \geq x^q_{j_1,i_1} + p^q + p^{cs} + \beta(1 - z^q_{j_1,j_2,i}(k)) + \beta(2 - v_{j_1,i_1}(k) - v_{j_2,i_2}(k)) \tag{4.27}$$
$$x^q_{j_1,i_1} \geq x^q_{j_2,i_2} + p^q + p^{cs} + \beta z^q_{j_1,j_2,i}(k) + \beta(2 - v_{j_1,i_1}(k) - v_{j_2,i_2}(k)) \tag{4.28}$$
$$j_1, j_2 \in J_{q_i}(k), j_1 < j_2, i_1, i_2 = 1, \dots, N_a, q_i = 1, \dots, N_q$$

Here, p^{cs} is the time to load a container on a ship by the quay crane. The system model is now complete. It remains to specify the objective function. Define variables $y_c(k)$ to be the total transfer time of ship c in cycle k. This can be modeled as inequality (4.29).

$$y_c(k) \geq x_{j,i}(k) + p^q + p^c s, \quad \forall j \in J_c(k),\ i = 1, \dots, N_a,\ c \in \{1, \dots, N_q\} \tag{4.29}$$

The total transfer time of cycles $k, \dots, N_p - 1$ and thus objective function is

$$J_{N_p}(k) = \sum_{\tau=0}^{N_p-1} \sum_{c=1}^{N_q} y_c(k+\tau) \tag{4.30}$$

Part II
Reducing complexity of the MPC-SMPL problem

Part I has shown what the SMPL model is and how computational complexity causes a problem to use MPC for SMPL systems in practice. Part II will explain some methods that might reduce the complexity of the MPC-SMPL problem. The first section will focus on reparameterizing the binary variables in the MILP to a reduced amount. The second section explaines the concept of model-based partitioning and the distributed algorithm that can be applied after the partitioning.

5 Reparameterization of Binary Variables

It is known from Appendix B that integer variables make the MILP problem difficult in its complexity. The number of nodes computed by the branch-and-bound algorithm (Appendix B.4) grows possibly exponentially in the problem size, which makes it hard to solve the MPC-SMPL problem in theory. It might be beneficial (if this is possible) to rewrite the problem such that there are less integer variables. Whether reducing the number of integer variables really reduces the running time of the MILP solver is to be answered as a result of this thesis. For now some boolean operators (Bousfield [10]) will be explained helping to understand the concept of binary variables used in integer programming. Table 5.1 shows an overview of the most common used logical operators and how they are defined when v_1 and v_2 are fixed. The variables represent statements and take values of $\{true, false\}$ depending on the legitimacy of the statement.

v_1	v_2	$\sim v_1$	$v_1 \vee v_2$	$v_1 \wedge v_2$	$v_1 \Rightarrow v_2$	$v_1 \Leftrightarrow v_2$	$v_1 \backslash v_2$	$v_1 \triangle v_2$
$true$	$true$	$false$	$true$	$true$	$true$	$true$	$false$	$false$
$true$	$false$	$false$	$true$	$false$	$false$	$false$	$true$	$true$
$false$	$true$	$true$	$true$	$false$	$true$	$false$	$false$	$true$
$false$	$false$	$true$	$false$	$false$	$true$	$true$	$false$	$false$

Table 5.1: The truth-table of boolean operators for max-plus binary variables.

These logical operators can be expressed in as well linear constraints of the form $a^T v \leq b$ as well as max-plus statements. Remark that for the conventional algebra binary variables are in $\{0, 1\}$. Table 5.2 shows the conversion of logical statements to max-plus algebra statements and linear constraints. For the linear constraints sometimes v^b is denoted in the table, this variable is 1 if the logical statement is true and zero otherwise. Here, the logical variables are denoted by v_1^l and v_2^l and take values from $\{true, false\}$.

logical statement	max-plus statement	linear constraint
$\sim v_1^l$	$\bar{v}_1$	$v^b = 1 - v_1^b$
$v_1^l \vee v_2^l$	$v_1 \oplus v_2$	$v_1^b \leq v^b, v_2^b \leq v^b, v^b \leq v_1^b + v_2^b$
$v_1^l \wedge v_2^l$	$v_1 \otimes v_2$	$v^b \leq v_1^b, v^b \leq v_2^b, v_1^b + v_2^b \leq v^b + 1$
$v_1^l \Rightarrow v_2^l$	$\bar{v}_1 \oplus v_2$	$v_1^b \leq v_2^b$
$v_1^l \Leftrightarrow v_2^l$	$(\bar{v}_1 \oplus v_2) \otimes (v_1 \oplus \bar{v}_2)$	$v_1^b = v_2^b$
$v_1^l \backslash v_2^l$	$v_1 \otimes \bar{v}_2$	$v_1^b - v_2^b \leq v^b, v^b \leq v_1^b, v^b + v_2^b \leq 1$
$v_1^l \triangle v_2^l$	$v_1 \otimes \bar{v}_2 \oplus \bar{v}_1 \otimes v_2$	$v_1^b + v_2^b = 1$

Table 5.2: Logical operators in max-plus and conventional algebra

Notice that $v_1 \Leftrightarrow v_2$ is equivalent to $(v_1 \Rightarrow v_2) \wedge (v_2 \Rightarrow v_1)$ and $v_1 \triangle v_2$ is equivalent to $(v_1 \backslash v_2) \vee (v_2 \backslash v_1)$ which explain the resulting max-plus statements. The equality constraints can be implemented by adding $a^T v \leq b$ as well as $-a^T v \leq -b$.

Example 5.1. Suppose there are 3 max-plus binary variables $v_1, v_2, v_3 \in \mathbb{B}_\epsilon$. It is given that $v_1^l \Rightarrow v_2^l$ and $v_3^l \Rightarrow v_1^l$. Note that there are only 4 possible true combinations of variables, given in equation (5.1).

$$(v_1, v_2, v_3) = \begin{cases} (\epsilon, \epsilon, \epsilon) \\ (\epsilon, 0, \epsilon) \\ (0, 0, \epsilon) \\ (0, 0, 0) \end{cases} \tag{5.1}$$

A reparameterization of these max-plus binary variables could be $v_1 = w_1$, $v_2 = w_1 \oplus w_2$, $v_3 = w_1 \otimes w_2$ for $w_1, w_2 \in \mathbb{B}_\epsilon$. The 4 possible truth combinations are shown in table 5.3 and do exactly coincide with the feasible binary solutions. The number of binary variables has been successfully reduced from 3 to 2.

	$w_2 = \epsilon$	$w_2 = 0$
$w_1 = \epsilon$	$(\epsilon, \epsilon, \epsilon)$	$(\epsilon, 0, \epsilon)$
$w_1 = 0$	$(0, 0, \epsilon)$	$(0, 0, 0)$

Table 5.3: Possible binary values for (v_1, v_2, v_2) after reparameterization.

Notice that when restricted to binary variables the minimum number of binary variables after reparameterization is always $\lceil \log_2(m) \rceil$ where m is the number of possible feasible solutions. This is concluded from the fact that for a set of n binary variables there are exactly 2^n binary combinations. SMPL system models of the form (2.16) or (2.18) at page 15 usually have a switching variable $v_i(k)$ for each switching matrix A_i where $v_i = 0$ if A_i is to be used for the model and $v_i = \epsilon$ otherwise. In this case the model has an equal number of variables $v_i(k)$ as feasible system models, and hence the number of binary variables can be reduced.
There are several ways to find a reparameterization, and they are not unique. Akers [3] shows how to find a reparameterization using binary decision trees.
Before going to the part of reparameterization for SMPL systems some notations are explained. First, when $N \in \mathbb{N}$ and $S \subseteq \underline{N} = \{1, \ldots, N\}$, define the vector $\mathbb{1}^S \in \mathbb{B}^N$ as the incidence vector of S with value 1 on the indices that are in $\underline{N}$ and value 0 on the other indices. Now define the function $\mathbb{2}_N : \mathbb{P}(\underline{N}) \to \{0, 1, \ldots, 2^N - 1\}$ (where $\mathbb{P}(\cdot)$ is the power function i.e. the set of all subsets) with

$$\mathbb{2}_N(S) = \sum_{i \in S} 2^{N-i}$$

In fact, $\mathbb{2}_N$ is a bijective function (Definition C.1 on page 124 or Koopman & Sportiche [36]). This can be seen if one takes $S \subseteq \underline{N}$, and transforms the vector $\mathbb{1}^S$ into a binary number ($[1\ 1\ 0\ 1]^T$ to 1101 for example), this number is exactly equal to $\mathbb{2}_N(S)$. Since all natural numbers have a unique binary representation, and with N binary values one can count up to $2^N - 1$, it follows that $\mathbb{2}_N$ must be bijective. Therefore, the inverse of $\mathbb{2}_N$ can be taken, which will be used later in this section.

5.1 Routing parameterization

A lot of SMPL systems can be modeled by constructing routing, ordering and synchronization system constraint matrices as described in section 2.2. The first two require a set of binary variables that are decision variables to determine which route a job will follow and in which order some processors execute the operations of the job. This section will elaborate on the routing variables from equations (2.21),(2.22) and (2.23) and explain reparameterizations such that there

are less binary variables in the system model. It is also important to do an analysis on what the least amount of possible binary variables is for a given instance or application.
Take the routing parameterization of equations (2.21),(2.22) and (2.23), and consider the routing of a single job j in a single cycle k. Let L_j be the set of all possible routes this job can follow. There now are $|L_j|$ variables, namely $v_{j,l}(k)$, that describe which route is chosen. However, this modeling framework requires that there is only one route $l \in L_j$ such that $v_{j,l}(k) = 0$ and it holds that $v_{j,\hat{l}}(k) = \epsilon$ for all other routes $\hat{l}$ of L_j. Since there are $|L_j|$ switching modes for the routing of job j, let $N^r_{1,j} = \lceil \log_2(|L_j|) \rceil$ be the smallest number of binary variable that is required to model $|L_j|$ switching modes. For reparameterization, introduce a set of new routing binary variables $w_j(k) \in \mathbb{B}_\epsilon^{N^r_{1,j}}$ and a function $f : \mathbb{B}_\epsilon^{N^r_{1,j}} \to \mathbb{B}_\epsilon^{|L_j|}$. This function can be pretty much anything, as long as $N_{1,j}$ is kept as small as possible and exactly one of the variables $v_{j,l}(k)$ is equal to 0. The following method provides such a function that can be used for reparameterization.

Routing parameterization 1. (R1)
For each job j and cycle k, make a reparameterization in the following way. First let $N^r_{1,j} = \lceil \log_2(|L_j|) \rceil$ for each $j \in J$. Now define a vector $w_j(k) \in \mathbb{B}^{N_{1,j}}$, and let

$$v_{j,l}(k) = \bigotimes_{a \in 2^{-1}_{N^r_{1,j}}(l-1)} w_{j,a}(k) \otimes \bigotimes_{a \in \underline{N^r_{1,j}} \setminus 2^{-1}_{N^r_{1,j}}(l-1)} \bar{w}_{j,a}(k) \tag{5.2}$$

The variables $v_{j,l}(k)$ can now be replaced expressions containing the variables $w_{j,i}(k)$ as in equations (5.2).
For example, suppose there are 6 routes for job j. Using the first reparameterization, equation (5.2) results in the following expressions.

$$\begin{aligned}
v_{j,1}(k) &= \bar{w}_{j,1}(k) \otimes \bar{w}_{j,2}(k) \otimes \bar{w}_{j,3}(k) \\
v_{j,2}(k) &= \bar{w}_{j,1}(k) \otimes \bar{w}_{j,2}(k) \otimes w_{j,3}(k) \\
v_{j,3}(k) &= \bar{w}_{j,1}(k) \otimes w_{j,2}(k) \otimes \bar{w}_{j,3}(k) \\
v_{j,4}(k) &= \bar{w}_{j,1}(k) \otimes w_{j,2}(k) \otimes w_{j,3}(k) \\
v_{j,5}(k) &= w_{j,1}(k) \otimes \bar{w}_{j,2}(k) \otimes \bar{w}_{j,3}(k) \\
v_{j,6}(k) &= w_{j,1}(k) \otimes \bar{w}_{j,2}(k) \otimes w_{j,3}(k)
\end{aligned} \tag{5.3}$$

Note that for $N^r_{1,j}$ binary variables, there are $2^{N^r_{1,j}}$ possible binary combinations. This means that $N^r_{1,j}$ is minimal when $N^r_{1,j} = \lceil \log_2(|L_j|) \rceil$. Applying such a reparameterization for each job results in $N^r_1 = \sum_{j \in J} N^r_{1,j} = \sum_{j \in J} \lceil \log_2(|L_j|) \rceil$ binary variables.
The good property of this parameterization is that it will always work. However, when $|L_j|$ is not an exact power of 2, there will be combinations of $w_j(k)$ that were not feasible in the first place. One must have the restrictions $w_{j,1}(k) \otimes w_{j,2}(k) \otimes \bar{w}_{j,3}(k) = \epsilon$ and $w_{j,1}(k) \otimes w_{j,2}(k) \otimes w_{j,3}(k) = \epsilon$ in order to keep the route feasible. In the sense of the conventional algebra, linear inequality (5.4) separates the non-feasible points from the formulation, for each $j \in J$.

$$\sum_{i=1}^{N^r_1} 2^{N^r_1 - i} w^b_{j,i}(k) \leq |L_j| - 1 \tag{5.4}$$

The second method of reparameterization might reduce the number of binary variables for routing even further. This method is also described by Van den Boom et al. [50]. It is similar to the first method, but there is a single reparameterization over all the routes of all the jobs.

Routing parameterization 2. (R2)
Here, a similar parameterization will be performed as described in the first method, but the modeling steps of equations (2.22) and (2.23) on page 17 are done in a different order. Instead of assigning a binary variable per route per job, now the variables will determine all the routes for all the jobs. Let L be the set of all route combinations of jobs of J. In other words, $L = L_1 \times L_2 \times \cdots \times L_n$. Denote $x_j(k) \geq A^r_{j,l_j} \otimes x_j(k)$ as a new simplification of equation (2.21). Now let $v_l(k) = 0$ if route combination l of L is selected in cycle k and let $v_l(k) = \epsilon$ otherwise. Redefine equations (2.22) and (2.23), take

$$A^r_l(k) = \begin{bmatrix} A^r_{1,l_1}(k) & \mathcal{E} & \cdots & \mathcal{E} \\ \mathcal{E} & A^r_{2,l_2}(k) & \cdots & \mathcal{E} \\ \vdots & \vdots & \ddots & \vdots \\ \mathcal{E} & \mathcal{E} & \cdots & A^r_{n,l_n}(k) \end{bmatrix} \tag{5.5}$$

where l is precisely the combination of routes $l_1, \ldots, l_n$. Now define

$$A^r(v(k), k) := \bigoplus_{l \in L} (v_l(k) \otimes A^r_l(k)) \tag{5.6}$$

as the new routing system matrix. The next step is to perform the same reparameterization trick as in the first method, on the new routing variables. When not having any restrictions on the combinations of routes, there are a minimal number of $N^r_2 = \lceil \log_2(|L|) \rceil = \lceil \log_2(\prod_{j \in J} |L_j|) \rceil$ new binary variables.

One can put restrictions on routes, by for example saying job j_1 cannot follow route l_{j_1} if job j_2 follows route l_{j_2}. In the first parameterization this can only be implemented by adding the constraint that of variables $v_{j_1,l_{j_1}}(k)$ and $v_{j_2,l_{j_2}}(k)$ at most one can be equal to 0. In the second parameterization this is easily done by removing all the combinations of routes from L that let job j_1 and j_2 follow routes l_{j_1} and l_{j_2} respectively. The question arises which parameterization has less binary variables and what the maximum difference is between the number of binary variables. The next proposition establishes a bound between the difference in number of binary variables N^r_1 and N^r_2.

Proposition 5.2. *Let the N^r_1 and N^r_2 be the number of binary variables after reparameterizations of methods 1 and 2 respectively. If there are no restrictions on the combinations of routes, then $0 \leq N^r_1 - N^r_2 \leq |J| - 1$ must hold.*[3]

Proof. It holds that $0 \leq N^r_1 - N^r_2$ is equivalent to $N^r_2 \leq N^r_1$. Note that $\lceil \lambda_1 + \lambda_2 \rceil \leq \lceil \lambda_1 \rceil + \lceil \lambda_2 \rceil$ for any real numbers λ_1, λ_2. Now it can be seen that

$$N^r_2 = \lceil \log_2(|L|) \rceil = \left\lceil \log_2(\prod_{j \in J} |L_j|) \right\rceil = \left\lceil \sum_{j \in J} \log_2(|L_j|) \right\rceil \leq \sum_{j \in J} \lceil \log_2(|L_j|) \rceil = N^r_1$$

For the second part of the proof it is first claimed that whenever $\lambda_1, \lambda_2 \in \mathbb{R}_+$ it holds that $\lceil \lambda_1 \rceil + \lfloor \lambda_2 \rfloor \leq \lceil \lambda_1 + \lambda_2 \rceil$. To prove this claim note that

$$\lceil \lambda_1 + \lambda_2 \rceil = \lceil \lambda_1 + (\lambda_2 - \lfloor \lambda_2 \rfloor) + \lfloor \lambda_2 \rfloor \rceil$$
$$= \lceil \lambda_1 + (\lambda_2 - \lfloor \lambda_2 \rfloor) \rceil + \lfloor \lambda_2 \rfloor = \begin{cases} \lceil \lambda_1 \rceil + \lfloor \lambda_2 \rfloor \\ \lceil \lambda_1 \rceil + 1 + \lfloor \lambda_2 \rfloor \end{cases}$$

[3]Proposition 5.2 is written and proved by the writer.

$$\Rightarrow \lceil \lambda_1 + \lambda_2 \rceil \geq \lceil \lambda_1 \rceil + \lfloor \lambda_2 \rfloor$$

which proves the claim. To prove that $N_1 - N_2 \leq |J| - 1$ use induction. Let $J_1 \subset J_2 \subset \cdots \subset J$ such that $|J_i| = i$ for all $i = 1, \dots, |J|$. It will be shown that

$$\sum_{j \in J_i} \lceil \log_2(|L_j|) \rceil - \lceil \sum_{j \in J_i} \log_2(|L_j|) \rceil \leq |J_i| - 1$$

for all $i = 1, \dots, n$. Now it is clear that $\sum_{j \in J_1} \lceil \log_2(|L_j|) \rceil - \lceil \sum_{j \in J_1} \log_2(|L_j|) \rceil = \lceil \log_2(|L_{j_1}|) \rceil - \lceil \log_2(|L_{j_1}|) \rceil = 0 = |J_1| - 1$ since $J_1 = \{j_1\}$. Now assume the same statement is true for $i = k$. For $i = k + 1$ it can now be determined that

$$\begin{aligned}
& \sum_{j \in J_{k+1}} \lceil \log_2(|L_j|) \rceil - \lceil \sum_{j \in J_{k+1}} \log_2(|L_j|) \rceil \\
&= \lceil \log_2(|L_{k+1}|) \rceil + \sum_{j \in J_k} \lceil \log_2(|L_j|) \rceil - \lceil \sum_{j \in J_{k+1}} \log_2(|L_j|) \rceil \\
\text{(by claim)} \quad &\leq \lceil \log_2(|L_{k+1}|) \rceil + \sum_{j \in J_k} \lceil \log_2(|L_j|) \rceil - \lceil \sum_{j \in J_k} \log_2(|L_j|) \rceil - \lfloor \log_2(|L_{k+1}|) \rfloor \\
\text{(induction assumption)} \quad &\leq \lceil \log_2(|L_{k+1}|) \rceil - \lfloor \log_2(|L_{k+1}|) \rfloor + |J_k| - 1 \\
&= |J_k| = |J_{k+1}| - 1
\end{aligned}$$

It can now be concluded from induction that

$$N_1^r - N_2^r = \sum_{j \in J} \lceil \log_2(|L_j|) \rceil - \lceil \sum_{j \in J} \log_2(|L_j|) \rceil \leq |J| - 1$$

which completes the proof. □

Proposition 5.2 shows that when any combination of of routes can be chosen the differences in number of binary variables per cycle of method 1 and 2 is at most the number of jobs minus 1. Method 2 always results in less binary variables, which is desired, but it can be harder to model all the combinations of routes than just all routes for all jobs themselves. Since $|J| - 1$ is a small bound on the difference between N_1^r and N_2^r it is therefore worth to take both reparameterization methods in consideration for this thesis.
One may ask whether it is a good idea to model a binary variable for each route in the first place. If there are many processors, say m, there can be potentially many routes of the jobs. This naturally depends on the application, but if this is the case there can be up tot $m!$ routes per job. The modeling methods explained so far rely on including all the possible routes into the model, as the third method will rely on including all $m!$ routes in the first place without having too much variables, and excluding the non-possibilities afterwards. In the case that there are many feasible routes this might be less modeling work.

Routing parameterization 3. (R3)
This method will only be applicable on the case of fixed processors for routing, so for every route $l \in L_j$ it must hold that $M(l) = R_j = \{i_1, i_2, \dots, i_{|R_j|}\}$. Consider the variable

$$w_{j,i_q,i_r}(k) = \begin{cases} 0 & \text{if job } j \text{ in cycle } k \text{ is processed on } i_r \text{ directly after it was processed on } i_q \\ \epsilon & \text{else} \end{cases}$$

The matrices $A_j^r(v(k), k)$ from equation (2.22) on page 17 can now be replaced by (5.7).

$$A_j^r(w(k),k) := \tag{5.7}$$

$$\begin{bmatrix} \epsilon & w_{j,i_2,i_1}(k) & w_{j,i_3,i_1}(k) & \cdots & w_{j,i_{|R_j|},i_1}(k) \\ w_{j,i_1,i_2}(k) & \epsilon & w_{j,i_3,i_2}(k) & \ddots & w_{j,i_{|R_j|},i_2}(k) \\ w_{j,i_1,i_3}(k) & w_{j,i_2,i_3}(k) & \epsilon & \ddots & w_{j,i_{|R_j|},i_3}(k) \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ w_{j,i_1,i_{|R_j|}}(k) & w_{j,i_2,i_{|R_j|}}(k) & w_{j,i_3,i_{|R_j|}}(k) & \cdots & \epsilon \end{bmatrix} \otimes \begin{bmatrix} p_{j,i_1}(k) & \epsilon & \cdots & \epsilon \\ \epsilon & p_{j,i_2}(k) & \ddots & \epsilon \\ \vdots & \ddots & \ddots & \vdots \\ \epsilon & \epsilon & \cdots & p_{j,i_{|R_j|}} \end{bmatrix}$$

where $p_{j,i}(k)$ is the processing time of operation (j,i) in cycle k. The matrix $A^r(w(k),k)$ follows from the same construction as in equation (2.23), but has switching variable $w(k)$ instead of $v(k)$. For each cycle, there are now $N_3^r = \sum_{j\in J} |R_j|(|R_j|-1)$ binary variables.
To make sure the variables take values that represent a feasible route, a number of constraints are added for each cycle k. The first constraint excludes solutions where an operation has two simultaneous successive processors. The constraint is

$$w_{j,i_q,i_r}(k) \otimes w_{j,i_q,i_s}(k) = \epsilon \ \forall j \in J,\ i_q,i_r,i_s \in R_j,\ i_r \neq i_s$$

which is equivalent to the next constraint in conventional algebra.

$$\sum_{i_r\in R_j} w^b_{j,i_q,i_r}(k) \leq 1 \ \forall j \in J,\ i_q \in R_j \tag{5.8}$$

Likewise, every operation has at most one preceding processor. This can be modeled by adding the constraint

$$w_{j,i_q,i_s}(k) \otimes w_{j,i_r,i_s}(k) = \epsilon \ \forall j \in J,\ i_q,i_r,i_s \in R_j,\ i_q \neq i_e$$

which is equivalent to

$$\sum_{i_q\in R_j} w^b_{j,i_q,i_r}(k) \leq 1 \ \forall j \in J,\ i_r \in R_j \tag{5.9}$$

in conventional algebra. The next step is to make sure every job has a beginning processor and an end processor. There may therefore be no circuits in $\mathcal{G}(A_j^r(w(k),k))$, and the number of entries of $A_j^r(w(k),k)$ that is different from ϵ must be exactly $|R_j|-1$. Moreover, $\mathcal{G}(A_j^r(w(k),k))$ should form a simple path. This is equivalent to setting

$$\sum_{i_q\in R_j}\sum_{i_r\in R_j} w^b_{j,i_q,i_r}(k) = |R_j|-1 \ \forall j \in J \tag{5.10}$$

$$\sum_{\substack{i_q,i_r\in S,\\ i_q\neq i_r}} w^b_{j,i_q,i_r}(k) \leq |S|-1 \quad \forall j \in J,\ S \subseteq R_j,\ |S| \geq 3 \tag{5.11}$$

Unfortunately, by constraint (5.11) the number of constraints is exponentially large in the size of R_j. How to deal with exponentially many constraints will be explained in section 8 or an alternatively one can apply advanced pruning, explained in Appendix B.5. Finally, the routes that are not included in L_j can be excluded. let $l = (i_{q_1}, i_{q_2}, \ldots, i_{q_{|R_j|}})$ be a route that is not included in L_j. In max-plus, this route can be excluded by setting

$$w_{j,i_{q_1},i_{q_2}}(k) \otimes w_{j,i_{q_2},i_{q_3}}(k) \otimes \cdots \otimes w_{j,i_{q_{|R_j|-1}},i_{q_{|R_j|}}}(k) = \epsilon$$

In linear constraints, this is equivalent to

$$\sum_{r=1}^{|R_j|-1} w^b_{j,i_{q_r},i_{q_{r+1}}}(k) \leq |R_j| - 1 \tag{5.12}$$

The formulation is now complete. By Proposition 5.3 it can be shown that routing parameterization 3 results in more variables than routing parameterization 1 (and thus more than parameterization 2 by Proposition 5.2). The third method however surpasses the initial step where a variable is modeled for every route. The third method of reparameterization for routing variables is therefore only recommended when the number of routes is very large. Note that the constraints (5.12) can sometimes also grow exponentially large. It is therefore advised to reformulate these constraints, depending on the structure of L_j.

Proposition 5.3. *Assuming a route set L_j is nonempty and there is at least one processor, then $N^r_1 \leq N^r_3$ must hold.*[4]

Proof. The proposition will be proved using induction. Notice that there are at most $|R_j|!$ possible routes for j, so $N^r_1 \leq \lceil \log_2(|R_j|) \rceil$. This means that proving $\lceil \log_2(p!) \rceil \leq p(p-1)$ for $p \geq 1$ would be sufficient to prove the Proposition. Now for $p = 1$ it holds that $\lceil \log_2(p!) \rceil = 0 = p(p-1)$ and that completes the first step of induction. It also holds for $p = 2$, since $\lceil \log_2(p!) \rceil = 1 \leq p(p-1)$. For the second step of induction, assume $\lceil \log_2((p-1)!) \rceil \leq (p-1)(p-2)$. Notice that for $p \geq 1$ it holds that $\log_2(p) \leq p$. It then follows for $p \geq 3$ that

$$\begin{aligned}
\lceil \log_2(p!) \rceil &\leq \log_2(p!) + 1 \\
&= \log_2((p-1)! \cdot p) + 1 \\
&= \log_2((p-1)!) + \log_2(p) + 1 \\
&\leq \lceil \log_2((p-1)!) \rceil + \log_2(p) + 1 \\
&\leq (p-1)(p-2) + \log_2(p) + 1 \\
&= p^2 - 3p + 3 + \log_2(p) \\
&\leq p^2 - 2p + 3 \\
&\leq p^2 - p \\
&= p(p-1)
\end{aligned}$$

and the proof is now complete. □

This means that the third method of reparameterization for routing variables has more binary variables than the other methods. However, it might be less modeling work to implement, or result in less constraint in the case of a large set L_j.

5.2 Ordering parameterization

With multiple jobs traversing multiple routes, it occasionally occurs that two operations simultaneously wind up at a single processor. To prevent the model from allowing a processor do two operations at the same time, an order of events should be constructed by the model. In section 2.2 the matrix $A^o_\mu(v(k), k)$ as defined in (2.26) on page 18 has been constructed such that one

[4]Proposition 5.3 is written and proved by the writer.

can use binary variables $v(k)$ to determine the ordering of operations. When decomposing the this matrix as in equation (2.26) on page 18, the only matrix that uses the ordering variables of $v(k)$ is the matrix $Q_\mu(v(k), k)$. Now denote the operations on processor i by $e_1, e_2, \ldots, e_p$. Notice that these operations can be from different cycles, therefor the cycle notation k is left out for the purpose of simplicity in this section. Now let Z_i be the matrix that has ϵ on the diagonal, $[Z_i]_{e_1,e_2} = 0$ if e_2 is processed after e_1 on processor i and $[Z_i]_{e_1,e_2} = \epsilon$ if otherwise. Now it becomes immediately clear from the interpretation that if $[Z_i]_{e_1,e_2} = 0$ that it follows that $[Z_i]_{e_2,e_1} = \epsilon$. The matrix Z_i can therefore be written as in equation (5.13).

$$Z_{\mu,i} = \begin{bmatrix} \epsilon & \bar{z}_{21} & \bar{z}_{31} & \cdots & \bar{z}_{p1} \\ z_{21} & \epsilon & \bar{z}_{32} & \ddots & \bar{z}_{p2} \\ z_{31} & z_{32} & \epsilon & \ddots & \bar{z}_{p3} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ z_{p1} & z_{p2} & z_{p3} & \cdots & \epsilon \end{bmatrix} \tag{5.13}$$

Notice that the matrix Z_i now contains only $\frac{1}{2}p(p-1)$ variables while there can be up to $p!$ possible orders. However, this parameterization is not of the minimal number of variables (unless $p \leq 3$). This can be seen from the fact that not every value of Z_i results in a feasible ordering. For example, the matrix

$$\begin{bmatrix} \epsilon & \epsilon & 0 \\ 0 & \epsilon & \epsilon \\ \epsilon & 0 & \epsilon \end{bmatrix}$$

results in the ordering operation e_2 after e_1, e_3 after e_2, and e_1 after e_3. This is a circuit, and makes the ordering infeasible. Notice that if Z_i represents order $(e_1, e_2, \ldots, e_p)$, then its communication graph $\mathcal{G}(Z_i)$ has an arc from e_q to j_r if and only if e_r is ordered after e_q. So if Z_i represents a feasible ordering, $\mathcal{G}(Z_i)$ cannot have a circuit. An example of a feasible ordering is $(e_3, e_2, e_5, e_6, e_1, e_4)$, and its matrix Z_i and communication graph $\mathcal{G}(Z_i)$ are shown in figure 5.1.

$$\begin{bmatrix} \epsilon & 0 & 0 & \epsilon & 0 & 0 \\ \epsilon & \epsilon & 0 & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ 0 & 0 & 0 & \epsilon & 0 & 0 \\ \epsilon & 0 & 0 & \epsilon & \epsilon & \epsilon \\ \epsilon & 0 & 0 & \epsilon & 0 & \epsilon \end{bmatrix}$$

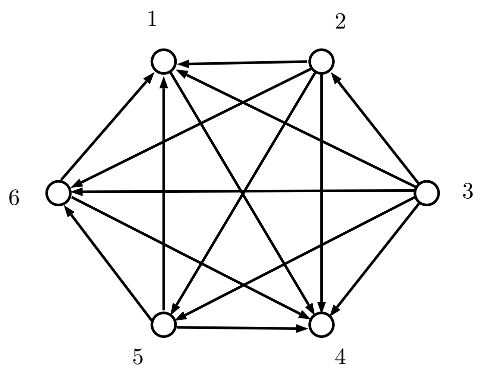


Figure 5.1: The matrix Z_i and graph $\mathcal{G}(Z_i)$ for order $(3, 2, 5, 6, 1, 4)$.

Notice that in $\mathcal{G}(Z_i)$ there is exactly one arc between every set of vertices. This can be seen by the definition of Z_i. When there is no arc from e_1 to e_2, then $z_{e_2 e_1} = \epsilon$ which means $z_{e_1 e_2} = 0$ so there must be an arc from e_2 to e_1. Conversely, the same argument implies that if there is an arc from e_1 to e_2, there is no arc from e_2 to e_1. To establish feasibility of the matrix Z_i, Theorem 5.4 brings us a helpful property of $\mathcal{G}(Z_i)$.

Theorem 5.4. *Let $\mathcal{G} = (N, D)$ be a directed graph with exactly one arc between every pair of vertices, and no arcs from a vertex to itself. If $\mathcal{G}$ does not contain any circuit of length 3, then $\mathcal{G}$ is acyclic.*[5]

Proof. Suppose $\mathcal{G} = (N, D)$ has exactly one arc between every pair of vertices and has no circuit of length 3. Assume D does not contain arcs from a vertex to itself. Now suppose $\mathcal{G}$ is cyclic. Since there are no circuits of length 1 and 2 by the assumptions of $\mathcal{G}$, there must be a circuit $C = (n_1, n_2, n_3, \dots, n_k, n_1)$ with $|C| \geq 4$. This means that (n_1, n_2) and (n_2, n_3) are in D. Since there are no circuits of length 3 it cannot be that (n_3, n_1) is in D. Now because there is exactly one arc between every pair of vertices, it must hold that $(n_1, n_3) \in D$. This means that $\tilde{C} = (n_1, n_3, \dots, n_k, n_1)$ is a circuit of $\mathcal{G}$. So there exists a circuit of length $|\tilde{C}| = |C| - 1$. So if $\mathcal{G}$ has a circuit with length 4 or larger, there must be a circuit with length one less. Using this argument repetively, there must be a circuit of length 3. This is a contradiction to the assumption which leads to the conclusion that $\mathcal{G}$ is acyclic. $\square$

With the help of Theorem 5.4, the first method of reparameterization of the ordering variables is constructed.

Ordering parameterization 1. (O1)
The first method of ordering parameterization uses precisely the $\frac{1}{2}p(p-1)$ parameters from the matrix Z_i in equation 5.13. The only thing that has to be added are some constraints, such that the values take a feasible ordering. Since, $\mathcal{G}(Z_i)$ has always $\frac{1}{2}p(p-1)$ arcs, the only worry is to exclude circuits of $\mathcal{G}(Z_i)$ from the communication graph in order to obtain a feasible ordering. Due to Theorem 5.4, it suffices to exclude only the circuits of length 3. So the following constraints are added to the problem.

$$z_{e_2e_1} \otimes z_{e_3e_2} \otimes \bar{z}_{e_3e_1} = \epsilon$$
$$\bar{z}_{e_2e_1} \otimes \bar{z}_{e_3e_2} \otimes z_{e_3e_1} = \epsilon$$
$$\forall S = \{e_1, e_2, e_3\} \subseteq \underline{p},\ e_1 < e_2 < e_3$$

In linear constraints, this is equivalent to

$$z^b_{e_2e_1} + z^b_{e_3e_2} + (1 - z^b_{e_3e_1}) \leq 2 \tag{5.14}$$
$$(1 - z^b_{e_2e_1}) + (1 - z^b_{e_3e_2}) + z^b_{e_3e_1} \leq 2 \tag{5.15}$$
$$\forall S = \{e_1, e_2, e_3\} \subseteq \underline{p},\ e_1 < e_2 < e_3$$

Notice that these are $2\binom{p}{3} = 2\frac{p!}{3!(p-3)!} = \frac{1}{3}p(p-1)(p-2) = \mathcal{O}(p^3)$ constraints that have to be added. Although they are many constraints, they are not exponential with respect to p. The total number of binary variables of this method is $N^o_1 = \frac{1}{2}p(p-1)$

The first method of the parameterization of ordering constraints does not have a minimum number of binary variables (assuming $p > 3$). When leaving out any restrictions on ordering, there are $p!$ feasible orders. This means the number of binary variables can be reduced to $\lceil \log_2(p!) \rceil < \frac{1}{2}p(p-1)$ for $p \geq 4$.

Ordering parameterization 2. (O2)
Assume every ordering of $\underline{p}$ is a feasible ordering. This method will reparameterize the values of

[5]Theorem 5.4 is written and proved by the writer.

equation (5.13) to a minimal number of binary variables. Let $N_2^o = \lceil \log_2(p!) \rceil$ be the number of resulting variables where $\gamma \in \mathbb{B}_\epsilon^{N_2^o}$. Now consider the function $f_{N_2^o} : \mathbb{B}_\epsilon \times \mathbb{N}^2 \rightarrow \mathbb{B}_\epsilon$ with

$$f_{N_2^o}(v, b, t) = \begin{cases} v & \text{if } t \in 2_{N_2^o}^{-1}(b) \\ \bar{v} & \text{else} \end{cases} \tag{5.16}$$

Now arrange all possible orders or $\underline{p}$ from 0 to $p! - 1$. Order them in parts, where the first part contains all the orders where 1 is processed first, in the second part 2 is processed first etc. Then per part, split again the orders into parts where the parts are arranged from processing the lowest remaining value of $\underline{p}$, and repeat. For $p = 3$, the resulting order is shown in (5.17). Here, every row is an ordering. The first row denotes the first operation, the second row the second operation etc. The last row is the index of the order.

$$\begin{array}{ccccc} 1 & 2 & 3 & & 0 \\ 1 & 3 & 2 & & 1 \\ 2 & 1 & 3 & & 2 \\ 2 & 3 & 1 & & 3 \\ 3 & 1 & 2 & & 4 \\ 3 & 2 & 1 & & 5 \end{array} \tag{5.17}$$

Now let $I_p^o(e_1, e_2)$ be the set of ordering indices where e_1 is processed after e_2. Now reparameterize the variables of the matrix Z_i as in equation (5.18).

$$z_{e_2 e_1} = \bigoplus_{b \in I_p(e_1, e_2)} \bigotimes_{t=1}^{N_2^o} f_{N_2^o}(\gamma_t, b, t) \tag{5.18}$$

Continuing with the example of $p = 3$, it results that for example the z_{21} is reparameterized to

$$z_{21} = \bar{\gamma}_1 \otimes \bar{\gamma}_2 \otimes \bar{\gamma}_3 \oplus \bar{\gamma}_1 \otimes \bar{\gamma}_2 \otimes \gamma_3 \oplus \gamma_1 \otimes \bar{\gamma}_2 \otimes \bar{\gamma}_3 \tag{5.19}$$

Finally, it must hold that there is always a feasible order selected. For this, the linear constraint (5.20) is added.

$$\sum_{t=1}^{N_2^o} 2^{N_2^o - t} \gamma_t^b \leq p! - 1 \tag{5.20}$$

The variable $z_{e_2 e_1}$ will now be 0 if and only if e_2 is ordered after e_1, and the matrix $Z_i(\gamma)$ will always be a feasible ordering under constraint (5.20). To implement this method, the MATLAB file `OrderSearchTree.m` in Appendix F computes $I_p^o(e_1, e_2)$ from e_1 and e_2. However, this file has a running time of $\mathcal{O}((p-2)!)$, which is terrible for large p. It is therefore recommended to perform this method only for small p.

5.3 The Quine-McClusky method

In logical design, there is a typical question whether a logical description is of minimal variables and operations. This is often seen in hardware design, where an engineer is supposed to minimize the number of electrical circuit operators, without changing the input to output structure. Every operator is implemented by a physical component in the electrical circuit. To minimize the costs, it is required to minimizes the number of electrical components. The same problem arises from the design of binary parameters as every logical operation results in a transformation to linear constraints. Minimizing the number of boolean operations is therefore equivalent to minimizing the number of constraints resulting in the MILP arising from the MPC-SMPL problem. It is however important to keep in mind that it is not guaranteed that less constraints reduce the complexity of the problem. The complexity comes from an exponential number of constraints, redundancy of constraints, and the quality of the constraints as in the strength of the relaxation. Since strong inequalities can always be added later in the form of cutting planes, the number of constraints will initially be minimized.

One of the methods that is often used for the simplification of a logical function, is the Quine-McClusky algorithm [40][45]. The algorithm takes an existing logical expression, and turns it into a minimal *sum of products* expression. In boolean algebra (Bousefield [10]), the $\vee$ (or) is the addition operator and the $\wedge$ (and) is the multiplication operator. For example, the expression

$$y = (x_1 \wedge x_2) \vee (\bar{x}_1 \wedge x_3) \tag{5.21}$$

is a sum of products, and $y = true$ if and only if $x_1 = x_2 = true$ or $x_1 = false$ and $x_3 = true$. The Quine-McClusky algorithm now transforms such a boolean expression into a new function of minimal variables and boolean operators. The input to the Quine-McClusky method is a specification on each binary combination of the original binary variables. Every binary combination has to be assigned a *true*, *false* or *don't care* value. The output of the logical expression must be equal to this value when evaluating the binary combination as input, unless the combination is a *don't care*. In this case the output does not matter. The output of the Quine-McClusky algorithm is a minimal expression for the output of the original expression. The resulting expression minimizes the number of variables in the first place, and then minimizes the number of boolean operators. More details on how the algorithm exactly works can be found in Quine [45] & McClusky [40]. The implementation of the method used for this thesis is the `minTruthtable.m` from Petter [43].

Notice that a sum of products in max-plus binary variables, is equivalent to a sum of products in boolean algebra (Bousefield [10]). The Quine-McClusky algorithm can be used to improve routing parameterizations 1 and 2, since they are a single product. The algorithm should be performed per variable. The only binary combination assigned a *true* is the one where only the correct routing variable is 0 and the others are ϵ in the max-plus sense. The infeasible combinations become *don't care* terms. This method now improves for example the parameterization (5.3) for 6 routes to parameterization (5.22).

$$\begin{aligned}
v_{j,1}(k) &= \bar{w}_{j,1}(k) \otimes \bar{w}_{j,2}(k) \otimes \bar{w}_{j,3}(k) \\
v_{j,2}(k) &= \bar{w}_{j,1}(k) \otimes \bar{w}_{j,2}(k) \otimes w_{j,3}(k) \\
v_{j,3}(k) &= w_{j,2}(k) \otimes \bar{w}_{j,3}(k) \\
v_{j,4}(k) &= w_{j,2}(k) \otimes w_{j,3}(k) \\
v_{j,5}(k) &= w_{j,1}(k) \otimes \bar{w}_{j,3}(k) \\
v_{j,6}(k) &= w_{j,1}(k) \otimes w_{j,3}(k)
\end{aligned} \tag{5.22}$$

The number of variables has not changed, but the number of operations has. This leads to less variables in the constraints for routing variables in the resulting MILP. Notice that $w_j(k) = (0, 0, \epsilon)$ enables $v_{j,3}(k) = v_{j,5}(k) = 0$, which cannot be possible. Similarly, $w_j(k) = (0, 0, 0)$ enables $v_{j,4}(k) = v_{j,6}(k) = 0$. To prevent the feasibility of multiple routes, constraint (5.4) should again be included.
Setting up the Quine-McClusky algorithm for the third routing parameterization is quite different. In this case the variables $w_{j,i_q,i_r}(k)$ take value 0 in multiple routes. Let all possible routes (including infeasible routes) be ordered systematically as in (5.17), for processors of R_j. Consider

the index function $I^r_j : R^2_j \to \mathbb{P}(\mathbb{N})$ where $I^r_j(i_q, i_r)$ denotes the indices of all the routes of job j where processor i_q is used directly after processor i_r. An implementation of this index function is `RoutingSearchTree.m` and can be found in Appendix F. Then for each variable $w_{j,i_q,i_r}(k)$ the Quine-McClusky algorithm has to be performed with a *don't care* term for every infeasible route, and a *true* value for every index of $I^r_j(i_q, i_r)$ that represents a feasible route. Note that if $i \in I^r_j(i_q, i_r)$ but it is the index of an infeasible route, the corresponding term should be considered to be a *don't care* term.

Example 5.5. Consider a job j that has fixed processor set $R_j = \{1, 2, 3, 4\}$. The infeasible routes are

$$(2, 3, 1, 4),\ (2, 4, 3, 1),\ (3, 2, 1, 4),\ (4, 3, 1, 2),\ (4, 3, 2, 1)$$

In this example it will be shown how the reparameterization on variable $w_{j,2,1}(k)$ will be performed, using the Quine-McClusky algorithm. The variable $w_{j,2,1}(k)$ is constructed as in the third routing parameterization method.
The corresponding indices to these infeasible routes are $8, 11, 14, 22$ and 23. Let D be the set containing these numbers. Now by computation it can be determined that $I^r_j(1, 2) = \{6, 7, 14, 17, 20, 23\}$, so the binary combinations that should get a *true* value are indexed by $6, 7, 17$ and 20. The *don't care* terms are the elements of D. Since there are $|R_j|! - 5 = 19$ feasible routes, there should be $\lceil \log_2(19) \rceil = 5$ resulting variables $\gamma_j(k)$. Now running the Quine-McClusky algorithm by the `minTruthtable.m` file from Petter [43] The output translates to the following parameterization:

$$\begin{aligned} w_{j,2,1}(k) &= \gamma_{j,1}(k) \otimes \bar{\gamma}_{j,3}(k) \otimes \bar{\gamma}_{j,4}(k) \otimes \gamma_{j,5}(k) \\ &\oplus \bar{\gamma}_{j,2}(k) \otimes \gamma_{j,3}(k) \otimes \gamma_{j,4}(k) \\ &\oplus \gamma_{j,1}(k) \otimes \gamma_{j,3}(k) \otimes \bar{\gamma}_{j,5}(k) \end{aligned} \tag{5.23}$$

When using the Quine-McClusky method on routing parameterization 3, the infeasible routes are translated to *don't care* terms. This means that they cannot be selected as an output of the resulting parameterization, and thus can constraints (5.12) be left out. Moreover, every outcome of the variables $w_{j,i_q,i_r}(k)$ forms a feasible routing. It can not be possible that an operation has two preceding or successive processors, and thus can constraints (5.8) and (5.9) be left out. Finally, every outcome will also provide an acyclic path on R_j, which means there are exactly $|R_j| - 1$ variables equal to 0, and there are no subsets of processors that have a route in a circuit. Conclusively, the constraints (5.10) and (5.11) can be left out. As a result, one benefit of the Quine-McClusky method on the third method of reparameterization is that many constraints drop out. Most importantly, the exponentially many constraints of (5.11) can be left out.

For the ordering variables one cannot directly apply the Quine-McClusky method to the first parameterization of ordering variables, since the variables z_{e_1,e_2} of (5.13) are not a sum of products. To find a sum of products for expression for z_{e_1,e_2} one can do the steps of the second method of parameterization of ordering constraints. This method has resulting variables in the form of equation (5.18) which is a nice sum of squares. It is therefore recommended to use the Quine-McClusky algorithm if one decides to use the second method of ordering parameterization. Notice that $I^o_p(e_1, e_2)$ is precisely the set of indices for combinations of z_{e_1,e_2} that are assigned the value *true*. The downside is that it takes $\mathcal{O}((p-2)!)$ steps to get the indices of *true* values. Computing $I^o_p(e_1, e_2)$ is however part of the parameterization method itself, so there is no reason not to use Quine-McClusky. Doing so, equation (5.19) reduces to equation (5.24).

$$z_{21} = \bar{\gamma}_1 \otimes \bar{\gamma}_2(k) \oplus \bar{\gamma}_2 \otimes \bar{\gamma}_3 \tag{5.24}$$

5.4 The implementation of reparameterizations

The discussed parameterizations are all different in some way, and some methods might be very impractical in some cases. However, they can all be used to substitute in the SMPL system matrices. All substitutions are a sum of products. The number of factors in a product determine the number of binary variables that are present in a constraint, and the number of terms in a sum determines the number of constraints. So if the sum of products becomes a sum of many terms there will also be an unfortunate amount of constraints.
The sum of products structure of a parameterization for a binary variable in the max-plus sense translates well into the MILP constraints. Suppose a max-plus binary variable $v \in \mathbb{B}_\epsilon$ is reparameterized into a sum of squares of N variables, the sum of products is of the following form.

$$v = \bigoplus_{q=1}^{c} \left(\bigotimes_{i \in S_q} w_i \otimes \bigotimes_{i \in \bar{S}_q} \bar{w}_i \right) \tag{5.25}$$

Here, S_k and $\bar{S}_k$ are disjoint subsets of $\underline{N}$. Now v is equal to 0 if at least one of the product of is equal to 0. This means that there is at least one $q \in \underline{c}$ such that $w_i = 0$ for all $i \in S_q$ and $w_i = \epsilon$ for all $i \in \bar{S}_q$. If this is not the case for all $a \in \underline{c}$ then $v = \epsilon$. In order to transform the linear constraints in v to a linear constraints for w all the constraints containing v need to be replaced. In fact, every constraint needs to be replaced by exactly c constraints. In the MPC-SMPL problem an original linear constraint in v is of the form

$$a^T x + \beta(1 - v^b) \leq b \tag{5.26}$$

or

$$a^T x + \beta v^b \leq b \tag{5.27}$$

To transform the constraint (5.26) into a constraint dependant on variables w, it can be replaced by the constraints of inequality (5.28).

$$a^T x + \beta \sum_{i \in S_q} (1 - w_i^b) + \beta \sum_{i \in \bar{S}_q} w_i^b \leq b \quad \forall q \in \underline{c}, w^b \in \{0, 1\}^N \tag{5.28}$$

To replace constraint (5.27), a parameterization in sum of product form of $\bar{v}$ has to be obtained. In the next steps it will be explained how such a sum of product form can always be obtained from equation (5.25). First, it is important to take note of a few properties in max-plus algebra. As described in Heidergott et al. [30], the max-plus semiring $\mathbb{R}_{\max}$ (Definition 1.1 page 7) submits to the property of distributivety of $\otimes$ over $\oplus$. This means that it holds that $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$. Using this property it can be seen that

$$\begin{aligned}(x \oplus y) \otimes (u \oplus w) &= ((x \oplus y) \otimes u) \oplus ((x \oplus y) \otimes w) \\ &= x \otimes u \oplus x \otimes w \oplus y \otimes u \oplus y \otimes w\end{aligned}$$

or in a more general form

$$\bigotimes_{i=1}^{p} \left(\bigoplus_{j=1}^{q_i} x_{ij} \right) = \bigoplus_{j_1=1}^{q_1} \cdots \bigoplus_{j_p=1}^{q_p} \left(\bigotimes_{i=1}^{p} x_{ij_i} \right) \tag{5.29}$$

for $x_{ij} \in R_\epsilon$. Next, notice from tables 5.1 and 5.2 it can be obtained that for variables $x, y \in \mathbb{B}_\epsilon$ equations (5.30) must hold.

$$\overline{x \oplus y} = \bar{x} \otimes \bar{y} \quad \text{and} \quad \overline{x \otimes y} = \bar{x} \oplus \bar{y} \tag{5.30}$$

Or described in general form it holds that

$$\overline{\bigoplus_{i \in \underline{p}} x_i} = \bigotimes_{i \in \underline{p}} \bar{x}_i \quad \text{and} \quad \overline{\bigotimes_{i \in \underline{p}} x_i} = \bigoplus_{i \in \underline{p}} \bar{x}_i \tag{5.31}$$

with $x_i \in \mathbb{B}_\epsilon$. With the help of expressions (5.25), (5.29) and (5.31) it can now be obtained that $\bar{v}$ from equation can be written as equation (5.32)[6].

$$\bar{v} = \overline{\bigotimes_{q=1}^{c} \left(\bigoplus_{i \in S_q} \bar{w}_i \oplus \bigoplus_{i \in \bar{S}_q} w_i \right)} = \bigoplus_{\substack{i_1 \in S_1 \\ j_1 \in \bar{S}_1}} \cdots \bigoplus_{\substack{i_K \in S_c \\ j_c \in \bar{S}_c}} \left(\bigotimes_{k=1}^{c} \bar{w}_{i_q} \otimes w_{j_q} \right) \tag{5.32}$$

This is a nice sum of products representation for $\bar{v}$, which means that it can be implemented as in equation (5.28). However, the expression in equation (5.32) often is a sum of many products, which means that implementing it directly leads to many constraints. It is rarely a minimal expression of boolean variables. Alternetively, one can use the Quine-McClusky method (section 5.3) to find a minimal sum of products expression for $\bar{v}$. In this case, the binary combinations assigned a $true$ and $false$ are exactly the opposite (unless it is a don't case term) for reparameterizing v.

One might ask why using the Quine-McClusky method would not always be the best choice for a reparameterization method. As stated before, The input of the QM method can be quit large for an ordering set can be quite expensive in computational matters. The MATLAB implementation of the Quine-McClusky method is `minTruthtable.m` and can only handle up to 2^{11} binary combinations. When reparameterizing p operations such that $\frac{1}{2}p! < 2^{11}$ it is obtained that $p = 6$ is the maximum number of operations of which all ordering variables can be reparameterized with the Quine-Mcclusky method.
There is a heuristic alternative called the ESPRESSO algorithm (Brayton [11]). It does not solve the problem to a minimum amount of logical operations, but it is much faster. The `minTruthtable.m` file from Petter [43] has an option to run the ESPRESSO algorithm, but it can only do up to 2^{13} binary combinations. In reducing ordering variables, this means that it can handle up to $p = 7$ operations at most, since $7! < 2^{13} < 8!$.

Because the Quine-McClusky method cannot fully reparameterize all ordering variables for processors containing more than 7 operations, the operations will be assigned to groups that will have their own reparemeterization. Let N_p be the prediction horizon, and $\mu_{\max}$ be the maximum difference in cycles of two operations that can be assigned an ordering. Choose a processor i and let p be the number of operations per cycle that is processed on the processor i. Based on the fact that one can only define at most 2^{11} binary combinations, consider the following cases for each processor:

- If $p(\mu_{\max} + 1) \geq 6$, consider the following three cases:
    1. If $\text{mod}6(pN_p) = 4$ make two groups of 5 operations, and all group the remaining operations in groups of 6.
    2. If $\text{mod}6(pN_p) = 5$ make one group of 5 operations and group the remaining operations in groups of 6.

[6]The derivation of equation (5.32) was performed by the writer.

3. In any other case group every operation in groups of 6.

- If $\mu_{\max} = 1$ and $p = 2$, Make groups of 8 operations.
- If $\mu_{\max} = p = 1$, make groups of 15 operations.
- If $\mu_{\max} = 2$ and $p = 1$, make groups of 10 operations.
- If $\mu_{\max} = 3$ and $p = 1$, make groups of 8 operations.
- If $\mu_{\max} = 4$ and $p = 1$, make groups of 7 operations.

Besides the problem of computation of the indices of *true* values, another problem is the number constraints will that will arise from the Quine-McClusky method. The result is a sum of products, and every term in the sum adds another constraint for every constraint containing the reparameterized variable. This can certainly become a lot of resulting constraints. For example, if one reparameterizes variable $z_{2,1}$ on a processor of 5 operations, the number of extra constraints becomes 18 for every constraint containing $z_{2,1}$. Note that this only for one of the 10 variables, while the number of binary variables is only reduced by 3. To solve an instance of the MILP, the relaxation is solved per iteration of the branch-and-bound algorithm (Appendix B.4 or Wolsey [54]). To solve the relaxation of the MILP (Appendix B.4 page 116), the simplex method of Dantzig et al. [16] or an interior point method (Kojima et al. [35]) is used and has a worst case scenario running time exponential in the number of constraints. This means that using the Quine-McClusky algorithm for reparameterization might cause bad running times for solving the relaxation of the MILP.

6 Model-based Partitioning

Due to the $\mathcal{NP}-$hardness of the MPC-SMPL problem (section 3.3 page 25), it can happen that the regularly used branch-and-bound algorithm (Appendix B.4 page 116) cannot solve the MILP in a reasonable amount of time, even if the algorithm is strengthened (explained in Appendix B.5 page 120). Low running times are essential for on-line performance of the MPC controller in practice. Because the MPC controller cannot take forever to compute, worse than optimal solutions often have to be accepted for control. If one cannot find the optimal solution to the MPC-SMPL problem in the given time, one should find a clever way to find a solution as good as possible within the time limit. These solutions are referred as *sub-optimal* solutions.

The idea of *model-based partitioning* (MBP) arises from reordering the rows and columns of the MILP constraint matrices such that most non-zero elements (in conventional algebra) are in block-diagonal structure. All elements outside the block-diagonal structure should correspond to continuous variables. The idea is to set these variables fixed such that the optimization process can be solved for every block on the diagonal. Proposition 6.1 is minor result from the writer that shows the use of such a block-diagonal structure.

Proposition 6.1. *Consider an optimization problem of the form* $\min_{x \in X} c^T x$ *such that* $Ax \leq b$ *where* A *is block diagonal:*

$$A = \begin{bmatrix} A_1 & 0 & \cdots & 0 \\ 0 & A_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & \cdots & A_n \end{bmatrix}, \quad \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{6.1}$$

Then $x^\star = (x_1^{\star T}, \ldots, x_n^{\star T})^T$ *where* $x_i^\star$ *solves* $\min c_i^T x_i$ *with* $A_i x_i \leq b_i$ *is the optimal solution to the original optimization problem.*

Proof. The theorem is proven by induction. Clearly the theorem always holds for $n = 1$. Now suppose $n = 2$ and let $x_1^\star, x_2^\star$ be the optimal solutions to the problems defined by the two diagonal blocks. Suppose that $x^\star = (x_1^{\star T}, x_2^{\star T})^T$ is not the optimal solution to the original problem. This means that there is exists a $\hat{x} \in X$ such that $c_1^T \hat{x}_1 + c_2^T \hat{x}_2 = c^T \hat{x} < c^T x^\star = c_1^T x_1^\star + c_2^T x_2^\star$. This means that either $c_1^T \hat{x}_1 < c_1^T x_1^\star$ or $c_2^T \hat{x}_2 < c_2^T x_2^\star$ which means that either $x_1^\star$ or $x_2^\star$ is not an optimal solution to their corresponding sub-problems. This is a contradiction to what is assumed and so it follows that $x^\star = (x_1^{\star T}, x_2^{\star T})^T$ is optimal for the original problem. The proof for $n > 2$ follows from a repetitive argument on the blocks $\text{diag}(A_1, \ldots, A_{n-1})$ and A_n. □

It is rarely the case that a large optimization problem can be formulated as a nice block-diagonal constraint matrix as in (6.1). Model-based partitioning focuses on creating a block-diagonal structure with a minimum amount of non-zero entries outside the blocks on the diagonal. Kersbergen [34] showed that such a block-diagonal matrix can be found for controlling the dutch railway-network by solving a quadratic integer program. The result is a permutation of rows and columns into a near-block-diagonal structure of the constraint matrix. Once such a permutation is found, Kersbergen [34] uses a *distributed optimization method* to solve the new problem. In distributed optimization the problem is solved locally for some segments in order to improve the computation speed. In the results of Kersbergen [34], the optimization can be solved much faster, while the solution found is still close to the optimum. In this thesis, a general partitioning method will be composed for the MPC-SMPL problem.

6.1 Partitioning

The first step of the model-based partitioning method is setting up the constraint matrix into a near-block-diagonal form. Desired is a permutation of rows and columns that creates this block-diagonal form. There are a number of design choices in the selection of this permutation as will be explained in this section. First an optimization program will be constructed that describes the feasible region of possible permutations, and afterwards the objective function will be designed, to find the best permutation.
The original MILP (B.3)-(B.4) on page 110 has constraints of the form $Ax \leq b$, where in this notation x contains as well integer as continuous variables. This is called the *centralized problem*. The new partitioned formulation will be solved with a distributed optimization method, and this is called the *partitioned problem*. The partitioned problem consists of a pre-determined amount of *segments* which correspond to the blocks on the diagonal of the partitioned constraint matrix. Before continuing to the partitioning program, make note of the following assumptions.

- The number of segments or blocks on the diagonal is determined before the partitioning is started.
- All constraints and variables are assigned to exactly one non-empty segment. So the segments are disjoint and the union of all segments contains all constraints and variables.
- Every entry for an integer variable must be inside the block diagonal structure. The entries outside the block diagonal structure can only correspond to continuous variables.
- The objective function for the program must be a trade-off between minimizing the number of entries outside of block-diagonal structure, and the maximum difference in number of integer variables between segments.

Let p be the number of segments that is desired for partitioning. Let V be the set of variables, and C be the set of constraints. Consider the following binary variables.

$$z_{v,j} \begin{cases} 1 & \text{if variable } v \in V \text{ is assigned to segment } j \\ 0 & \text{else} \end{cases}$$

$$w_{a,j} \begin{cases} 1 & \text{if constraint } a \in C \text{ is assigned to segment } j \\ 0 & \text{else} \end{cases}$$

First, every variable and constraint must be assigned to exactly one segment. Therefore begin with the following constraints.

$$\sum_{j=1}^{p} z_{v,j} = 1 \quad \forall v \in V$$

$$\sum_{j=1}^{p} w_{a,j} = 1 \quad \forall a \in C$$

Next, all entries for integer variables cannot be outside of the block diagonal structure. This means that if a constraint has multiple entries corresponding to integer variables, all these variables must be assigned to the same segment. Let $S(a)$ be the set of variables that is contained in constraint $a \in C$. Now $S(a)$ can be disjunctive split up into $S_i(a)$ and $S_c(a)$ which contain the integer and respectively continuous variables of constraint a. To eliminate entries for integer variables outside the block diagonal structure, add the constraint (6.2) to the program.

$$\sum_{i \in S_i(a)} z_{v,j} = |S_i(a)| w_{a,j} \quad \forall a \in C, j = 1, \ldots, p \tag{6.2}$$

The feasibility of the resulting constraints is now equivalent to partitioning following the made assumptions. However, it is left to construct a correct objective function, and some extra variables are required for this. Let T_1 be the maximum number of integer variables in a segment, and let T_2 be the minimum. To make these variables correct, add the constraints

$$\sum_{v \in V \text{ integer}} z_{v,j} \leq T_1 \quad j = 1, \ldots, p$$
$$\sum_{v \in V \text{ integer}} z_{v,j} \geq T_2 \quad j = 1, \ldots, p$$

To make sure they become exactly the maximum and minimum number of integer variables of a segment, T_1 and T_2 must be placed in the objective function correctly.
Next, it is desired to construct some variables that count the number of continuous variables outside of the block diagonal structure. Let variable Q_a denote the number of continuous variables outside the segment where constraint a is assigned to. In order to let Q_a take the correct values add constraint (6.3) to the partitioning program.

$$w_{a,j}|S_c(a)| - \sum_{v \in S_c(a)} z_{v,j} \leq Q_a \quad \forall a \in C, \ j = 1, \ldots, p \tag{6.3}$$
$$Q_a \geq 0 \quad \forall a \in C$$

The objective function that satisfies the last assumption now is given by

$$u = \min \ \alpha(T_1 - T_2) + \sum_{a \in C} Q_a$$

with $\alpha \geq 0$. The objective function forces the variables T_1 and T_2 to be the maximum and respectively minimum number of integer variables of the segments in the optimal solution. Likewise, Q_a becomes the number of entries in constraint a outside the block diagonal structure in the optimal solution. The program for partitioning the centralized problem into p segments is now complete.

One might argue that this is a difficult optimization problem since it is an MILP with a large amount of binary variables. This is certainly true. If constraint matrix A from the centralized problem is an $n \times m$ matrix, this program has $p(n + m)$ binary variables, which will likely cause a large running time. However, once completed, the partition obtained can be used to solve the partitioned problem in a continuous process. Although obtaining the optimal partition is a difficult problem, it only has to be computed once. In addition, this optimization problem can be solved before implementing in a continuous working MPC controller, so there is no on-line MPC deadline for solving the partitioning problem.

The question arises if it is always possible to find a partition to the centralized problem under the assumptions. Example 6.2 shows by a simple counterexample that this is not always the case.

Example 6.2. Consider the centralized problem under constraints (6.4).

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} x \leq b \tag{6.4}$$

Let $x \in \mathbb{Z}^4$ and $b \in \mathbb{R}^4$. It is desired to partition this problem into 2 segments. Notice that the first constraint must be placed in the same segment as the second and fourth constraint, otherwise there are resulting integer variables outside the block-diagonal structure. Now notice that the third constraint must also be placed in the same segment as the second and fourth constraint. It is concluded that the only feasible and therefore optimal partition is given by one segment containing all the constraints and variables, and the other segment is empty. An empty segment is in violation of the second assumption, and it is therefore not possible to find a partition under the assumptions.

Hence, why would one use model-based partitioning on the MPC-SMPL problem, if it is not known if it is even possible? The answer comes from the structure of events that are being processed on processors. Under the fixed processor case, notice that the MPC-SMPL problem has at most one integer variable per constraint if no reparameterization is applied. This means that partitioning under the assumptions is certainly possible. When reparameterizing a set of binary variables as explained in section 5, the resulting constraints will have multiple binary variables per constraint. The set of new variables must therefore be assigned to the same segment.

In most cases of the MPC-SMPL problem, there are only few routing constraints, but a lot of ordering constraints. The ordering constraints contain only variables corresponding to a certain processor. The optimal partition of a problem will therefore most of the time have all variables of an ordering constraint contained in the same block.
The last design step is to find a suitable value for α. This can be done by an iteratively tuning process. If α is large the optimization will be stressed on finding a partition with equally many integer variables per segment. For a small α the size of the blocks matters less and the number of variables outside the diagonal block structure is considered more important.

Definition 6.3. *Let c be the objective vector of an MILP. An* MILP *is said to be well-partitioned into p segments with respect to $\theta \in [0, 1]$ if*

$$\sum_{v \in V} c_v z_{v,j} \geq \frac{\theta}{p} \sum_{v \in V} c_v \tag{6.5}$$

for $j = 1, \ldots, p$ If an MILP *is not well-partitioned, it is said to be ill-partitioned.*[7]

An optimal partition might result in having all (or most) variables with positive objective weight in a single block. In the distributed optimization this will cause the solution to outcome rather arbitrary solution for some segments due to their insignificance. To make sure every segment has variables that directly influence the objective value, the partition should be well-partitioned. This can directly be obtained by adding inequality (6.5) directly as a constraint into the partition program. Now choosing $\theta = 1$ will likely result in infeasibility. Choose θ therefore iteratively as high as possible without causing mayor difference in block sizes of infeasibility.

6.2 Partition-based optimization

In the previous section it was explained how one could split the MPC-SMPL problem into a fixed number of segments. In almost any case, each processor is assigned to a segment, and each segment has its own model predictive controller. To ensure the solutions found by all controllers are feasible, it is required that the controllers communicate with each other. If each controller works independently, some events may be selected such that the solution of the centralized problem is infeasible. To prevent infeasibility of the centralized problem, the problem is

[7]This definition is written by the writer to qualify a partition.

solved for each segment while taking the solutions to the problems of other segments into account.

In order to obtain feasibility of the solution of the first segment, a starting point of the centralized problem is required. Any feasible solution of the centralized problem can be used to initialize the partitioned model predictive controller. However, since the partitioned solver is a local optimizer, it is recommended to use a starting point that already has an acceptable objective value. Obtain a good initial solution, the easiest and fastest way is to use a heuristic. Section 7 contains some heuristics that can be used to find an initial solution.
Once an initial solution is obtained, the distributed optimization can begin. Let the centralized problem be denoted by $Ax \leq b$, where $A = [A_1, A_2, \ldots, A_p]$ is split into p sub-matrices as a result of the model based partitioning. The objective function is $c^T x$. Let x_i be the vector that contains all the variables corresponding to segment i, and let c_i be the corresponding objective vector. To solve the distributed problem is solved as in the following steps. Here, I_v contains the indices of integer variables after partitioning.

Partition-based optimization

1. Initialize an initial solution $\hat{x}$ that satisfies $A\hat{x} \leq b$, $x_v \in \mathbb{Z}$ if $v \in I_v$, where $A = [A_1, A_2, \ldots, A_p]$ is a partitioned matrix. Define a time- and iteration limit.
2. For $i = 1, \ldots, p$ do:
   2.1. Solve the program $\min_{x_i \in P_i} c_i^T x_i$ such that $A_i x_i \leq b - \sum_{j \in \underline{p} \setminus \{i\}} A_j \hat{x}_j$ and $(x_i)_v \in \mathbb{Z}$ if $v \in I_v$.
   2.2. Set $\hat{x}_i := x_i$.
3. Repeat step 2 until $\hat{x}$ does not improve for all $i \in \underline{p}$ or the time- or iteration limit is reached.
4. Output $\hat{x}$ as sub-optimal solution to the centralized problem.

Notice when $c^T \hat{x}$ does not improve in step 2, there is no use of iteration till the time or iteration limit is reached. The sub-optimal solution $\hat{x}$ is in this case stuck in a *local minimum.* The time and iteration limit are there to prevent the algorithm from performing too many iterations.
The main goal of the distributed optimization is to get a good solution in a reasonable amount of time. The algorithm is likely to be much faster than the centralized problem due to the exponential complexity (Appendix B.3) of the problem. Especially for large instances, solving the centralized problem can take much longer than solving small partitioned segments a number of times. On the downside, the algorithm is a *local optimizer*, it does often get stuck in a local optimum. This means that it is unlikely that the output $\hat{x}$ will be a global optimum for the centralized problem. Using the partition-based optimization ultimately results in a trade of objective value for computation speed.

Example 6.4. This example is given to illustrate why the distributed solver does not always find the optimal value. Suppose one wants to minimize the following program.

$$\min 2x_1 - x_2 \quad \text{s.t.} \tag{6.6}$$
$$x_1 \geq x_2 \tag{6.7}$$
$$x_1 \leq 4 \tag{6.8}$$
$$x_2 \geq \frac{1}{2} \tag{6.9}$$
$$x_1 \in \mathbb{Z},\ x_2 \in \mathbb{R} \tag{6.10}$$

Consider the partitioned system where x_1 and constraints (6.7) and (6.8) are assigned to the first segment and x_2 and constraint (6.9) are assigned to the second segment. The resulting partitioned matrix and constraint vector are given by

$$A = \left[\begin{array}{c|c} -1 & 1 \\ 1 & 0 \\ \hline 0 & -1 \end{array}\right] \qquad b = \begin{bmatrix} 0 \\ 4 \\ -\frac{1}{2} \end{bmatrix}$$

Now select $\hat{x} = [3\ \frac{3}{2}]^T$ as the starting point of the partition-based optimization. In the first iteration, the first segment will be

$$\min 2x_1 \quad \text{s.t.} \tag{6.11}$$
$$-x_1 \leq 0 - \hat{x}_2 = -\frac{3}{2} \tag{6.12}$$
$$x_1 \leq 4 \tag{6.13}$$

which attains its optimum at $x_1 = 2$ so update $\hat{x} = [2\ \frac{3}{2}]^T$. Next, optimize the second segment which is given by:

$$\min -x_2 \quad \text{s.t.} \tag{6.14}$$
$$x_2 \leq 0 - (-\hat{x}_1) = 2 \tag{6.15}$$
$$-x_2 \leq -\frac{1}{2} \tag{6.16}$$

This program attains its optimum at $x_2 = 2$ so fix $\hat{x} = [2\ 2]^T$. Computing the next iteration will result in the exact same value of $\hat{x}$, so partition-based optimization algorithm stops and outputs $\hat{x} = [2\ 2]^T$. The resulting objective value of (6.6) is equal to 2. The optimal value to the centralized problem (6.6)-(6.10) is however attained at $x = [1\ 1]^T$ and has objective value equal to 1. The distributed solver did therefore not find the optimum by the starting point $\hat{x} = [3\ \frac{3}{2}]^T$. However, verify that that the starting point $\hat{x} = [3\ \frac{1}{2}]^T$ would find the optimal solution. Figure 6.1 shows the graphical interpretation of the centralized problem and the solution of the partition-based optimization from starting point $[3\ \frac{3}{2}]^T$.

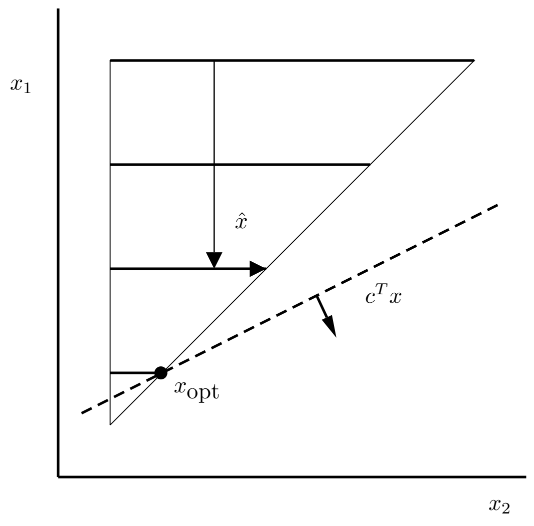


Figure 6.1: The centralized problem and the evolution of partition-based optimized solution $\hat{x}$ in example 6.4.

To find the right initial solution, one can use a heuristic. Examples of heuristics can be found in Section 7. What kind of heuristic is recommended, depends on the objective function of the centralized problem. If the centralized problem is to minimize the makespan of the system, it is advised to choose a heuristic that also is designed to minimize the makespan of the system. Notice that a multi-start version of the partition-based optimization could improve the best found solution. This would however be at the cost of computation time.

Part III
Bounds on the MPC-SMPL Problem

Part III of the thesis will focus on upper- and lower bounds on the MPC-SMPL problem. Both can be used to strengthen the branch-and-bound approach (Appendix B.4) as explained in Appendix B.5. The primal bounds come from heuristics, which can also be used to solve the MPC-SMPL problem in a practical matter.

7 Primal Bounds

A primal bound is a bound on the objective function of an optimization program that tells how good the optimal solution at least is. In case of a minimization, the primal bound is an upper bound and in case of maximization it is a lower bound. Primal bounds can be found by finding feasible solutions to the optimization program. Since the objective value of the optimum is at least as good as the objective value of a given feasible solution, the objective value of any feasible solution can be used as a primal bound.
In the case of the MPC-SMPL problem, the objective function $J_{N_p}(k)$ is supposed to be minimized, so any primal bound is an upper bound. To find feasible solutions for the MPC-SMPL problem, one can use *heuristics*. A heuristic generates a feasible solution in a very short amount of time. The goal is to find a feasible solution as close to the actual optimum as possible in this very limited amount of time. The found solution can of course be used for control, but most of the time it can be improved. It can be used to strengthen the branch-and-bound algorithm or it can be improved in a heuristic approach.

Before continuing to some heuristic algorithms, some terms will be explained first. Let v be the vector containing all the binary variables. From a binary value of v one can obtain an ordering and routing for the SMPL system. Every feasible ordering and routing has a graphical interpretation from a graph $G^\star(v)$. The set of vertices of $G^\star(v)$ is the set of operations (j, i) If the selected route of job j is $(i_1, i_2, \ldots, i_m)$, then $G^\star(v)$ contains the arcs $(i_1, i_2), (i_2, i_3), \ldots, (i_{m-1}, i_m)$. Additionally, if the order of operations on processor i is $(j_1, j_2, \ldots, j_n)$, the graph $G^\star(v)$ has the arcs $(j_1, j_2), (j_2, j_3), \ldots, (j_{n-1}, j_n)$.

Example 7.1. Suppose $J = \{1, 2, 3, 4\}$, $M = \{1, 2, 3\}$. The selected routes for the jobs of J are $l_1 = (1, 3, 2)$, $l_2 = (3, 1)$, $l_3 = (1, 2, 3)$ and $l_4 = (2, 3)$. The ordering is $(3, 1, 2)$ for processor 1, $(4, 3, 1)$ for processor 2 and $(1, 3, 4, 2)$ for processor 3. The graph $G^\star(v)$ is now given in figure 7.1.

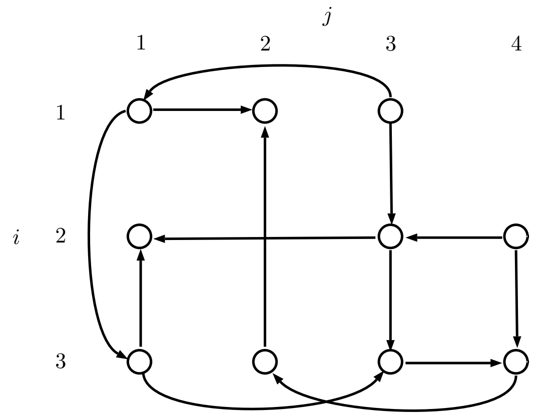


Figure 7.1: The graph $G^\star(v)$ from example 7.1.

7.1 Greedy algorithms

A *greedy* heuristic is a very fast algorithm that makes a series decisions based on what is the best choice to be made for every step. Any decision made cannot be changed in the future, which makes it very unlikely that a greedy solution will be optimal. It is also important to notice that a

greedy solution is not always feasible. It should always be verified if a greedy heuristic can result in infeasible solutions or not.
For the minimization of makespan ($\mathcal{C}_{\max}$) a simple but effective greedy heuristic is given by the *longest remaining processing time on other processors first* (LTRPOP) rule from Pinedo [44]. An operation (j, i) is said to be *available* if there is a route such that all processors before i are exactly the processors j already visited and completed. Now let $\mathcal{B}_j$ contain all the processors of j that have not processed yet. Whenever a processor i is idle, the algorithm selects the available operation for i with the longest total remaining processing time on other processors. When no processor is idle or no operation is available, move up in time until a processor becomes idle or an operation available. When multiple processors are idle, the next decision will be made first for the processor with the longest remaining processing time. The algorithm stops when all jobs are completed. This greedy heuristic always results in a feasible solution.

Example 7.2. Let $J = \{1, 2, 3\}$ be a set of jobs that should be processed on $M = \{1, 2, 3\}$. The first job has routes $(1, 2, 3)$ and $(1, 3, 2)$. The second job has routes $(2, 1, 3)$ and $(2, 3, 1)$. The third job only has route $(3, 2, 1)$. The processing times are given by $p_{ji} = [P]_{ji}$ with

$$P = \begin{bmatrix} 3 & 3 & 1 \\ 2 & 4 & 2 \\ 2 & 3 & 2 \end{bmatrix}$$

Applying the LTRPOP greedy heuristic will give the following results. First, all processors are idle, but the total remaining processing time is the highest on processor 2. The only available operation on this processor is job 2, so operation (2,2) is set to be processed. Likewise, job 1 is processed on 1, and job 3 is processed on 3. The next moment a processor becomes idle is at $t = 2$ on processor 3. There are no operations available, and all the other processors are busy, so there is no new operation to be processed. The next decision is made at time $t = 3$, and operations are only available for processor 3. The only other idle processor is 1, but has no operations available, thus job 1 is scheduled to processor 3. Now on $t = 4$ all processors become available. Since processor 2 has the longest remaining processing time, this processor will get an operation first. Job 2 is available with no remaining processing time on other processors, but job 3 has still a processing time of 2 on processor 1. It is therefore decided that job 3 is processed on processor 2. Next, processor 1 has the longest total remaining processing time, and it is assigned job 2 since (2,1) is the only available operation. From now on, every processor has only one operation to process so the rest of the schedule is trivial. The resulting solution is shown figure 7.2.

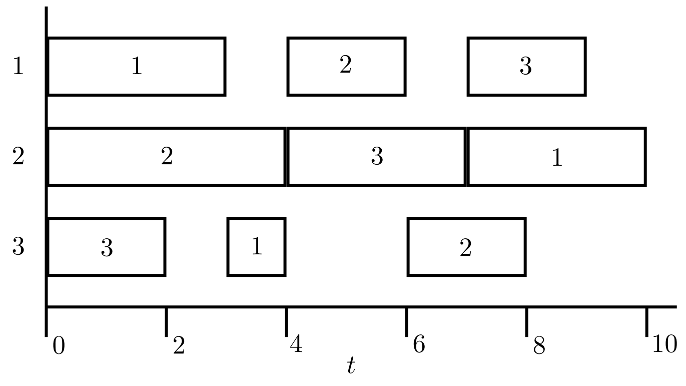


Figure 7.2: The resulting solution for example 7.2.

The makespan of the solution is $\mathcal{C}_{\max} = 10$, and is optimal in this case. It can be seen that this is optimal since the makespan cannot be smaller than the maximum total processing times of all jobs and processors. The total processing time of processor 2 is 10, so the makespan cannot be lower than 10. Note that in most cases the LTRPOP rule does not result in an optimal solution.

The LTRPOP rule is a so called *dispatching rule* that makes decisions based on the available operations. An overview of some well-known dispatching rules is given in Appendix C.2. The selection of a good dispatching rule depends on the objective function one wants to minimize. Also, the behaviour of an MPC controller on a SMPL system depends fully on the objective function (3.2). In the case of the minimization of makespan, only the latest completion time matters. If one minimizes the sum of completion times, the resulting events are probably much different from minimizing makespan. A good greedy heuristic for the generalized objective function would therefore depend on the weights δ, κ, α of the objective function (3.2). The *generalized greedy*[8] method, is a heuristic for the generalized objective function. Just like in LTRPOP, it processes an available operation on an idle processor, depending on a priority rule. The priority rule is based on the following 3 priority functions.
Suppose $y(k)$ is a combination of the completion times, and the makespan. The weights are δ_j for the completion times of jobs and $\delta_{\max}$ for the makespan. Let $\mathcal{I}_j(t)$ be the *idleness* of job j at time t, defined as

$$\mathcal{I}_j(t) = \begin{cases} 0 & \text{if } j \text{ just became available at time } t \\ 1 & \text{else} \end{cases}$$

The first priority function, is given in (7.1), where $\mathcal{P}_{\max} = \max_{j \in J} \sum_{i \in R_j} p_{j,i}$ and $\theta_1 \geq 0$ is a tuning parameter.

$$f_1(j,i,t) = \delta_j (1 + \theta_1 \mathcal{I}_j(t))(\mathcal{P}_{\max} - \sum_{m \in \mathcal{B}_j} p_{j,m}) + \delta_{\max} \sum_{m \in \mathcal{B}_j \setminus \{i\}} p_{j,m} \tag{7.1}$$

Function (7.1) is a balanced priority rule between the weighted sum of completion times and the makespan of the problem. The first term gives high priority to jobs with small total remaining processing times. When the operation just became available, it gets even a higher priority. The second therm gives jobs with high remaining processing times on only other processors high priority. Both terms are weighted by δ, since it determines the relation in weight in the final objective function.
The second priority function determines the weighted priority for the operations. It is given in equation (7.2).

$$f_2(j,i) = \kappa_{j,i} \left(\max_{q \in J, m \in R_q} p_{q,m} - p_{j,i} \right) \tag{7.2}$$

It makes sense to give operations with low processing times priority, since processing these operations first will result in processing the other available operations sooner. Of course, the weight $\kappa_{j,i}$ also determines the priority.
The last priority function looks at the tardiness of events. Define $\mathcal{D}_{j,i}$ as the remaining minimum processing time of an operation until it reaches a processor where it has a due date, and let $\sigma(i)$ be the processor where j has this due date. The priority is given by (7.3), where $\theta_3 > 0$ is a tuning parameter.

[8]The generalized greedy method works like any other dispatching rule, but has a priority rule based on a function designed by the writer.

$$f_3(j,i,t) = \begin{cases} 2\alpha_{j,\sigma(i)}(\mathcal{P}_{\max} - \mathcal{D}_{j,i})e^{\frac{t-d_{j,\sigma(i)}+\mathcal{D}_{j,i}}{\theta_3}} & \text{if } t \leq d_{j,\sigma(i)} - \mathcal{D}_{j,i} \\ \alpha_{j,\sigma(i)}(\mathcal{P}_{\max} - \mathcal{D}_{j,i})(1 + \frac{d_{j,\sigma(i)}-t}{\mathcal{D}_{j,i}}) & \text{if } d_{j,\sigma(i)} - \mathcal{D}_{j,i} < t \leq d_{j,\sigma(i)} \\ \alpha_{j,\sigma(i)}(\mathcal{P}_{\max} - \mathcal{D}_{j,i}) & \text{if } t > d_{j,\sigma(i)} \end{cases} \tag{7.3}$$

When $d_{j,\sigma(i)} - t - \mathcal{D}_{j,i}$ gets close to zero as time progresses, the more likely j will complete close to its due date. The value $d_{j,\sigma(i)} - t - \mathcal{D}_{j,i}$ is also said to be the *slack* of j when entering processor i. When the slack becomes zero it is not possible anymore for j to be completed before its due date. Priority function $f_3(j,i,t)$ gives a balanced priority to operations with their slack close to zero, a low $\mathcal{D}_{j,i}$ and a high objective weight $\alpha_{j,\sigma(i)}$ of the next due date. The smaller $\mathcal{D}_{j,i}$ is, the higher the priority around the point $t = d_{j,\sigma(i)} - \mathcal{D}_{j,i}$. The emphasis is also put on jobs with a slack close to zero, since they make the difference between a schedule with or without a penalty. In the case a job cannot be completed in time it loses priority over time. When time reaches the due date the penalty of this job is similar to a completion time penalty. Jobs in this spectrum should be prioritized like a weighted sum of completion times. That is why low $\mathcal{D}_{j,i}$ and/or high $\alpha_{j,\sigma(i)}$ are given priority, regardless of the time. Finally, the tuning parameter $\theta_3 > 0$ determines the importance of jobs that can still meet their due date. With low θ_3, jobs with a negative slack are almost in any case more important than jobs with a negative slack. With a large θ_3, jobs with a slack close to zero become more important, regardless if the slack is negative of positive. A graphical representation of $f_3(j,i,t)$ is shown in figure

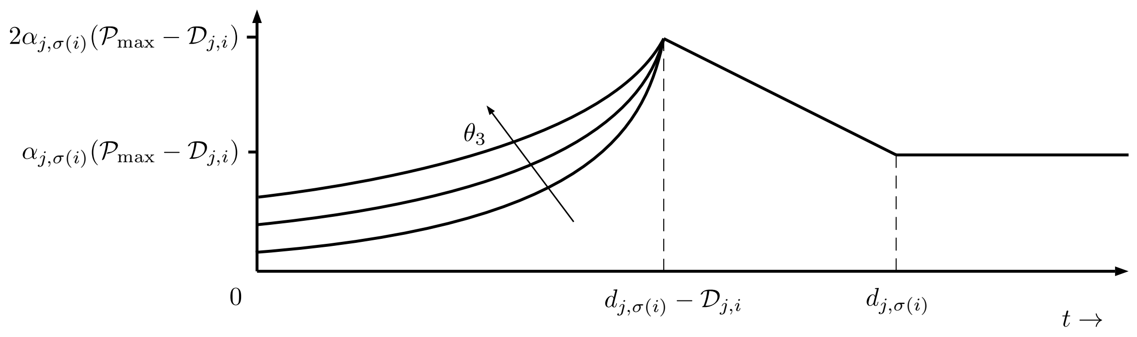


Figure 7.3: Function $f_3(j,i,t)$ from (7.3).

Eventually, the priority functions (7.1), (7.2) and (7.3) are combined to the generalized priority function (7.4).

$$f(j,i,t) = f_1(j,i,t) + f_2(j,i) + f_3(j,i,t) \tag{7.4}$$

The generalized greedy heuristic now processes the operation (j,i) with the highest value $f(j,i,t)$ whenever a processor i is idle at time t. Whenever multiple processors are idle, the algorithm first chooses to determine an operation for the processor with the longest total remaining processing time.

Greedy algorithms are very fast, but often not the best solution. They are in many cases, used as a starting solution for other heuristic algorithms. Since the MPC-SMPL problem is not convex (Appendix B.2 for definition or Barvinok [8]) due to the presence of integer variables, it can be beneficial to use a *multi-start* optimization. In a multi-start optimization an algorithm is solved multiple times, each time with a different starting solution. The greedy heuristics in this chapter so far are deterministic, so they will result in the same starting point every time. To generate multiple starting solutions, the *randomized generalized greedy* method is introduced. Consider the same priority function (7.4) as before. In this method, whenever i is idle at time t, a random operation (j, i) is chosen to process on i with probability

$$\frac{f(j,i,t)}{\sum_{\hat{j} \text{ available}} f(\hat{j},i,t)}$$

This means that the probability that (j, i) will be chosen as the next operation on i is proportional to the priority of (j, i). Whenever multiple processors are idle, choose to evaluate processor i with probability

$$\frac{\text{total remaining processing time on } i}{\text{total remaining processing time on all idle processors}}$$

With this randomized greedy heuristic multiple starting points can be generated for multi-start optimization.

7.2 Tabu search

Originally found by Glover [24, 25], *tabu search* is a heuristic algorithm for combinatorial and integer optimization. The algorithm is similar to a local search heuristic, but with extra features. In local search, one defines a *neighbourhood* of a feasible solution $\hat{x}$. The neighbourhood is a set of feasible solutions that are similar in a way to $\hat{x}$. What the relation between $\hat{x}$ and its neighbours is, is defined by the structure of the neighbourhood and is to be designed by the algorithm designer. The local search heuristic finds the optimal solution $\tilde{x}$ of the neighbourhood and sets $\hat{x} := \tilde{x}$. This process is repeated until a certain stopping criteria is met.
The problem with local search in most problems is that it gets stuck in a *local minimum*. A local minimum $\hat{x}$ is a feasible point for which there is a neigbourhood $N(\hat{x})$ such that $N(\hat{x})\backslash\{\hat{x}\} \neq \emptyset$ and $\hat{x}$ is the optimal solution in $N(\hat{x})$. The difference between local search and tabu search is that tabu search keeps track of a list of characteristics of previous solutions that are excluded from the neighbourhood. The structure of these characteristics can be determined by the algorithm designer. In tabu search, the best solution from the neighbourhood of $\hat{x}$ (excluding $\hat{x}$ itself) is chosen for the next iteration. Due to the tabu list it is not possible to directly fall back into the local optimum. This makes it possible for the tabu search algorithm to escape local minima, and continue the search for good solutions. A heuristic that can escape local minima is called a *meta-heuristic* (Glover & Kochenberg [26] and Wolsey [54]). The best solution found in the entire search is the final output of the algorithm.

The neighbourhood

One can define a neighbourhood structure for the MPC-SMPL problem by changing some routing and/or ordering variables. Liaw [38] suggests a good neighbourhood structure for the $Om||\mathcal{C}_{\max}$ problem, but the MPC-SMPL problem is not exactly an open shop scheduling problem, and it does not always minimize the makespan. The neighbourhood structure for this tabu search heuristic will be quite similar, but undergo some adjustments.
For the neighbourhood of a given solution in the form of a graph $G^{\star}(v)$ consider only arcs that describe the order of two operations on the same processor i_1. Let (j_1, i_1) in k_1 be processed directly before (j_2, i_1) in k_2. For this arc, there are potentially four alteration moves to create neighbourhood solutions, formulated by Liaw [38]. The first move, which is always possible, is to simply reverse the order of the two operations between the given arc. Note that in order to remain a feasible ordering the arcs adjacent to the two given operations must also be adjusted. This is done such that the rest of the order on the specific processor remains the same. The

adjustment is illustrated in figure 7.4, where the striped arcs are the initial arcs, and the full line arcs denote the new arcs.

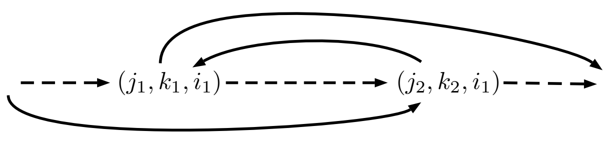


Figure 7.4: The first move to create a neighbour.

If possible, the second move also includes reversing the ordering of the given arc, but additionally change the route of j_1. In this move, the operation (j_1, i_1) in k_1 moves one up in the route of j_1 in k_1. This means that (if possible), j_1 is processed on another processor first before it continues to processor i_1. This move is shown in figure 7.5.

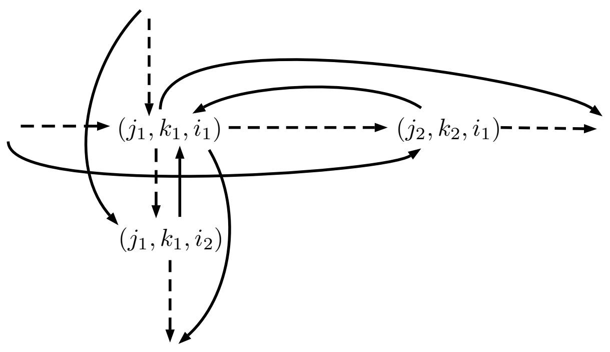


Figure 7.5: The second move to create a neighbour.

The third move is somewhat similar to the second move, but instead the route of j_2 in k_2 is changed. Processor i_1 is now chosen one processor earlier in the new route. An illustration of move 3 is shown in figure 7.6. Again, this move can only be performed if the possible routes of job j_2 allow this change.

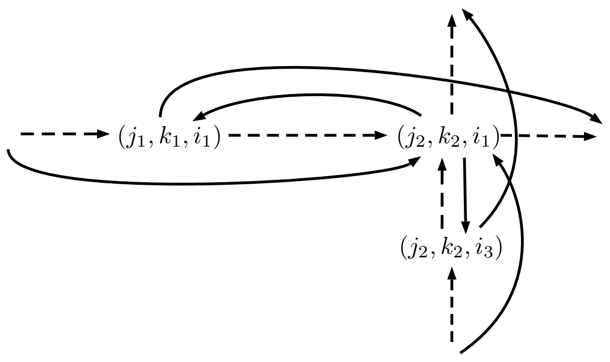


Figure 7.6: The third move to create a neighbour.

The last move, is a combination of the second and third move. Here, if possible, the change of routes of both jobs are taken to form a new graph $G^\star(v)$. This move is shown in figure 7.7.

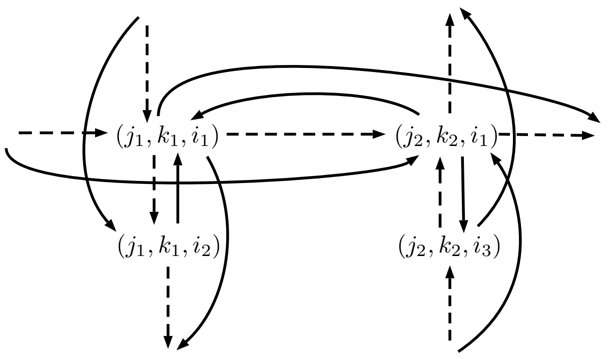


Figure 7.7: The fourth move to create a neighbour.

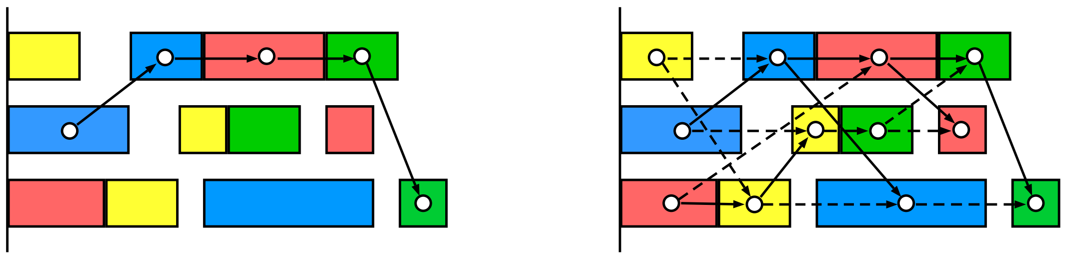
It is important to note that a move is only possible if $G^\star(v)$ remains acyclic. It makes sense to apply these moves because when the order of two operations interchanges, there might appear or disappear time to process one of the corresponding jobs on another processor. It is therefore sometimes beneficial to also change the route of jobs when it is ordered differently on a processor. If a neighbourhood of a solution is very large, it could take too much time to find the best neighbour. To make tabu search efficient, it is therefore desired to not make the neighbourhood structure too large. Evaluating every possible move for every arc of ordering in $G^\star$ would become too big of a neighbourhood. To keep the neighbourhood small, a subset of arcs is selected for the evaluation of neighbours.
Before showing how the subset of arcs will be chosen, some terms are explained to help define the structure of this subset of arcs. Define a weight function on the arcs of $G^\star(v)$ by $w^\star : D(G^\star(v)) \to \mathbb{R}$ with

$$w^\star(((j_1, k_1, i_1), (j_2, k_2, i_2))) = x_{j_2,i_2}(k_2) - (x_{j_1,i_1}(k_1) + p_{j_1,i_1}(k_1)) \tag{7.5}$$

For the rest of this section, it is assumed that in any solution there no operations postponed while it could have started earlier.

Definition 7.3. *Let $w^\star$ be defined as in (7.5) on a graph $G^\star(v)$. The critical path of job j is defined as the longest path in $G^\star(v)$ with only arcs of weight $w^\star = 0$, and with the last operation on the last processor of j as the end of the path. The critical forest of $G^\star(v)$, is the union of all the critical paths of all the jobs.*[9]

Figure 7.8a shows an example of a critical path and critical forest. The jobs are indicated by a colour, and are being processed on 3 processors. All the arcs in figure 7.8b are the arcs of $G^\star(v)$. The critical forest consists only of the non striped arcs.



(a) A critical path (b) $G^\star(v)$ and its critical forest.

Figure 7.8: A critical path and the critical forest.

[9]The definition of a critical path was formulated by Liaw [38], the extension to a critical forest was written by the writer.

Notice that the critical path of j defines exactly the series of operations for which j is waiting before it can be completed. Now let $\hat{v}$ be the vector containing all the binary variables of a solution $\hat{x}$. The neighbourhood for this tabu search algorithm consists of all the possible moves on all the arcs in the critical forest of $G^\star(\hat{v})$, where the operations connected by these arcs have the same processor. This is significantly smaller than all the arcs in $G^\star(\hat{v})$ for large instances, and gives a more accurate description of the set of operations that cause a congestion in the system.

The tabu list

To prevent the algorithm of getting stuck in local minima, the same tabu list of Liaw [38] is implemented. The tabu list on this algorithm is very simple: The arc of the critical forest that was selected for the last move, now has a reversed arc in $G^\star(v)$, which is put on the tabu list. In other words, the arc that has been reversed cannot be reversed again.
The length of the tabu list (denoted N_{tabu}) is highly important for the algorithm. If the list is too short, cycling of the search will often occur, and this may prevent the search from escaping local minima. If the list is too long, the algorithm might slow down because the list uses a lot of memory. A too long list may also cause a neighbourhood to be empty. The size of a neighbourhood, and the likelyness to escape local minima depends on the size of the problem. The length of the tabu list is therefore left as a tuning parameter.
When the tabu list is full, an arc needs to leave the list upon every new entry on the tabu list. The arc leaving the tabu list is selected by the first-in-first-out (FIFO) rule. So the arc that is in the list the longest will leave the list upon a new entry in a full list.

Search strategy

The first step of the tabu search algorithm is to generate a feasible starting solution for the search. For this, a greedy heuristic from section 7.1 is used. These algorithms are fast, and always generate a feasible solution, and the tabu search algorithm itself can now begin.
Once evaluating the tabu search, a few complications might occur. It might happen that the entire neighbourhood of the current solution is tabu, in this case the oldest possible reversion of all the moves in the tabu list is performed.
If the search does not improve for I_{bt} iterations, one can apply *backtracking* (Liaw [38]). If so, the solution jumps to the best solution got from this starting point but keeps the current tabu list. The idea is that the search will try to escape a local minimum with new information.
Another strategy that is often used in tabu search is *restarting* (Liaw [38]). Here, in the case that the solution does not improve for I_{re} iterations, starts over with a new starting point and an empty tabu list.

The stopping criterion

The tabu search algorithm can continue its search forever. In any applications of SMPL systems, it is ultimately required that there is a time limit such that the prediction horizon recedes for control. There must therefore always be a stopping criterion in the form of a time limit T_{lim}, but what other stopping criterion can be used?
Even though tabu search is a heuristic designed to escape local minima, it is still possible a search does not improve. There is always a possibility that cycling of solutions occurs in the search. It is therefore wise to implement an iteration limit I_{lim}. When the iteration limit is reached, backtracking or restarting can be performed.

7.3 Simulated annealing

Another local search based heuristic is *simulated annealing* described in Van Laarhoven & Aarts [51]. Just as in tabu search, it is able to escape a local minimum. The main difference with tabu search is that simulated annealing chooses its neighbours randomly. Because doing so already enables the heuristic to escape local minima, there is no tabu list required. The process in selecting neighbours is designed such that selecting a good neighbour has a higher probability than selecting a bad neighbour. The probability of selecting a neighbour also depends on a temperature schedule.
Let $T > 0$ be an initial parameter, called the *temperature*. The neighbourhood used of the MPC-SMPL problem is the same neighbourhood as used for tabu search in 7.2. From an initial solution $\hat{x}$, a random neighbour x is selected. If the neighbour is a better solution set $\hat{x} := x$. If not make the following decision.

$$\hat{x} := \begin{cases} x \text{ with probability } e^{\frac{c^T x - c^T \hat{x}}{T}} \\ \hat{x} \text{ with probability } 1 - e^{\frac{c^T x - c^T \hat{x}}{T}} \end{cases}$$

This process is repeated I_Q times with $I_Q \in \mathbb{N}_+$. With a high T it is more likely worse than the current solutions are chosen, but the probability is still proportional to the quality of the solution. After I_Q operations, if the stopping criteria is not met, decrease the temperature by setting $T := rT$ for $r \in (0, 1)$. Then the process of selecting neighbours is repeated again, but it will now less often select worse neighbours since the temperature T is decreased. When the stopping criteria is met, the best solution found in the whole process is be output of the algorithm. A schematic diagram of the algorithm is shown in figure 7.9.

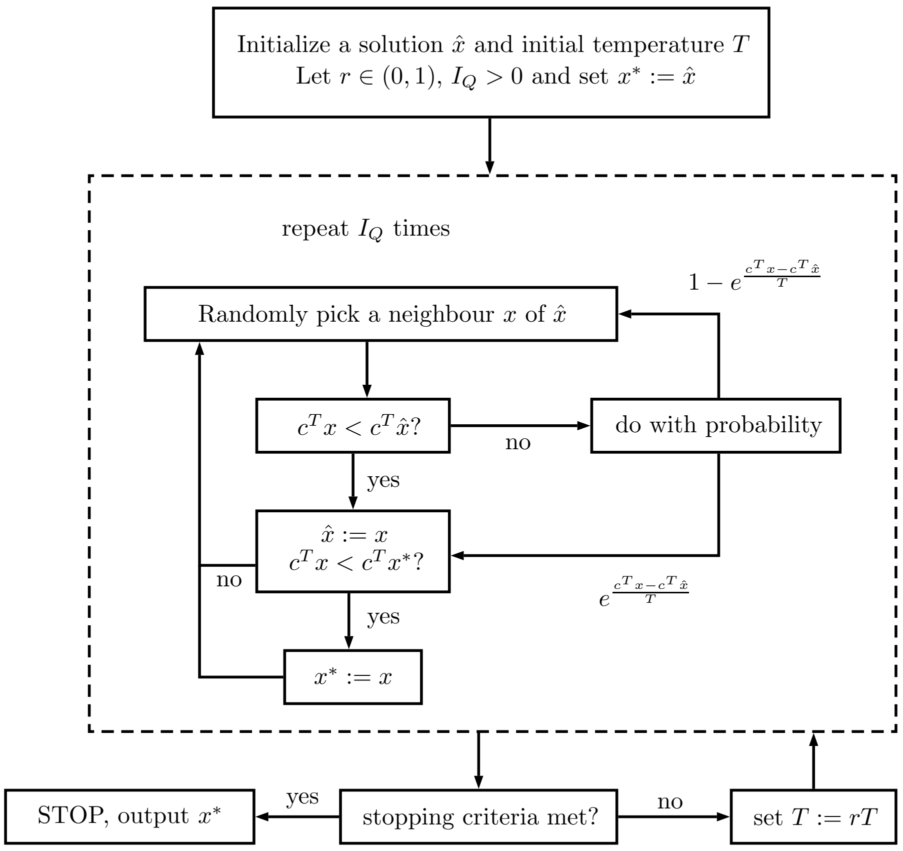

Initialize a solution $\hat{x}$ and initial temperature T
Let $r \in (0,1)$, $I_Q > 0$ and set $x^* := \hat{x}$
repeat I_Q times
$1 - e^{\frac{c^T x - c^T \hat{x}}{T}}$
Randomly pick a neighbour x of $\hat{x}$
$c^T x < c^T \hat{x}$?
no
do with probability
yes
$\hat{x} := x$
$c^T x < c^T x^*$?
no
$e^{\frac{c^T x - c^T \hat{x}}{T}}$
yes
$x^* := x$
STOP, output x^*
yes
stopping criteria met?
no
set $T := rT$

Figure 7.9: Flow diagram of the simulated annealing algorithm

Selecting a neighbour

In this thesis two ways of selecting neighbours are discussed. The first method selects a random neighbour uniformly. Each neighbour in the neighbourhood as an equal probability of being selected. The other method is to select them with probability proportional to their objective values. The probability of a neighbour being selected is equal to its objective value divided by the sum of all objective values of all neighbours. The advantage of the first method is that one does not have to compute all the objective values of the neighbourhood, which saves a lot of time. However, it is more likely that bad neighbours are selected, which is not desired. The second method prioritizes good neighbors, such that they are more often selected. However, this method invests time in computing the objective value of neighbours, which makes the algorithm slower.

8 Dual Bounds

A dual bounds gives an indication how good the optimal solution to a problem at most can be. The tighter the dual bound is, the better it estimates the objective value of the optimal solution. Dual bounds are mainly used to prove a solution is optimal. Since the MPC-SMPL problem is a minimization, dual bounds are lower bounds in this case.
one can compute lower bounds by solving the relaxation of the MILP. A stronger formulation (see section B.2 for definition) for the MILP provides a better bound. One can strengthen the relaxation by adding cutting planes (see definition B.13). Ideally one finds the FDI's (definition B.15) for the problem.

A part of the variables in the MPC-SMPL problem are ordering variables. Bolotashvili et al [9] and Doignon and Rexheb [20] found some sets of FDI's for ordering polytopes, which describe the feasible solutions for ordering variables in the MPC-SMPL problem. Both articles point out that the 3-length circuit elimination constraints in (5.14) and (5.15) on page 43 are FDI's of the ordering polytope.
The next two subsetions will develop cutting planes for the MPC-SMPL problem other than cuts for ordering variables, since there is already a lot of research devoted to obtaining the FDI's of the ordering polytope. For more details and more FDI's on the ordering polytope one can read Bolotashvili et al [9] and Doignon & Rexheb [20].

One can measure the strength of cutting planes by comparing the solution the relaxation with the optimal solution. Dividing the relaxed optimal objective value by the actual optimal value is called the *integrality gap*. An integrality gap close to 1 means the used formulation is strong.

In the case one has to deal with exponential many cutting planes it is not a good idea to implement them all. This will result in a high computational load on the solver for instances of a reasonable size. The right approach is to develop a technique that recognizes which cutting plane is violated by the relaxed optimal solution. One can then add only that cutting to the formulation and solve the relaxation again to repeat the process. There are techniques that can do this repeatedly until the integrality gap becomes 1 (Wolsey [54]) and the problem is directly solved, but this can take an unpractical amount of time. It is therefor best to add cutting planes until it becomes too time consuming to improve te relaxation.

8.1 Cuts for starting times

In this section, a few cutting planes designed by the writer are developed that can be applied to the MPC-SMPL problem. They take only a part of the SMPL polytope (Definition in Appendix B.2 page 111) in consideration, so their effectiveness on the entire problem might be less than indicated in this section.

Case study: 2 operations on 1 processor

Suppose there are two operations to be scheduled on a single processor. Let p_1 and p_2 be their processing times. The possible schedules are given by the feasible region of (8.1-8.4).

$$x_2 \geq x_1 + p_1 + \beta(1 - z_{12}^b) \tag{8.1}$$
$$x_1 \geq x_2 + p_2 + \beta z_{12}^b \tag{8.2}$$
$$x_1, x_2 \geq 0 \tag{8.3}$$
$$z_{12}^b \in \{0, 1\} \tag{8.4}$$

Now suppose one wants to minimize the objective function $J = \delta^T y + \kappa^T x$ with $y_j = x_j + p_j$ and $\kappa, \delta \in \mathbb{R}_+^2$. When relaxing feasible region to all continuous solutions, the relaxation is solved by $x_1 = x_2 = 0$ and $-\frac{p_2}{\beta} \leq z_{12}^b \leq 1 - \frac{p_1}{\beta}$ if β is sufficiently large enough in absolute value. This solution is of course no feasible solution for the original problem, since z_{12}^b is not integer in the relaxed solution and two operations are processed simultaneously on the same processor. Hence, the relaxation can be improved. To improve the relaxation, add constraint (8.5) to formulation (8.1-8.4).

$$p_1 x_1 + p_2 x_2 \geq p_1 p_2 \tag{8.5}$$

The graphic interpretation of this cut is shown in figure 8.1. Notice that inequality (8.5) separates the previous solution from the feasible region. In fact, the new relaxed optimum will be either $x = [0\ p_1]^T$ or $x = [p_2\ 0]^T$ depending on the values p_1, p_2, δ and κ. The objective value is in this case optimal, so the integrality gap is 0. However, z_{12}^b can still take non-integer values in this solution. So the cutting plane (8.5) provides a solid lower bound, but does not necessary force the relaxation to find the optimal solution.

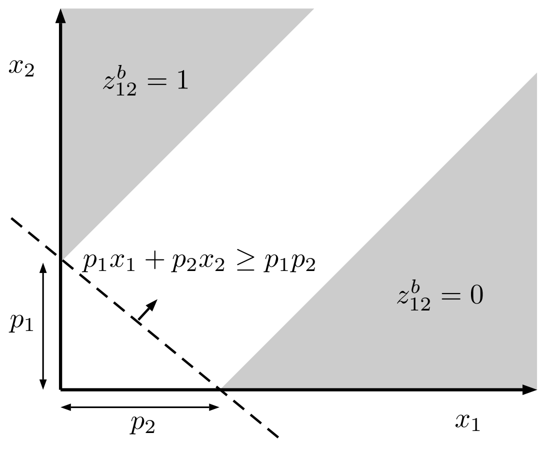


Figure 8.1: The feasible region of (8.1-8.4) and the cut (8.5).

To improve the cut in the sense that it also cuts values of z_{12}^b, one needs to take a look at the 3-dimensional feasible region, including the axis of the variable z_{12}^b. Clearly (also seen from figure 8.1), the points $(p_2, 0, 0)$ and $(0, p_1, 1)$ are feasible points for (x_1, x_2, z_{12}^b). They are tight with three linearly independent inequalities of (8.1-8.4) if taken $0 \leq z_{12}^b \leq 1$ instead for (8.4). This means that they are extreme points of the feasible region. Now to find another extreme point suppose $z_{12}^b = 0$ but (8.1) is tight. It follows that $x_2 = x_1 + p_1 + \beta$ and since β is chosen sufficiently large enough in absolute value, (8.2) cannot be tight. Also, $z_{12}^b \geq 0$ is tight, and if $x_1 = 0$ then $x_2 \leq -p_2$ which is not possible. The last inequality left is $x_2 \geq 0$, so setting $x_2 = 0$ gives the extreme points $(-\beta - p_1, 0, 0)$. Similarly, the extreme points $(0, -\beta - p_2, 1)$ can be found. A graphical interpretation of the feasible region and the found extreme points is shown in figure 8.2.

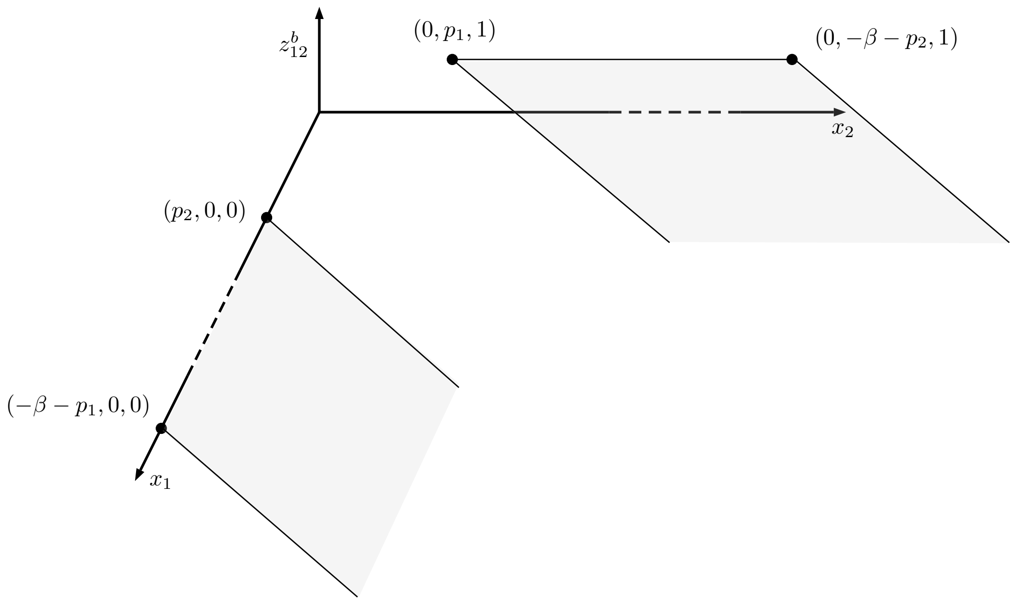


Figure 8.2: Feasible region of 2 operations and their ordering variable on the same processor.

Since the event timings are always to be minimized, only 2 candidate cuts are considered. For the first cut, consider the points $(p_2, 0, 0)$, $(-\beta - p_1, 0, 0)$ and $(0, p_1, 1)$. Desired is the hyperplane that contains these 3 points. To determine the angle of this hyperplane, consider the cross product

$$\left(\begin{bmatrix} p_2 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ p_1 \\ 1 \end{bmatrix}\right) \times \left(\begin{bmatrix} -\beta - p_1 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ p_1 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} p_2 \\ -p_1 \\ -1 \end{bmatrix} \times \begin{bmatrix} -\beta - p_1 \\ -p_1 \\ -1 \end{bmatrix} \tag{8.6}$$

$$= \begin{bmatrix} p_1 - p_1 \\ p_2 + \beta + p_1 \\ -p_1 p_2 - \beta p_1 - p_1^2 \end{bmatrix} = (\beta + p_1 + p_2) \begin{bmatrix} 0 \\ 1 \\ -p_1 \end{bmatrix}$$

Now take any of the given points, for example $(p_2, 0, 0)$. It follows that

$$[p_2 \; 0 \; 0] \left((\beta + p_1 + p_2) \begin{bmatrix} 0 \\ 1 \\ -p_1 \end{bmatrix} \right) = 0$$

and thus is 0 the constant determining the position of the hyperplane. The resulting hyperplane is now given by $x_2 - p_1 z^b_{12} = 0$. From the interpretation of the variables, the valid cutting plane is then

$$x_2 - p_1 z^b_{12} \geq 0 \tag{8.7}$$

A similar process can then be performed to find another hyperplane for the points $(p_2, 0, 0)$, $(0, p_1, 1)$ and $(0, -\beta - p_2, 1)$. The result in the valid cutting plane

$$x_1 + p_2 z^b_{12} \geq p_2 \tag{8.8}$$

The cuts can be seen in figure 8.3.

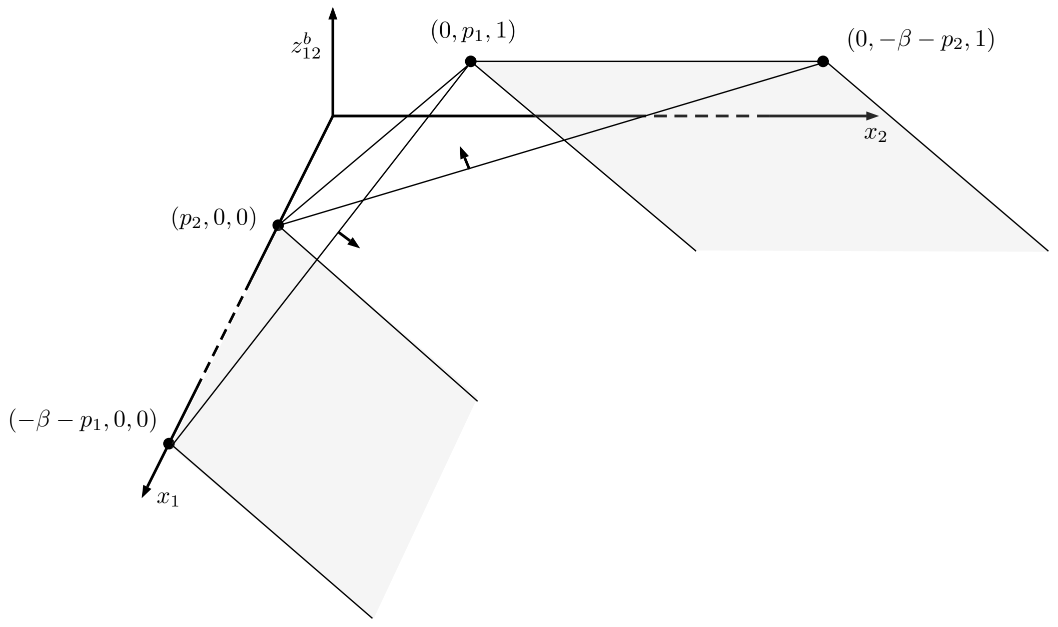


Figure 8.3: 2 cuts for (8.1-8.4).

Proposition 8.1. *The cutting plane* (8.5) *is redundant*[10] *to*[11]

$$\{(x_1, x_2, z_{12}^b) \ : \ x_2 - p_1 z_{12}^b \geq 0, x_1 + p_2 z_{12}^b \geq p_2\} \tag{8.9}$$

Proof. The intersection of the hyper planes $x_2 - p_1 z_{12}^b = 0$ and $x_1 + p_2 z_{12}^b = p_2$ is found by solving these equations. It follows that $z_{12}^b = \frac{x_2}{p_1}$. substituting into the second equation results into

$$x_1 + p_2 \frac{x_2}{p_1} = p_2 \Rightarrow p_1 x_1 + p_2 x_2 = p_1 p_2$$

If both of the inequalities of 8.9 are valid but one is not tight, x_1 or x_2 can only grow larger, and thus is (8.5) still valid. This concludes that (8.5) is redundant to (8.9). □

From proposition 8.1 it follows that it is at least as beneficial to add cutting planes (8.7) and (8.8) instead of (8.5).

Case study: 3 operations on 1 processor

Cutting plane (8.5) can be extended to the case where there are 3 operations on a single processor. Consider the 6 values for x ($3! = 6$ possible orderings) where the processor is not idle from the beginning until all operations are completed. These points are shown in figure 8.4. These 6 points are the candidates for the optimal solution of the MILP. However, just as in the case with 2 operations, the relaxation of the problem allows the solution $x = [0\ 0\ 0]^T$.

[10]Redundancy of inequalities is explained in Appendix B.2 page 112.
[11]Proposition 8.1 was written and proved by the writer.

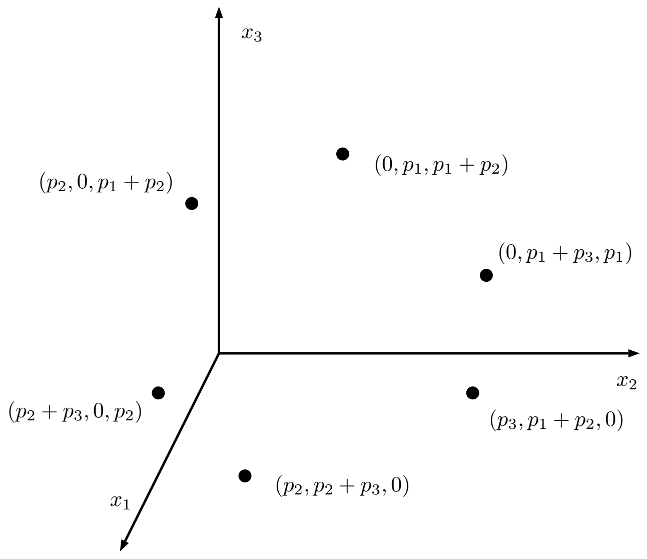


Figure 8.4: The six minimal candidates, for 3 operation on a single processor.

It appears to be the case the six points of figure 8.4 lie on a plane, no matter the values of p_1, p_2, p_3. This can be seen from the fact that if one extends a line through every pair of points representing the schedule with the same starting operation, every line will cross with the other lines. For example, verify that the line through $(0, p_1, p_1 + p_2)$ and $(0, p_1 + p_3, p_1)$ crosses the x_2 axis at $(0, p_1 + p_3 + \frac{p_1 p_3}{p_2}, 0)$. Then notice that so does the line through $(p_2, p_2 + p_3, 0)$ and $(p_3, p_1 + p_2, 0)$. This makes separating the relaxed optimal a lot easier. To separate the point $x = [0\ 0\ 0]^T$ from the relaxed polytope, inequality (8.10) is therefore added to the MILP.

$$p_1 x_1 + p_2 x_2 + p_3 x_3 \geq p_1 p_2 + p_1 p_3 + p_2 p_3 \tag{8.10}$$

Note that all candidates are tight with this constraint, as they all lie on the same plane. This plane is shown in figure 8.4.

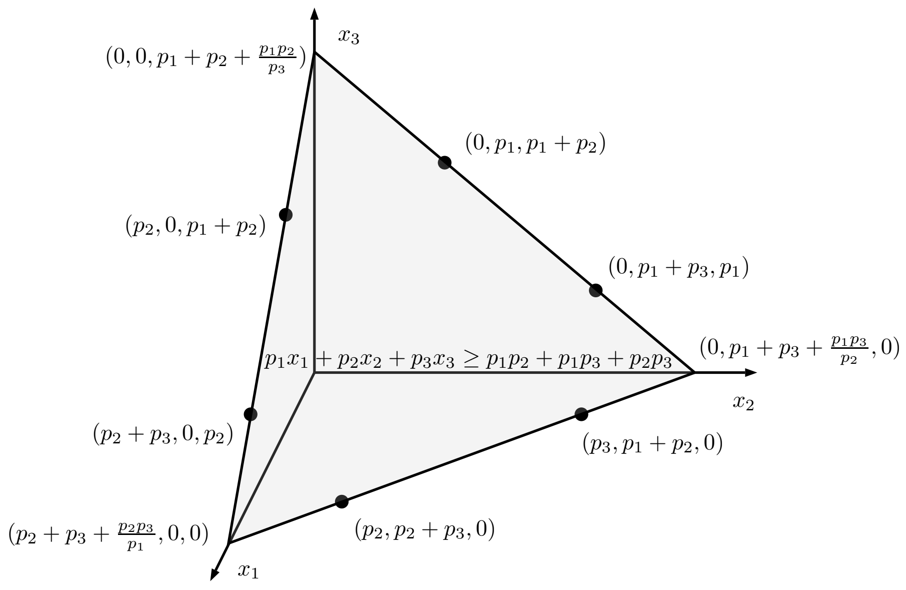

x_3
$(0, 0, p_1 + p_2 + \frac{p_1 p_2}{p_3})$
$(0, p_1, p_1 + p_2)$
$(p_2, 0, p_1 + p_2)$
$(0, p_1 + p_3, p_1)$
$p_1 x_1 + p_2 x_2 + p_3 x_3 \geq p_1 p_2 + p_1 p_3 + p_2 p_3$
$(0, p_1 + p_3 + \frac{p_1 p_3}{p_2}, 0)$
x_2
$(p_2 + p_3, 0, p_2)$
$(p_3, p_1 + p_2, 0)$
$(p_2 + p_3 + \frac{p_2 p_3}{p_1}, 0, 0)$
$(p_2, p_2 + p_3, 0)$
x_1

Figure 8.5: Separation cut (8.10)

The resulting polytope, does still not has the 6 candidates as extreme points. This can be seen graphically from figure 8.5, where the angles of the triangle represent the extreme points. In each of the extreme points, there are still two operations with starting time equal to 0. This means that these points can be separated using the previous constraint obtained in the case of 2 operations, for every pair of operations. Applying these cuts, the resulting formulation is graphically displayed in figure 8.6.

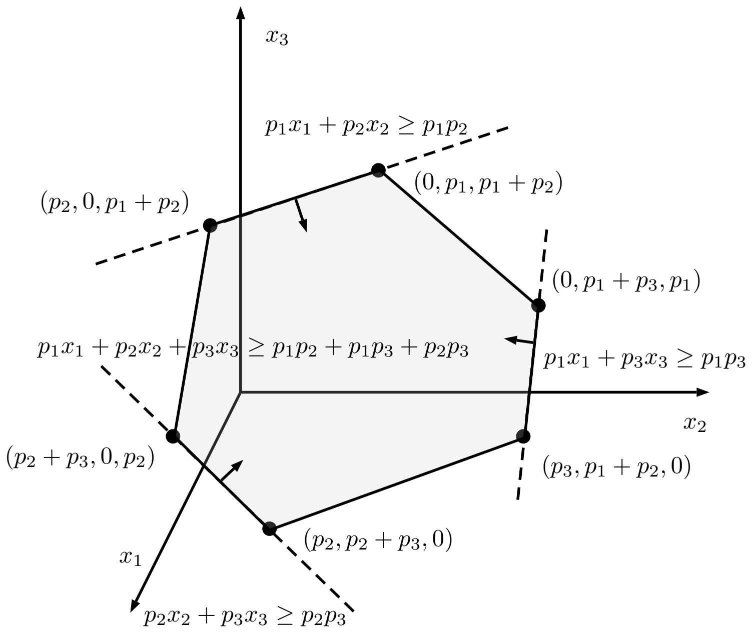

x_3
$p_1x_1 + p_2x_2 \geq p_1p_2$
$(0, p_1, p_1 + p_2)$
$(p_2, 0, p_1 + p_2)$
$(0, p_1 + p_3, p_1)$
$p_1x_1 + p_2x_2 + p_3x_3 \geq p_1p_2 + p_1p_3 + p_2p_3$
$p_1x_1 + p_3x_3 \geq p_1p_3$
x_2
$(p_2 + p_3, 0, p_2)$
$(p_3, p_1 + p_2, 0)$
$(p_2, p_2 + p_3, 0)$
x_1
$p_2x_2 + p_3x_3 \geq p_2p_3$

Figure 8.6: Separation cuts (8.5) and (8.10).

If one solves the relaxation now, depending on the values p_1, p_2, p_3, δ and κ, the optimal solution will coincide with one of the 6 candidate solutions. It is however still the case, that the ordering variables $z^b_{j_1j_2}$ can possibly not integer. So the formulation is not optimal, but its relaxation provides the optimal lower bound.

Case study: n operations on 1 processor

The single processor cuts for starting times can be easily extended tot n-dimensional cuts. To do so, apply Theorem 8.2.

Theorem 8.2. *Let there be n operations that have to be processed on a single processor. It holds that*

$$\sum_{j=1}^{n} p_j x_j \geq \sum_{j_1=1}^{n} \sum_{\substack{j_2=1 \\ j_2 \neq j_1}}^{n} p_{j_1} p_{j_2} \tag{8.11}$$

is valid for any feasible schedule and tight with all solutions where the processor is not idle until all operations are completed.[12]

Proof. Let the order of a solution be $(j_1, j_2, \dots, j_n)$. The starting time of j_r is at least $p_{j_1} + \ldots + p_{j_{r-1}}$. It then holds that

[12]Theorem 8.2 was written and proved by the writer.

$$
\begin{aligned}
\sum_{j=1}^{n} p_j x_j \geq p_{j_2} p_{j_1} & \\
& + p_{j_3}(p_{j_1} + p_{j_2}) \\
& + p_{j_4}(p_{j_1} + p_{j_2} + p_{j_3}) \\
& \quad \vdots \\
& + p_{j_n}(p_{j_1} + \ldots + p_{j_{n-1}}) \\
& = \sum_{j_1=1}^{n} \sum_{\substack{j_2=1 \\ j_2 \neq j_1}}^{n} p_{j_1} p_{j_2}
\end{aligned}
\tag{8.12}
$$

which shows that (8.11) is valid for any feasible solution. Now if the processor is non-idle until all operations are completed, the starting times are exactly $x_{j_r} = p_{j_1} + \ldots + p_{j_{r-1}}$. This will result in an equality in (8.12), and this shows that (8.11) is tight. □

Just like in the 3-dimensional case, applying cut (8.11) is not sufficient to let the extreme points coincide with the solutions forthcoming from a non-idle processor. Therefore, the lower-dimensional cuts are also required. Consider therefore a subset $S \subseteq J$, to apply

$$
\sum_{j \in S} p_j x_j \geq \sum_{\substack{j_1, j_2 \in S \\ j_1 \neq j_2}} p_{j_1} p_{j_2} \tag{8.13}
$$

as a new constraint. If this is performed for any $S \subseteq J$ with $|S| \geq 2$, the extreme points of the formulation become the solutions forthcoming from a non-idle processor. There are however an exponential many subsets S of J. It is therefor wise to only add cutting planes for sets S that contain jobs that have overlapping processing time in the relaxed optimal solution. The relaxation can then be solved again to find new sets S to implement cutting plane (8.13).

Case study: 1 processor and 2 operations with release dates

Suppose the operations of jobs in J have release dates r_j. First consider the case that $n = 2$. It is desired to find a solution where operations start as soon as possible, but according to a given ordering. If either $r_1 + p_1 \leq r_2$ or $r_2 + p_2 \leq r_1$, the events are equal to their release dates and no cutting plane is required.
Now assume $r_1 + p_1 > r_2$ and $r_2 + p_2 > r_1$. The candidate extreme points are $(r_1, r_2 + p_2)$ and $(r_2 + p_2, r_1)$. Notice that they lie on a line given by the equation

$$
x_2 = -\frac{r_1 + p_1 - r_2}{r_2 + p_2 - r_1} x_1 + r_1 + p_1 + r_1 \frac{r_1 + p_1 - r_2}{r_2 + p_2 - r_1}
$$

which can be rewritten as inequality (8.14).

$$
(r_1 + p_1 - r_2)x_1 + (r_2 + p_2 - r_1)x_2 \geq p_1 p_2 + p_1 r_2 + p_2 r_1 \tag{8.14}
$$

A graphical interpretation of cutting plane (8.14) is shown in figure 8.7.

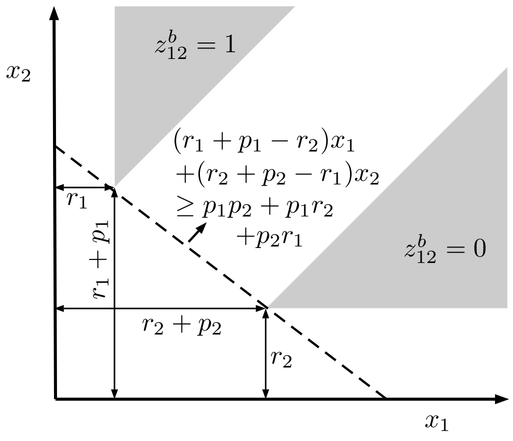


Figure 8.7: Cut on 2 operations with release dates

Case study: 1 processor and n operations with release dates

Now suppose there are n operations with release dates that are supposed to be scheduled on a single processor. Before looking at the cutting plane, example 8.3 shows whether or not finding a single cutting plane that is tight with the extreme points is possible or not, as it was the case in the absence of release dates.

Example 8.3. Let 3 operations to be scheduled on a single processor, with release dates $r_1 = 0, r_2 = 1$ and $r_3 = 2$. The processing times are $p_1 = p_2 = p_3 = 1$. The candidate extreme points for x are $(0, 1, 2), (0, 3, 2), (2, 1, 3), (3, 1, 2), (3, 4, 2)$ and $(4, 3, 2)$ by trying all possible orders of 3 operations. Now if there is a single cutting plane that is tight with the extreme points it must hold that $\exists a \in \mathbb{R}^3, b \in \mathbb{R}$ such that

$$a_1x_1 + a_2x_2 + a_3x_3 = b$$

for all 6 extreme points. Values a and b can be found by solving

$$\begin{bmatrix} 0 & 1 & 2 \\ 0 & 3 & 2 \\ 2 & 1 & 3 \\ 3 & 1 & 2 \\ 3 & 4 & 2 \\ 4 & 3 & 2 \end{bmatrix} a = \begin{bmatrix} b \\ b \\ b \\ b \\ b \\ b \end{bmatrix} \sim \begin{bmatrix} 0 & 1 & 2 \\ 0 & 2 & 0 \\ 2 & 0 & 1 \\ 3 & 0 & 0 \\ 3 & 3 & 0 \\ 4 & 2 & 0 \end{bmatrix} a = \begin{bmatrix} b \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \sim \begin{bmatrix} 0 & 0 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} a = \begin{bmatrix} b \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The only solution is $a_1 = a_2 = a_3 = b = 0$. This means that unlike in the case without release dates there is no cutting plane that is tight with all the extreme points.

Finding good cutting planes is unfortunately not as easy as in the case without release dates.

Theorem 8.4. *Suppose there are n operations on a processor with $r_{j_1} < r_{j_2} + p_{j_2}$ and $r_{j_1} \geq 0$ for any two operations j_1 and j_2. Let $r = \min\{r_1, \dots, r_n\}$. Then*

$$\sum_{j=1}^{n} p_j x_j \geq \sum_{j_1=1}^{n} \sum_{\substack{j_2=1 \\ j_2 \neq j_1}}^{n} p_{j_1} p_{j_2} + r \sum_{j=1}^{n} p_j \tag{8.15}$$

is valid for any feasible schedule.[13]

[13]Theorem 8.4 was written and proved by the writer.

Proof. The proof follows the same steps as the proof of Theorem 8.2. Since $r_{j_1} < r_{j_2} + p_{j_2}$ for any two operations j_1 and j_2, there is no idle time in a schedule where all operations start as soon as possible. Hence, add r_{j_1} to every x_j in the steps of (8.11). Then by applying $r_{j_1} \geq r$ the result follows from the same steps.

□

8.2 Cutting on binary parameterizations

This section will focus on cutting planes for the binary variables of the MILP from the MPC-SMPL problem. The literature already suggests FDI's for the ordering variables (Bolotashvili et al [9] and Doignon & Rexheb [20]). So this section will focus on developping cutting planes for the routing variables. Recall that $\mathbb{1}^S$ is the incidence vector of a set $S \subset \mathbb{N}$ and $\mathbb{2}_N(S)$ is defined as in section 5 page 36.

Suppose a set of $u \in \mathbb{N}$ binary variables is parameterized as in the first or second method for routing variables (R1 and R2) in section 5.1. Let $X_u \subseteq \mathbb{B}^N$ denote the set of feasible reparameterized binary vectors with $N = \lceil \log_2(u) \rceil$. By the parameterization, for $S \subseteq \underline{N}$ it holds that $\mathbb{1}^S \in X_u$ if and only if $\mathbb{2}_N(S) \leq u - 1$. Now let $P_w = \{w \in \mathbb{R}^N \ : \ 0 \leq w_i \leq 1 \ \forall i \in \underline{N}\}$. In order to ensure P_w is a formulation for X_u inequality (5.4) page 37 suggested by Van den Boom et al. [50] can be added, unless u is an exact power of 2. A graphical drawing of (5.4) for $u = 5$ is shown in figure 8.8.

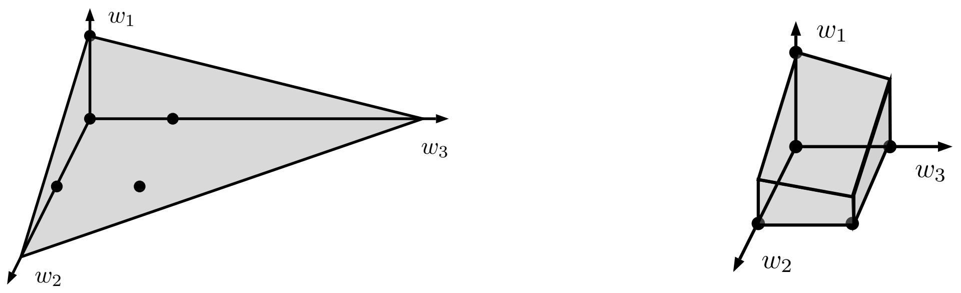


Figure 8.8: The cut $4w_1 + 2w_2 + w_3 \leq 4$ for $u = 5$. On the right the binary boundaries are added to form the new formulation P_w.

By graphical inspection it becomes clear that only adding $4w_1 + 2w_2 + w_3 \leq 4$ does not result in P_w being the convex hull of X_u. This can also be verified by stating that there are extreme points of the formulation that are not integer. One can verify that $(\frac{1}{3}, 1, 1)$, $(\frac{1}{2}, 1, 0)$ and $(\frac{3}{4}, 0, 1)$ are indeed extreme points but not integer.

Theorem 8.5. *Let $u \in \mathbb{N}$ be no exact power of 2, $N = \lceil \log_2(u) \rceil$ and $X_u = \{w \in \mathbb{B}^N \ : \ \exists U \subseteq \underline{N}$ such that $\mathbb{1}^U = w$ and $\mathbb{2}_N(U) \leq u - 1\}$. Now let $S \subseteq \underline{N}$ such that it indexes only a 0 in the binary notation of $u - 1$ for the highest value of S and it contains all indexes of 1s before the 0. It then holds that $(\mathbb{1}^S)^T w \leq |S| - 1$ induces a facet of* $\operatorname{conv}(X_u)$.[14]

Proof. Suppose S is given as in the Theorem, by Definition B.15 on page 113 it is to be shown that $\dim(\operatorname{conv}(X_u) \cap F) = \dim(\operatorname{conv}(X_u)) - 1$ where $F = \{w : (\mathbb{1}^S)^T w = |S| - 1\}$. Now notice that by definition of X_u it follows that $\mathbf{e}_1, \ldots, \mathbf{e}_N$ and $\underline{0}$ are $N + 1$ affinely independent vectors in X_u. Since $X \subseteq \{0, 1\}^N$ its dimension cannot exceed N. It follows that $\dim(\operatorname{conv}(X_u)) = N$.

[14]Theorem 8.5 was written and proved by the writer.

Now define

$$v_i = \begin{cases} \mathbb{1}^S - \mathbf{e}_1 + \mathbf{e}_i \text{ if } i \in \underline{N}\backslash S \\ \mathbb{1}^S - \mathbf{e}_i \text{ if } i \in S \end{cases}$$

It is certain that $1 \in S$, because otherwise 1 indexes a 0 in the binary notation of $u-1$, which means $|S| < 2$. First assume that $i \in \underline{N}\backslash S$. Now consider $w \in \mathbb{B}^N$ with $w_1 = 0$. It holds directly that $w \in X_u$, because otherwise

$$\begin{aligned} u - 1 < \mathbb{2}_N(\{j : w_j = 1\}) &\leq 2^{N-2} + 2^{N-3} + \ldots + 2 + 1 \\ = 2^{N-1} - 1 = 2^{\lceil \log_2(u) \rceil - 1} - 1 &\leq 2^{\log_2(u)} - 1 = u - 1 \end{aligned}$$

which is contradictory. This means that it holds that $v_i \in X_u \subseteq \text{conv}(X_u)$. Now note that

$$(\mathbb{1}^S)^T v_i = (\mathbb{1}^S)^T (\mathbb{1}^S - \mathbf{e}_1 + \mathbf{e}_i) = |S| - 1$$

so $v_i \in F$. Now assume $i \in S$ and let $i^\star := \max_{i \in S} i$. Then

$$\mathbb{2}_N(\{j : [v_i]_j = 1\}) \leq \mathbb{2}_N(S \backslash \{i^\star\}) \leq u - 1$$

which shows $v_i \in X_u \subseteq \text{conv}(X_u)$. Now it also holds that

$$(\mathbb{1}^S)^T v_i = (\mathbb{1}^S)^T (\mathbb{1}^S - \mathbf{e}_i) = |S| - 1$$

and it also holds that $v_i \in F$ in this case. It is now shown that $v_i \in (\text{conv}(X_u) \cap F)$ for all $i \in \underline{N}$ and it is only left to prove that all v_i are affinely independent such that $\dim(\text{conv}(X_u) \cap F) = N - 1 = \dim(\text{conv}(X_u)) - 1$ follows. Now consider the vectors $N-1$ vectors $v_i - v_1$ where $i \neq 1$. If $i \in \underline{N}\backslash S$ then

$$v_i - v_1 = \mathbb{1}^S - \mathbf{e}_1 + \mathbf{e}_i - \mathbb{1}^S + \mathbf{e}_1 = \mathbf{e}_i$$

while if $i \in S$ then

$$v_i - v_1 = \mathbb{1}^S - \mathbf{e}_1 - \mathbb{1}^S + \mathbf{e}_1 = \mathbf{e}_1 - \mathbf{e}_i$$

This means that every vector $v_i - v_1$ is the only one among others that has a non-zero entry on the i-th index. It follows that $v_i - v_1$ are linearly independent for all $i \in \underline{N}\backslash\{1\} \Rightarrow v_1, \ldots, v_N$ are affinely independent and the proof is complete.

□

With Theorem 8.5 one can easily find some facets of $\text{conv}(X_u)$ by a few steps. First translate $u-1$ to its binary notation. Then for every 0 in this notation, let S consist its index and the indices of all ones before the 0. Then add the cutting plane $(\mathbb{1}^S)^T w \leq |S| - 1$ to your formulation. So for $u = 5$ the binary notation of 4 is 100, this means that the constraints $w_1 + w_2 \leq 1$ and $w_1 + w_3 \leq 1$ are added. The graphical result is shown in figure 8.9.

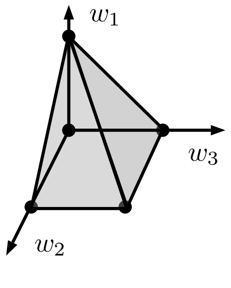


Figure 8.9: The constraints $w_1 + w_2 \leq 1$ and $w_1 + w_3 \leq 1$ make the convex hull for P_w.

Notice that the earlier found extreme points $(\frac{1}{3}, 1, 1)$, $(\frac{1}{2}, 1, 0)$ and $(\frac{3}{4}, 0, 1)$ are now separated from the formulation. The cut $4w_1 + 2w_2 + w_3 \leq 4$ is now redundant and can be removed.

Part IV
Simulation & Results

Parts II and III have shown how to reduce the complexity of the MPC-SMPL problem. To see if reparameterization and the cutting plane method actually reduce the complexity of the MPC-SMPL problem the first section of Part IV will contain the results of their implementation and simulation. Afterwards, the 3 main heuristic algorithms are compared. Part IV also includes some final words and recommendations to conclude this thesis.

Data Specification & Instance Generation

The results included in this thesis come from MATLAB simulations only. The MPC-SMPL problem specified in section 3.3 contains a lot of parameters to specify. For any simulation it is stated what the parameters of the MPC-SMPL problem instance is, as well as the algorithm parameters in case of presence. If not specified, the model and the algorithm parameters are specified in Appendix E.1. This interlude makes a few notes on the instances generated for simulation before continuing to the results.

The first two values that come to mind that define the instance of the MPC-SMPL problem are n and m. They respectively denote the number of jobs and processors. In any application, it should be clear what these values are. For this thesis, n and m are selected different values for different simulations. Since n and m define the computational complexity on the MPC-SMPL problem, they will highly effect the results. All results in thesis will therefore always specify what values of n and m are used.
Two other values that define the computational complexity of the MPC-SMPL problem are N_p and $\mu_{\max}$. Just like n and m, the higher these values are, the more time consuming the problem becomes. Therefore, N_p and $\mu_{\max}$ will always be specified along the presence of results in this thesis.
The cycle time λ of the system should always be specified. The choice of λ should also be motivated for simulation. If λ is to large, the cycles don't depend on each other. If it is too small, it is probably impossible to schedule the events efficiently. The binary conversion parameters β should be chosen negative and large enough in absolute value. However, if $|\beta|$ is too large there may be computational issues with the optimization (Nazareth [41]). It is therefore very important to choose a suitable β. It must hold that $|\beta|$ is at least the maximum time between two events with undetermined order, but make $|\beta|$ not higher than necessary. If two events can interchange order, the difference in cycle is at most $\mu_{\max}$. Therefore, for simulations the value $\beta = -\lambda(\mu_{\max} + 1)$ is usually chosen.
The set of routes of a job $j \in J$ can be pretty much anything. In the use of an application the route sets L_j should follow directly from the model. In the general case L_j is randomly generated such that for every $l \in L_j$ there is a route that differs on at most 1 processor or in the order of 2 processors. The routes contain processors of R_j which is randomly generated with a fixed size in the general case.

For every operation, a processing time has to be defined. Processing times are usually *exponentially distributed* samples (definition C.2 Appendix C.1 or Rice [47]). If not stated otherwise, the mean processing times are exponentially distributed with mean 3 and a minimal value of 1.
The output $y(k)$ of the SMPL system can have multiple definitions. For most simulations in this thesis, $y(k)$ will be the vector containing *normalized completion times*. The normalized completion time of job j in cycle k is equal to $\mathcal{C}_j(k) - \lambda k$. It denotes the time for a job to be completed after it's cycle starts. Whenever it is the case that this does not defines the output of the system, the structure of $y(k)$ will be specified.

Any release dates, due dates or synchronization constraints are neglected in the general instances of the MPC-SMPL problem. It feels unnatural to add these characteristics without working on a real application. Generating random due dates would make it unfair to compare the simulation of different instances. Release dates and synchronization constraints could also sometimes completely change the system and sometimes not. Therefore, if not stated otherwise, release dates, due dates and synchronization constraints are absent in the generation of general MPC-SMPL instances.

Since due dates are mostly absent, the weight α of the objective function is normally selected to be zero. To make sure events are set to their earliest possible time, but do not change the optimal value, κ is a vector containing very small numbers (usually 10^{-4}). The standard objective function is to minimize the sum op normalized completion times, so if not stated otherwise, δ is a vector of all ones.

9 Reformulation Results of the MPC-SMPL Problem

This section will elaborate on the results of reformulations of the MPC-SMPL problem. The results of this section correspond to the reparameterizations of binary variables, partitioning of the constraint matrix and strengthening the MILP formulation with cutting planes.

9.1 Results on reparameterization

Most of the computational complexity comes from the ordering variables. Besides that they are binary variables, there are many possible orderings which makes finding the optimum ordering difficult. To find a suitable parameterization method, proposed are the original method and two reparameterizations of section 5.2, namely O1 and O2QM (sections 5.2 and 5.3). In addition, it is left a question if it is good to add the circuit elimination constraints to the MILP. Since this it not necessary for O2QM, 5 parameterization methods are tested: no parameterization with and without circuit elimination, O1 with and without circuit elimination and R2QM. For the simulations, small instances of the MPC-SMPL problem are used: $n = m = Np = 3$ and $\mu_{\max} = 1$. Since comparison focusses on ordering parameterizations all jobs are assigned a single uniform randomly generated route over M. The processing times are generated by the exponential distribution (Definition C.3 page C.3) with $\mathbb{E}[p_{j,i}(k)] = 3$ for all operations (j, i) and every k. The objective function that is to be minimized is the sum of completion times. There are 100 instances of the MPC-SMPL problem generated, parameterized and solved for each parameterization with the GUROBI optimizer. The computational results in the form of box-plots (Williamson [52]) are shown in figure 9.1.

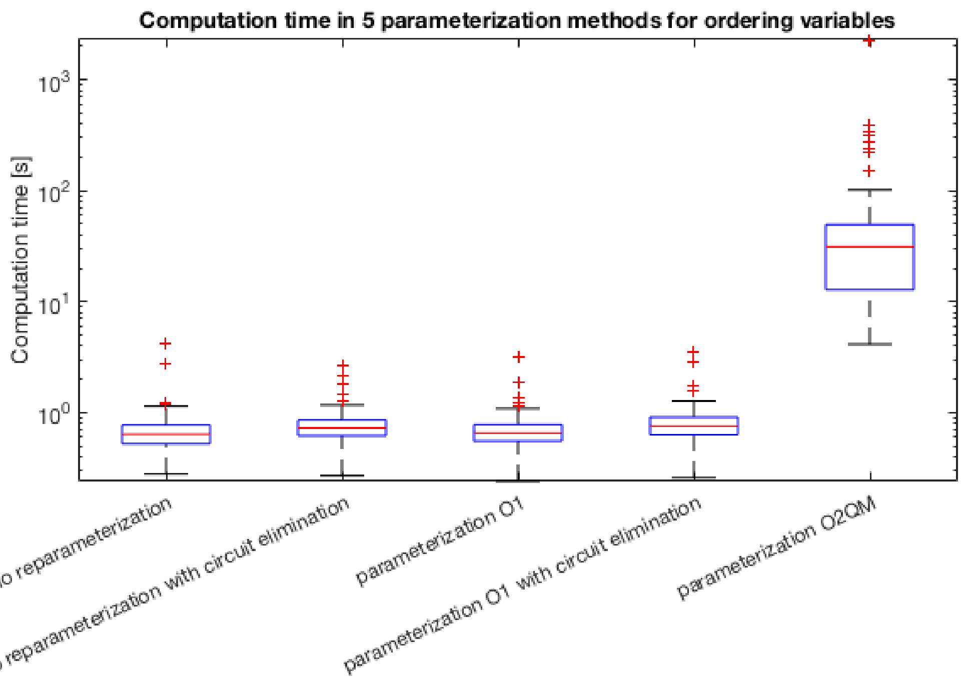


Figure 9.1: Computation time of 5 parameterizations for $n = 3, m = 3, N_p = 3$.

It stands out that the results of O2QM are terrible in comparison with the rest. As explained in section 5.4, the method O2QM has a lot more constraints than the other methods. This explains

the low running time. This explanation can be verified by watching the GUROBI output log, where O2QM explores much less nodes than other methods in the same amount of time. If a problem has more constraints, exploring a node takes more time so it makes sense that solving the O2QM parameterization explores less nodes in the branch-and-bound tree while consuming the same amount of time.
To do a better comparison on the other methods slightly larger instances are generated for simulation. This time, $n = 5$ and $m = 5$ are set for instance generation, and O2QM is left out. The results are shown in figure 9.2.

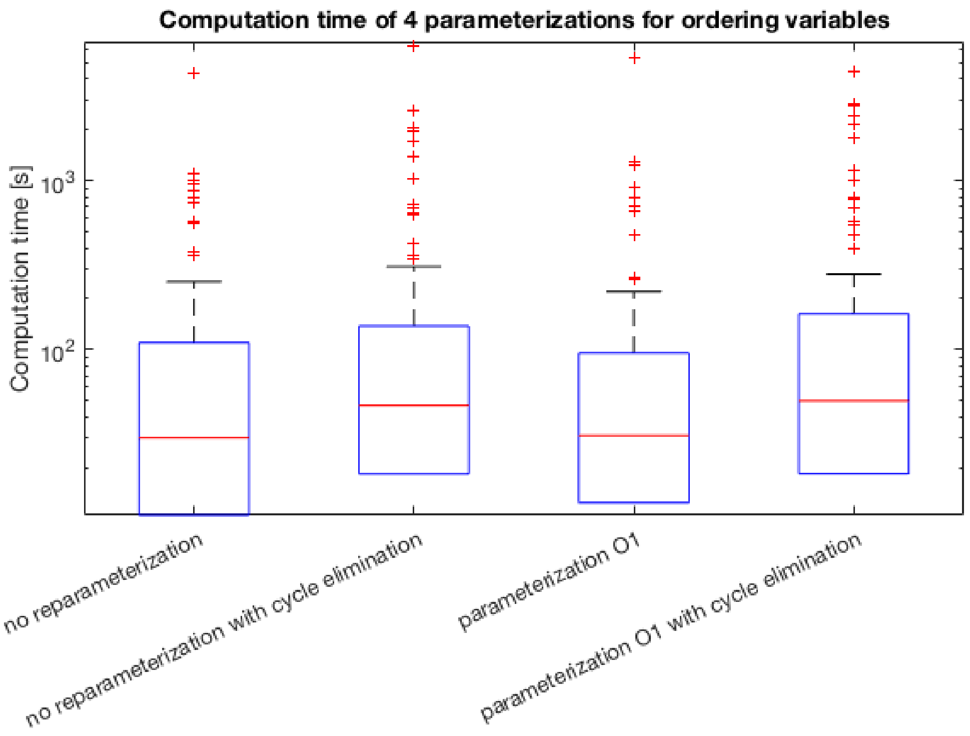


Figure 9.2: Computation time of 4 parameterizations for $n = 3, m = 3, N_p = 3$.

It seems that the parameterization method does not matter except that leaving out the circuit elimination constraints is beneficial. Of course, the O1 method uses only half of the ordering variables, but there is no noticable difference between O1 and the original parameterization. This can be explained by the fact that GUROBI uses a technique that is called *pre-solving* (Achterberg et al. [1]). With pre-solving, before evaluating a node in the branch-and-bound search tree, it is determined if some variables can already be solved regardless of the optimal solution. This means that whenever a variable z_{e_1,e_2} is set fixed by branching, the variable $z_{e_2,e_1} := \bar{z}_{e_1,e_2}$ is solved by pre-solving. GUROBI can do this very fast, and the time consumed is not noticeable. This explains why there is no noticaeble difference between no reparameterization and O1. It can therefore be concluded that both methods without circuit elimination work the best.

The routing variables of the MPC-SMPL problem seem less of a difficulty for the computational complexity, because most applications do not have an awful lot or routes. Before continuing to see how the routing variables can be parameterized the best, an example is given to show the weakness of the second parameterization for routing variables.

Example 9.1. Suppose there are $n = 3$ jobs, each have $|L_j| = 5$ routes. Then it follows that $N_1^r = n\lceil\log_2(|L_j|)\rceil = 9$ and $N_2^r = \lceil\log_2(|L_j^n|)\rceil = 7$ are the number of routing variables after the first and second reparameterization for routing variables. Originally, there are $\sum_j |L_j| = 15$ routing variables.

In the first method of parameterization (with or without the QM method), every routing variable can be replaced by a single max-plus product as in (5.3) and (5.22). As explained in section 5.4, this means that there are no resulting additional constraints in the MILP after reparameterization. However, in the second method of reparameterization of for ordering variables this is not the case. In the second reparameterization all the possible combinations of routes are put in an order which define the set if feasible solutions. The following array shows a possible ordering of the routing combinations, and the right column shows the combination number.

$$\begin{array}{ccccc}
l_1 & l_2 & l_3 & & \\
1 & 1 & 1 & & 1 \\
1 & 1 & 2 & & 2 \\
1 & 1 & 3 & & 3 \\
1 & 1 & 4 & & 4 \\
1 & 1 & 5 & & 5 \\
1 & 2 & 1 & & 6 \\
 & \vdots & & & \vdots \\
3 & 1 & 1 & & 51 \\
3 & 1 & 2 & & 52 \\
3 & 1 & 3 & & 53 \\
 & \vdots & & & \vdots \\
5 & 5 & 3 & & 123 \\
5 & 5 & 4 & & 124 \\
5 & 5 & 5 & & 125
\end{array}$$

Using this ordering, the QM method (section 5.3 or Quine [45], McClusky [40]) is applied in MATLAB to find minimal parameterizations for each of the 15 original routing variables. The results are 15 max-plus sum of product statements. The number of resulting terms of the sum of products per route is shown in table 9.1.

		l_j				
		1	2	3	4	5
	1	3	5	6	4	3
j	2	12	13	13	12	11
	3	25	25	25	25	25

Table 9.1: Number of terms in sum of products resulting from R2QM.

Since there are multiple resulting terms, there will be some extra constraints in the new MILP. Let $|l_j|$ be the length of route l of job j and let $s_{j,l}$ be the entry in table 9.1 in index (j, l). The number of additional constraints in the MILP is then equal to

$$\sum_{j=1}^{3}\sum_{l=1}^{5}(s_{j,l} - 1)(|l_j| - 1)$$

This means that if every route has length 3, there are 384 additional constraints resulting from R2QM. In comparison with R1 and R1QM, this is a trade of 2 variables for 384 constraints, which seems like a bad trade in terms of computational complexity.

Example 9.1 illustrates that already for small problems the R2QM method results in a lot of additional constraints. The number of constraints that have to be added grows only exponential,

so for larger instances this becomes even worse. As shown in Proposition 5.2 on page 38, decrease in number of binary binary variables is bounded by $\mathcal{O}(n)$ for R2 and R2QM in comparison to R1 and R1QM. Since this is a very low bound, the trade off between number of variables and constraints seems always to favour the first method of reparameterization over the second method. It is therefore concluded that the methods R2 and R2QM are never a good choice for reparameterization.
For the routing parameterization it remains to establish if the first or third method provides a better formulation. Proposition 5.3 on page 41 shows that it is certain that R1QM result in less binary variables than R3. This is however not always true for the number of constraints. Let j be a fixed job in the fixed processors case, so every $l \in L_j$ contains every processor of R_j. In R1, the number of constraints is equal to

$$(|R_j| - 1)|L_j|$$

In R3, constraints (5.11) on page 40 can be left out since they will never separate the optimal solution to the MILP. This can be seen that if one inequality of (5.11) does not hold, some events will wait for each other forever, and this cannot be optimal. The optimal solution would therefore not change if these constraints are left out. If so, the number of constraints resulting from R3 is equal to

$$2|R_j| + |R_j|! - |L_j| + |R_j|(|R_j| - 1) = |R_j|^2 + |R_j| + |R_j|! - |L_j|$$

where the constraint (5.10) is neglected since it is always a single constraint for any job j (and thus the number of constraints does not grow with $|R_j|$). The value of $|L_j|$ where the number of constraints is equal can now be computed.

$$\begin{aligned}(|R_j| - 1)|L_j| &= |R_j|^2 + |R_j| + |R_j|! - |L_j| \\ \Rightarrow |R_j||L_j| &= |R_j|^2 + |R_j| + |R_j|! \\ \Rightarrow |L_j| &= |R_j| + (|R_j| - 1)! + 1\end{aligned}$$

This means that if $|L_j| \leq |R_j| + (|R_j| - 1)! + 1$ it is certain that it is better to use R1 instead of R3, because in that case R1 results in less constraints as well as variables. In the case that $|L_j| > |R_j| + (|R_j| - 1)! + 1$ the R1 method still has less variables than R3, so it would be left a study to determine which method is better. Of course, the result depends on the structure of L_j and R_j, so for the general MPC-SMPL problem this cannot be answered.

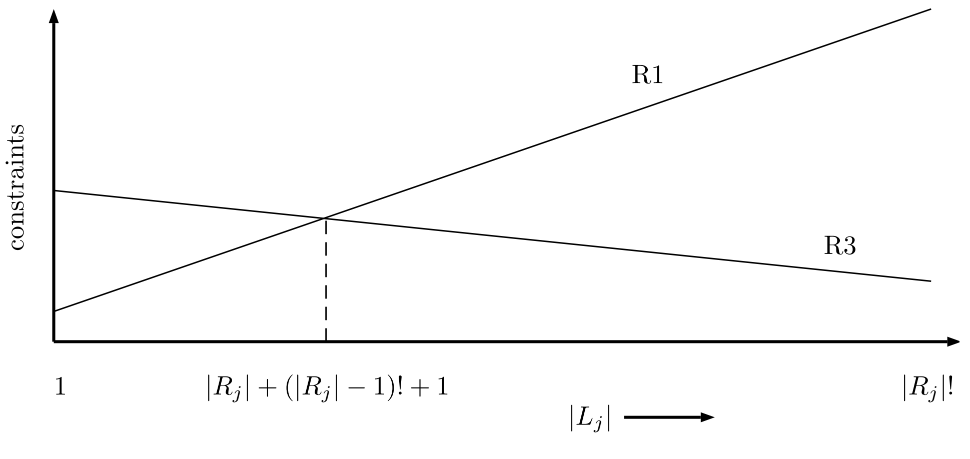


Figure 9.3: The number of constraints in R1 and R3

The reader might now be wondering why R1 is discussed instead of R1QM, since there is no reason not to apply Quine-McClusky on R1. The reason is that the research about the general MPC-SMPL problem, and when applying R1QM the number of constraints highly depend on the structure of the specific instance of the problem. However, the results found with R1 provide an upper bound for the number of constraints in R1QM. It can therefore still be concluded that when $|L_j| \leq |R_j| + (|R_j| - 1)! + 1$, method R1QM will outperform R3. In the case that $|L_j| > |R_j| + (|R_j| - 1)! + 1$, it should be a study on the instance which method is better.
It is left to discuss if it is a viable option to use R3QM. However, this method is similar to O2QM, and it is shown in figure 9.1 that this has a terrible performance. Therefore, no further simulations are tested on this method.

9.2 Results on partitioning

To make the partition-based optimization work well, the MPC-SMPL problem is desired to be partitioned optimally. The partitioning is formed as explained in section 6.1. Take for example an instance of the MPC-SMPL problem with 6 processors, partitioned into 3 segments. The structure of the constraint matrix and its partition are shown in figure 9.4. The non-zero entries of the matrix are indicated in blue.

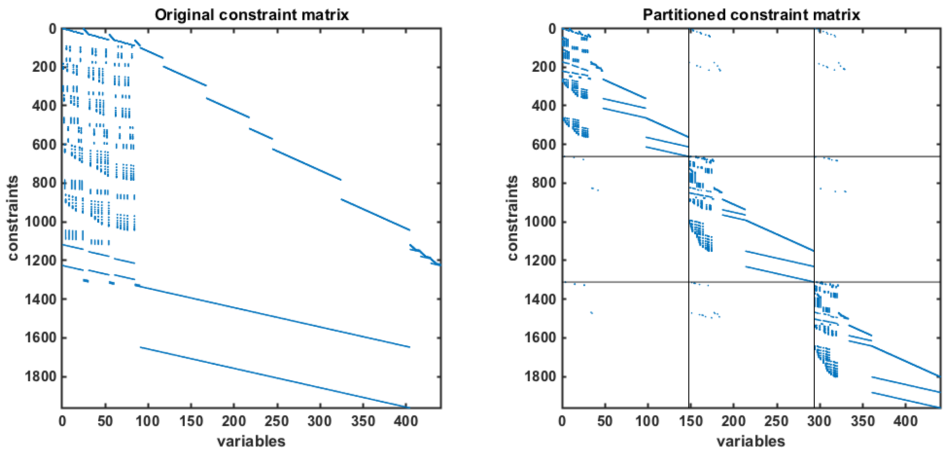


Figure 9.4: Structure of the constraint matrix before and after partitioning.

As you can see in figure 9.4 most variables are put in the block-diagonal structure of the matrix. The exact model parameters are shown in table E.1 in Appendix E.1. There are only a few which are outside the block-diagonal.

An important choice for partitioning is to determine a number of segments p for local optimization. First notice that it is important that all the blocks are of a similar size, for computational benefits. If the number of operations per processor is similar among processors, it is wise to choose p such that it divides m. The number of options then of course depends on m. So in case of few options, for example if $m = 7$, it is wise to also consider p to be a non-divider of m. To make a fair comparison, the rest of this chapter will only consider p values as dividers of m, where m is chosen such that it has more than 1 divider besides 1.
By the exponential nature of the running time of the GUROBI optimizer, choosing a larger p would result a shorter computation time. It is however likely that this will cost objective value. It is therefore essential to find a good value p that results in a good balance between speed and objective value. In applications, this of course depends on the demand for speed and quality. In

figure 9.5, a few instances of the MPC-SMPL problem are partitioned into different numbers of segments, and the number of variables outside the block diagonal constraint matrix is measured, denoted Q. The number of jobs and processors vary from 2 to 10 and are always even. The result is compared for $|L_j|$ is equal to 1, 3 and 5 for all $j \in J$. The partitioning parameters can be found in table E.2 in Appendix E.1.

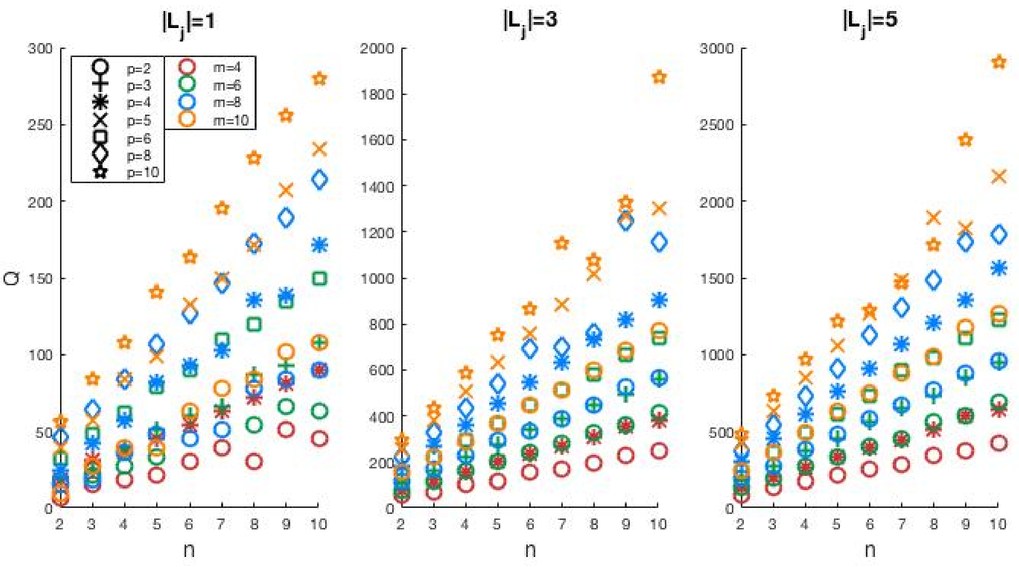


Figure 9.5: The number of variables outside the block diagonal structure Q as a result of optimal partitioning, for different n, m, $|L_j|$ and number of segments p.

Figure 9.5 can be used to determine the trade of between speed and objective value when choosing a suitable p. For example, suppose a system has 10 processors, and every job has 3 routes. If there re 10 jobs, one can see a large difference in Q between partitioning into 5 or 10 segments. This means that there is likely a large difference in objective value when partitioning into 10 segments in stead of 5. Now if there are 9 jobs, the difference in Q is very small. This means it is a good idea to partition into 10 segments instead of 5, since the cost of objective value is relatively low for the increase of speed.

9.3 Strength of the dual bound

When aiming to solve the the MPC-SMPL problem with the branch-and-bound approach, it is very important to find a good formulation (see Appendix B.2 for definition) that represents the MILP. Section 8 explains how the formulation for the MPC-SMPL problem can be improved with the use of cutting planes. The idea is that cutting planes close the gap between the lower bound from the relaxation and the optimal value.
Adding every cutting plane proposed in section 8 for every set of events is computationally too expensive because there is an exponential amount of subsets of events. For this a cutting plane algorithm is implemented in MATLAB. It only adds cutting planes for events that overlap in the relaxed solution. Afterwards, the relaxation is solved again and the process is repeated.
To see the effectiveness of the cutting plane algorithm, a number of MPC-SMPL problems have been simulated. The gap to the optimal value is computed for the solution to the relaxation of the original formulation, the GUROBI improved formulation and the formulation resulting from the cutting plane algorithm. A higher gap means the formulation is closer to the convex hull of

the feasible region, which is desired. Ideally the gap is as close to 1 as possible. The results for different n and m are shown in figure 9.6.

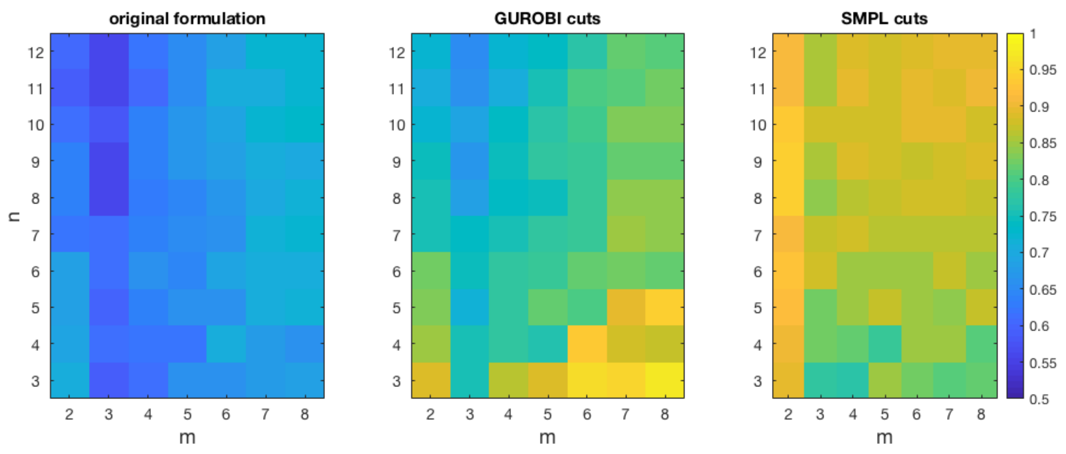


Figure 9.6: Gap between the relaxed optimal and global optimal for 3 formulations and different sized instances.

It can be seen that GUROBI pushes the dual bound to a significant higher value for some instances, but a lot of times the gap remains very low. The SMPL cuts of section 8 do an overall better job at strengthening the lower bound for most problem sizes.

10 Computational Results

The MPC-SMPL problem is a generalized framework for controlling many type SMPL systems. To make the study on algorithms on this problem a bit narrowed down, this section will focus on the results on simulation where the objective function is the sum of normalized outputs. So the output of the system is a vector containing the times when jobs are completed (starting from the beginning of their cycle), and $\delta_j = 1$ for all $j \in J$ is taken for the objective function.
In this section, the 3 heuristics designed in this thesis are compared in their computational speed and objective quality. They are simulated on general instances of the MPC-SMPL problem and on the applications described in section 4.

10.1 Algorithm comparison on general SMPL systems

Before tuning the parameters of the heuristic algorithms, it is desired to know the impact of the starting points on the heuristics. The partition-based optimization, tabu search and simulated annealing all require a feasible starting point, which impacts the outcome of the algorithm. All three methods can also do a multi-start algorithm, where they are performed from multiple starting points. Generating initial solutions from a greedy heuristic takes considerably less time than the actual performance of the main heuristic algorithms. It might therefore be worth considering looking for a better starting point before continuing to the actual optimization.
To find out how a good starting point can be selected, a first look is taken upon the correlation between the quality of a starting point and the quality of the solution after the heuristic optimization. Consider a generated instance of the MPC-SMPL problem with $n = m = 8$, $\lambda = 40$, $5 \leq |R_j| \leq 7$. The number of routes is considered in the cases $|L_j| = 1, 3, 5$.
Starting points are generated using greedy heuristics from section 7.1. Starting points with a good objective value do not in general always result in better results of heuristic solutions. Therefore, a test is performed in MATLAB to find out if there is any relation between the quality of a starting point and the quality of the heuristic outcome. Let 1000 starting points be generated by the generalized random greedy heuristic, but with a dispatching rule that selects an available operation by the uniform distribution (Rice [47]). The starting points are used for all 3 heuristic algorithms and the results are ordered by the resulting objective value. Tabu search and simulated annealing keep improving until no better solution has been found for 30 iterations. The size of the tabu list is 40, the initial temperature T is 10, and the temperature decreases with ratio 0.9. In figures E.1, E.2 and E.3 of Appendix E on page 134 the results are compared with their initial starting point. To normalize the data, the mean is subtracted from as well the starting points as heuristic objective value. Table 10.1 shows the estimated correlation coefficient (defined in Appendix C.1 page 124) of the results. The exact model and heuristic parameters are specified in table E.4 in Appendix E.1.

	$\lvert L_j\rvert = 1$	$\lvert L_j\rvert = 3$	$\lvert L_j\rvert = 5$
Partition-based optimization	0.4471	0.6415	0.8723
Tabu search	0.3063	0.4141	0.1126
Simulated annealing	0.5228	0.5732	0.7498

Table 10.1: Correlation Coefficient of normalized objective values between starting points and algorithm result.

From table 10.1 it is clear that a good starting point does not always have a high probability in resulting in a good solution from a heuristic algorithm. With tabu search, the starting points are

barely correlated, which means that it is probably best to begin the tabu search algorithm right away from the first starting point. With partition-based optimization and simulated annealing the quality of the starting solution seems to matter more, especially when there are multiple routes. For partition-based optimization and 5 routes per job the result of the optimization is even highly correlated. In this case it is highly advised to find a good starting solution first before doing the partition-based optimization.
Now to do the comparison between the heuristic algorithms, another 100 MPC-SMPL problems are generated. The model parameters are $n = m = 8$, $\lambda = 40$ and $6 \leq |R_j| \leq 7$ and $|L_j| = 1$. The objective value is the sum of completion times. Now each problem is solved by a multi-start version of each of the three algorithms until the 90 seconds have passed. The current objective value of all the 100 instances are compared over time. Figure 10.1 shows the number of times each algorithm provides the best objective value for each given time between 0 and 90 seconds.

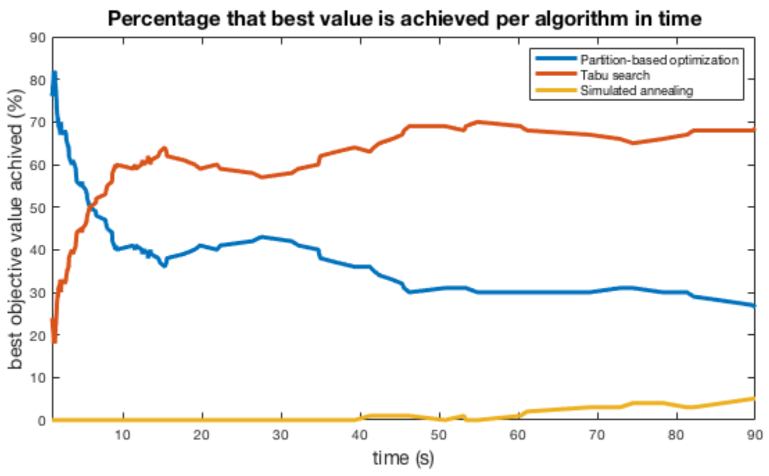


Figure 10.1: (Main result) The percentage of time an algorithm was the best.

From figure 10.1 it can be seen that tabu search gives most times the best solution if the computation time is higher than 5 seconds. If the computation time is lower than 5 seconds the partition-based optimization is most of the time coming up with the best solutions. Simulated annealing is however barely competing with the other two algorithms. It comes out as the best algorithm a few times, but never when the computation time is under 40 seconds.
Figure 10.1 does not provide any information about the difference in objective value between the algorithms. Therefore, figure 10.2 shows the moving box plots (Williamson [52]) of the gap between the current objective value and the eventually best found solution for each algorithm.

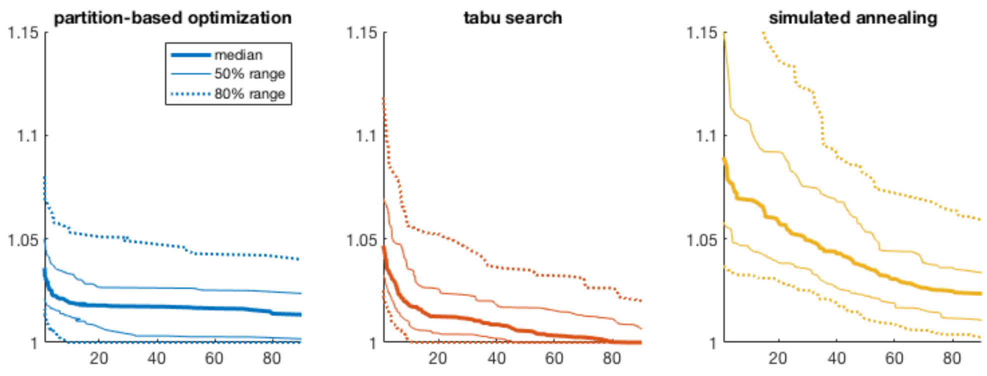


Figure 10.2: Moving box plots of the primal gap for each heuristic.

Figure 10.2 illustrates the speed of the partition-based optimization. It is clear that it has difficulties to improve after some time. The other algorithms do improve the solution later in time. From figure 10.2 it is also clearly seen that tabu search works best on the general MPC-SMPL problem if the computation time is not too small.

10.2 Results on applications

The railway network

As was shown by Kersbergen [34], the railway network modeled in Section 4.1 can be nicely partitioned because every processor corresponds to a railway track. The railway model in this thesis is a more detailed model but only a part of the Dutch railway network. The main difference is that in this thesis trains cannot overtake everywhere and they are able to arrive and depart at alternative platforms sometimes.

A few partitions are put to the test to find which one is suitable for optimization. The railway network is split into 2, 3, 4, 5 and 6 segments. The partitioned constraint matrices are shown in figures D.3 and D.4 in Appendix D.2. The models are created with $N_p = 4$ (The prediction horizon is thus 2 hours) and $\mu_{\max} = 1$.
Now let MATLAB generate a lot of random disturbances. They are generated by increasing some of the processing times with 0 to 20 minutes. This causes a lot of trains to be stuck behind other trains if no control is applied. For 100 of these disturbed situations the problem is solved by all the partitioned systems with the partition-based optimization algorithm from section 6.2 page 54. For the starting point the delayed solution resulting from the regular routing and ordering is used. The resulting normalized objective value and computation times are shown in figure 10.3. The normalized objective value is the total tardiness (or delay) minus the mean resulting total tardiness for that situation.

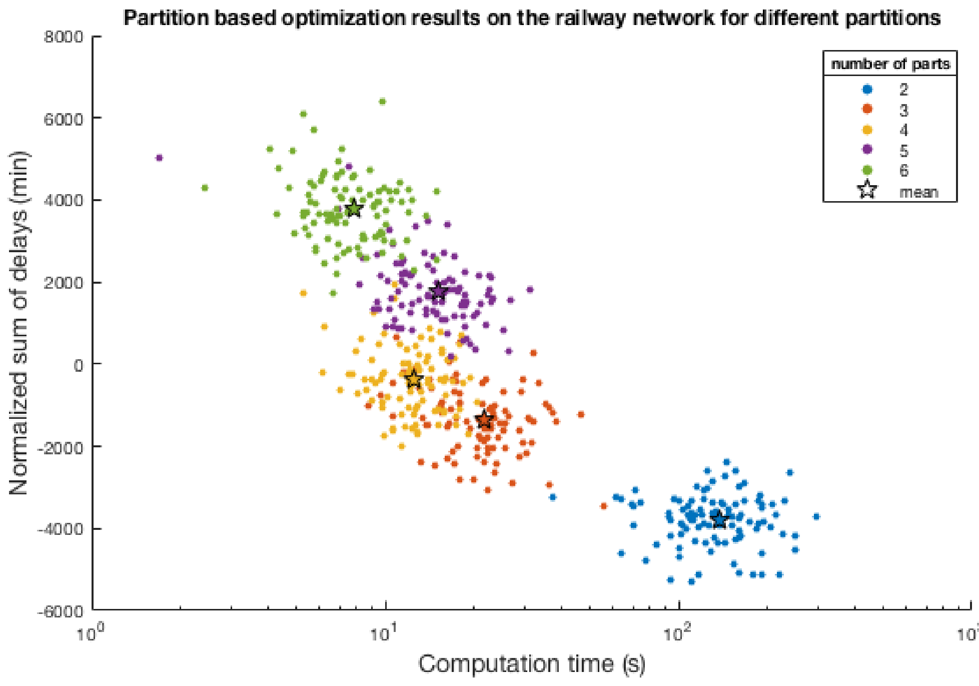


Figure 10.3: Normalized objective value and computation time of 100 disturbed railway situations, computed with different sized partitions.

From figure 10.3 it can clearly be seen that it is not a good idea to partition the railway model into 5 segments, since partitioning into 4 segments results better objective value and computation time on average. The other partitions have either a better average computation time or better average objective value in comparison with the othe partitions. A realistic computation time for the railway network would be somewhere in between 5 and 60 seconds. Extending the computation time beyond 60 seconds is not a good idea, since a lot can change on the railway network in a minute. Almost every minute a train arrives or departs somewhere on the network for example. Since partitioning into 6 segments results in very low reduction of the delays, recommended is to choose to partition the network into 3 or 4 segments depending on the desired computation time.

For the railway network a few alterations are made to the tabu search and simulated annealing algorithms. It does not make sense to swap the order of 2 trains on a track and leave the order on the rest of the network unchanged. This would mean that one train surpasses an other while being surpassed by the other train at the next track. The neighbour solutions are therefore changed to change the order of 2 trains on a track and every track following where these trains both go.
The results of the railway network optimization by heuristics is shown in figure 10.4.

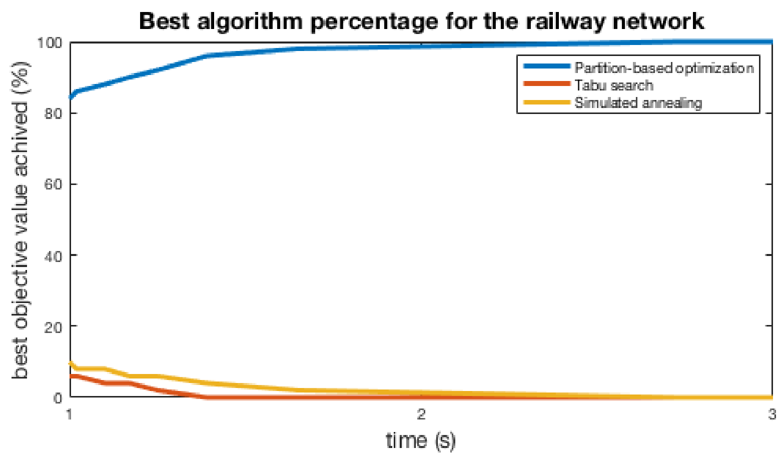


Figure 10.4: Best algorithm per time on the railway network.

It can clearly be seen from figure 10.4 that the partition-based optimization is the real winner on the railway network. In the first 2 seconds of the optimization the other algorithms do better than the partition-based optimization a few times, but after 3 seconds the partition-based optimization is the best algorithm every single time.

The container terminal

In the following results the system of the container terminal is partitioned into N_a segments. The parameters for simulation can be found in table E.7 on page 136. After partitioning, figure 10.5 shows the presence of variables in the partitioned constraint matrix.
The partition does not look well partitioned due to an amount of 816 appearances of variables outside of the block-diagonal structure. The next step is to run again a number of instances and compare the results of the 3 heuristics. The results are shown in figure 10.6.

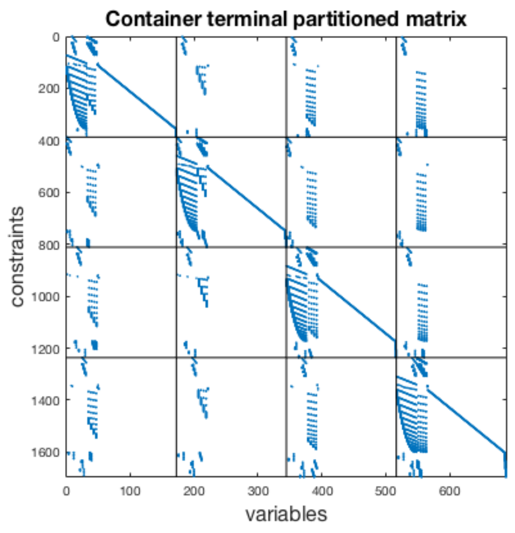


Figure 10.5: Partitioned constraint matrix for the container terminal.

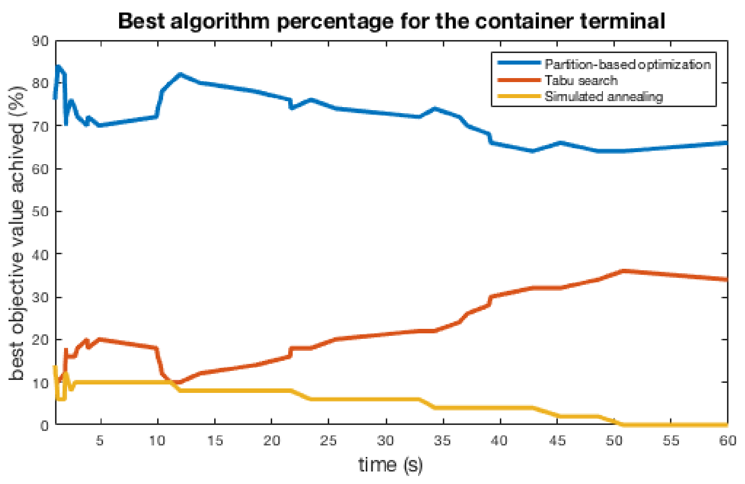


Figure 10.6: Best algorithm per time on the container terminal.

Despite the high number of variables outside the block-diagonal structure in figure 10.5, the partition-based optimization performed the best. It seems that the tabu search is not working as well as for general systems. The partition-based optimization seems to do a lot better. This could possibly be explained by one of the following reasons.

1. Switching routes in the container terminal is a bit different than for general systems, since every route is completely different. Normally, tabu search switches the route by switching the order of 2 processors for a job. In the container terminal this is not possible. Here, it chooses to swap the AGV's of two containers. This means that the number of containers each AGV handles does not change in the neighbourhood. This limits the amount of the possible solutions which tabu search and simulated annealing can reach to a severe extend.

2. Greedy solutions seem to work very well on the container terminal. By table 10.1 on page 135 and figures E.1-E.3 on page 134 it can be seen that the results of the partition-based optimization are highly correlated with the objective value of the starting points. With good starting points from greedy solutions, the partition-based optimization gets better results.

11 Final Words

The main goal of this thesis was to solve the MPC-SMPL problem in a practical way. The aim of the thesis focusses on reducing the computation time of the solver, such that MPC on SMPL systems can be implemented on-line. The complexity of the MPC-SMPL problem was attempted to be reduced via reparameterization, partitioning and cutting planes. To solve the actual MPC-SMPL problem 3 heuristics were designed and implemented. This section will give a recap of what the contributions of this thesis are, followed by the main conclusions. Afterwards some recommendations are made.

11.1 Contributions

The current state of the art of the MPC-SMPL problem as described in the introduction was that many ideas have been suggested to solve the MPC-SMPL problem but only a few have been fully worked out. SMPL systems are similar to scheduling problems as they are described in Pinedo [44], but not many of the scheduling literature has been used for SMPL systems. This thesis contributes in classifying the MPC-SMPL problem as a scheduling problem as described in section 2.3 such that the literature of scheduling can be useful to solve the MPC-SMPL problem.

The next contribution of this thesis is the detailed evaluation of reparameterizations suggested by van den Boom et al. [50]. Some other parameterizations are suggested in this thesis as well. Propositions 5.2 and 5.3 are composed and proved by the writer. They provide bounds on the differences in numbers of binary variables for the different parameterization methods for routing. This thesis also used the techniques of the Quine-McClusky method (section 5.3 or Quine [45], McClusky [40]) to reduce the number of binary variables and constraints of the MILP.

The formulation (Definition B.11 page 112) of the MILP resulting from the MPC-SMPL problem matters for the computational results. Therefore this thesis developed a number of cutting planes (Definition B.13 page 113) to improve the formulation. Since Doignon & Rexhep [20] and Bolotashvili et al. [9] already provide solid cutting planes on ordering variables, this thesis focused mainly on developing cuts for routing variables (section 8.2) and mixed-integer cuts (section 8.1). Van den Boom et al. [50] suggested a single valid cutting plane for the routing variables. However, it has been shown in section 8.2 that this formulation of routing variables can be improved. Written and proved by the writer, Theorem 8.5 on page 77 even shows that the given cuts of that section are facet defining. Proposition 8.1 on page 72 is written en proved by the writer and compares the strength of the corresponding cuts. Theorems 8.2 and 8.4 are also written and proved by the writer and validate the described cutting planes. Theorem 8.2 can sometimes even provides the strength of the suggested cutting plane. In order to prevent adding too much cutting planes the writer wrote and proved Theorem 5.4 on page 42.

Lastly, the work of this thesis contributes in the development of heuristics for the MPC-SMPL problem. Partition based optimization was already designed and implemented by Kersbergen [34]. This thesis improved the work of the algorithm by forcing the partition to be well partitioned (Definition 6.3 page 53) and by designing a multi-start version of the algorithm. Starting points are either generated as suggested in Pinedo [44], Chiang & Fu [14], Rajendran & Holthaus [46], Kaban et al. [32] and Dominic et al. [21] or by the (random) generalized greedy method (page 62) designed by the writer.

The other heuristics designed are tabu search and simulated annealing. The design of their neighbourhood structure was inspired by Liaw [38], but the writer made some changes to make it work for MPC-SMPL instead of open shop scheduling. Liaw [38] describes a critical path on which neighbours are constructed, but since the MPC-SMPL cost function is of a more general form the writer chose to do the same thing on a critical forest (Definition 7.3 page 65). Since the MPC-SMPL problem is not exactly an open shop problem, some adjustments in constructing neighbours had to be made in order to maintain feasibility.

11.2 Conclusions

The SMPL system model opens up a lot of possibilities to model discrete events systems. The ability to switch between modes allows the user to model routing and ordering of operations. In addition, there are possibilities to specify synchronization, timetable references, tardiness, deadlines and other specified constraints. The SMPL model has many applications of which the railway network and container terminal were discussed in this thesis.
To control SMPL systems one can use model predictive control. The objective function for MPC is free to design, depending on the desired behaviour of the system. To implement an MPC controller one must solve the MPC-SMPL problem on-line. The MPC-SMPL problem is $\mathcal{NP}$-hard which means there is an unreasonable computation time required to solve the problem for fairly large instances. This makes it difficult to make MPC work in on-line applications. Even commercial solvers such as GUROBI cannot solve large instances practically well.

To make the MPC-SMPL problem better solvable an attempt has been made by reducing the number of binary variables, since they have a large contribution to the complexity of the problem. However, the factorial number of solutions (with respect to the problem size) always stays the same after reparameterization. Reparameterizing routing variables does not seem to have an effect. Since there are usually not many routing combinations this makes sense. The factorial number of solutions with respect to the problem size come from the ordering variables. The only good parameterization for ordering is O1, which is a standard parameterization for ordering. Parameterization O2QM results in such an unreasonably large number of extra contraints that it actually slows down the optimization.

Another attempt to reduce the complexity of the MPC-SMPL problem was through the cutting plane method. Multiple cutting planes have been devised to strengten the formulation of the problem. However, one cannot add every cutting plane because there are simply too many to compute. It is therefore best to only add cutting planes to the formulation that separate the relaxed optimal solution. This approach significantly improved the relaxation. However, the duality gap remains too large to solve the MPC-SMPL problem directly.

By this point it is clear that the MPC-SMPL problem must be solved heuristically. Model based partitioning is a method that splits the MPC-SMPL problem into smaller local problems. Each local problem can then be solved individually taking in account the constraints of the other local problems. This process is iterated until no improvement can be obtained, this is called partition-based optimization. It is effective at solving the MPC-SMPL problem very fast, but with a resulting local optimum.

The other heuristics designed in this thesis are tabu search and simulated annealing. Both are local search meta-heuristics. They define the same neighbourhood function in which they iteratively change the current solution to a suitable one. To prevent getting stuck in local minima, tabu search keeps track of a list of forbidden solutions. Simulated annealing chooses its neighbours randomly in order to escape local minima.

It seems that for the general MPC-SMPL problem tabu search is the best algorithm. Simulated annealing is almost never better than both the other heuristics. If the given computation time is less than 5 seconds, partition-based optimization performs best however. It takes some time for tabu search to find a good solution, it is a little slower in that sense. Because partition-based optimization is really fast it can try way more starting points than the other algorithms. If the MPC-SMPL increases further in size, the running time seems to affect tabu search and simulated annealing more than the partition-based optimization.

The results for applications are completely different from the results for general SMPL systems. In the railway network, the partition-based optimization completely dominates the other heuristics. The system seems to be too large for the other algorithms to get up to a good solution within a reasonable amount of time. In the container terminal the partition-based optimization is also most often the best algorithm, but tabu search and simulated annealing do beat the partition-based optimization a fair amount of times.

11.3 Recommendations

In this section a few final statements are made based on the results included in this report. They are short recommendations for further research and implementation of the MPC-SMPL solver.

Reparameterization

The main complexity from the MPC-SMPL problem does not come from the number of integer variables, but from the number of feasible solutions. This is factorial in the input size. Having much integer variables in a model should not always directly be a problem. A problem caused by having more integer variables than necessary is that the branch-and-bound algorithm does more node iterations. This will however result in more pruning by infeasibility, so the loss of time is not terrible. If one desires to prevent this loss in time, one can presolve the MILP in every node iteration. Presolving is explained by Achterberg et al. [1], and is used by every commercial solver. One can also use advanced pruning (Appendix B.5) to prevent skip the pruning of infeasibility. In short, there are plenty of alternatives that are better than reducing the number of integer variables.

Branch-and-bound approach

The main problem with the branch-and-bound approach is the strength of the dual bound. Commercial solvers often have poor dual bounds for instances of the MPC-SMPL problem during the optimization, resulting in too high computational time. This thesis has however shown some improvements on strengthening the dual bound for the MPC-SMPL problem. Pushing the dual bound even further would be a very interesting study for future research. The dual bound can also be improved per node in the branch-and-bound tree. This was not included in this thesis and could help the branch-and-bound solver prune more often. If the dual bound can be improved enough there might be a possibility to make the branch-and-bound approach usable for on-line MPC on SMPL systems. The additional cutting plane algorithm must be based on special properties of the MPC-SMPL problem, regular cutting planes are already included in commercial solvers.

Heuristic approach

The heuristic approach have shown to be the most effective approach of getting MPC working on SMPL systems in practice. Simulated annealing turned out to be the least favorable algorithm, but can maybe be improved by tuning the model parameters. Note that the main computations of the partition-based optimization come from GUROBI, which is programmed really well. One could try to implement tabu search and simulated annealing such that they compute most time-efficient. The results can then only improve for these heuristics.

Part V
Appendix

A An MPC-SMPL Example

The examples in this Appendix follow the modelling structures explained in section 2. The example of Appendix A.3 applies the control method of section 3.1 to the example of Appendix A.2.

A.1 The scenario and SMPL system model

Let the processors of this example be three machines that are able to operate on a series of jobs. Each cycle consists of only one job, so in this case cycle k stands for job k. Every job has to be processed first by machine 1, and is then processed by machine 2 or machine 3. So every job is processed at exactly 2 machines. Let $u(k)$ be the time that job k becomes available and let $y(k)$ be the time that job k is completed. Denote $p_i(k)$ as the time a job k spends on machine i. An illustrative overview of the machine setup is shown in figure A.1.

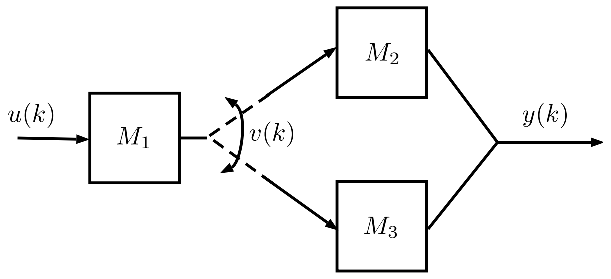


Figure A.1: Machine system setup of the example of section A.

The switching signal $v(k) \in \mathbb{B}_\epsilon$ determines whether job k is going to be processed on machine 2 or 3 after it is completed on machine 1. Notice that a single variable is used, and that this is possible because there are only two situations. If any more switching modes should be implemented, more binary switching variables should be used. Let $v(k) = 0$ if job k is processed by machine 2 and $v(k) = \epsilon$ otherwise. To model the system in SMPL form it is required to define the state variables. Let $x_i(k)$ be the starting time of the k-th job on machine i, unless job k is not being processed on machine i. If job k is not to be processed on machine i, $x_i(k)$ denotes the latest completion time of a job before k on machine i. This approach will make sure that the model in the form of equation (2.16) can be written with $\mu_{\max} = 1$. In other words,

$$x_1(k) \text{ is the starting time of job } k \text{ on machine 1}$$

$$x_2(k) \begin{cases} \text{is the starting time of job } k \text{ on machine 2 if } v(k) = 0 \\ \text{is the latest completion time of jobs } 0, \ldots, k-1 \text{ on machine 2 if } v(k) = \epsilon \end{cases}$$

$$x_3(k) \begin{cases} \text{is the starting time of job } k \text{ on machine 3 if } v(k) = \epsilon \\ \text{is the latest completion time of jobs } 0, \ldots, k-1 \text{ on machine 3 if } v(k) = 0 \end{cases}$$

Now by definition of $x_i(k)$ the switching is not only determined by the selected machine of job k, but also by the selected machine of the previous job. Also note that it is sufficient to let $x_i(k)$ depend on $x_i(k-1)$ instead of letting it also depend on $x_i(k-2), \ldots, x_i(0)$ because the latest completion time of jobs $0, \ldots, k-1$ is always larger or equal than the latest completion time of jobs $0, \ldots, k-u$, $u > 1$. There are 4 possible modes described on the top of the next page.

- job k and $k-1$ are scheduled both on processor 2.
- job k is scheduled on processor 2, job $k-1$ is scheduled on processor 3.
- job k is scheduled on processor 3, job $k-1$ is scheduled on processor 2.
- job k and $k-1$ are scheduled both on processor 3.

A switching mechanism should be defined for each of the 4 cases. Now define

$$\phi(k) = \begin{cases} 1 \ : \ \phi(k-1) \in \{1,2\}, \ v(k) = 0 \\ 2 \ : \ \phi(k-1) \in \{3,4\}, \ v(k) = 0 \\ 3 \ : \ \phi(k-1) \in \{1,2\}, \ v(k) = \epsilon \\ 4 \ : \ \phi(k-1) \in \{3,4\}, \ v(k) = \epsilon \end{cases}$$

Notice that if $\phi(k-1) \in \{1,2\}$ it holds that $v(k-1) = 0$ and job $k-1$ is processed by processor 2. Otherwise it can be concluded that job $k-1$ is processed on machine 3. It can now be derived that

$$\begin{aligned}
&x_1(k) \geq x_1(k-1) + p_1(k-1) && \\
&x_1(k) \geq u(k) && \\
&x_2(k) \geq x_1(k) + p_1(k) && \text{if } \phi(k) \in \{1,2\} \\
&x_2(k) \geq x_2(k-1) + p_2(k-1) && \text{if } \phi(k) \in \{1,3\} \\
&x_2(k) \geq x_2(k-1) && \text{if } \phi(k) \in \{2,4\} \\
&x_3(k) \geq x_1(k) + p_1(k) && \text{if } \phi(k) \in \{3,4\} \\
&x_3(k) \geq x_3(k-1) && \text{if } \phi(k) \in \{1,3\} \\
&x_3(k) \geq x_3(k-1) + p_3(k-1) && \text{if } \phi(k) \in \{2,4\}
\end{aligned}$$

One can now define the system matrices for each mode of operation. It follows that the system can now be modelled as follows.

$$\begin{aligned}
x(k) &= A_0(\phi(k),k) \otimes x(k) \oplus A_1(\phi(k),k) \otimes x(k-1) \oplus B \otimes u(k) \\
A_0(1,k) = A_0(2,k) &= \begin{bmatrix} \epsilon & \epsilon & \epsilon \\ p_1(k) & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon \end{bmatrix}, \ A_0(3,k) = A_0(4,k) = \begin{bmatrix} \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon \\ p_1(k) & \epsilon & \epsilon \end{bmatrix} \\
A_1(1,k) = A_1(3,k) &= \begin{bmatrix} p_1(k-1) & \epsilon & \epsilon \\ \epsilon & p_2(k-1) & \epsilon \\ \epsilon & \epsilon & 0 \end{bmatrix}, \\
A_1(2,k) = A_1(4,k) &= \begin{bmatrix} p_1(k-1) & \epsilon & \epsilon \\ \epsilon & 0 & \epsilon \\ \epsilon & \epsilon & p_3(k-1) \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \epsilon \\ \epsilon \end{bmatrix}
\end{aligned}$$

It is now clear that $\hat{\mu} = 0$ and $\mu_{\max} = 1$ in equation (2.16) for this example from the system matrices. Notice that the input matrix B is the same for each mode of operation. For determining the output $y(k)$ it depends whether a job is processed on machine 2 or 3. Let

$$C_0(1,k) = C_0(2,k) = [\epsilon \ p_2(k) \ \epsilon], \ C_0(3,k) = C_0(4,k) = [\epsilon \ \epsilon \ p_3(k)]$$

The output can now be written as

$$y(k) = C_0(\phi(k),k) \otimes x(k)$$

A.2 Modeling with routing and ordering

Consider the machine setup of example A.1, but let k denote the cycle in which 2 jobs should be processed. The order of the jobs can only be determined before the process on machine 1. When one job starts before the other on machine 1 the order is determined for the rest of the process. Each job has different processing times on each machine, let $p_{j,i}(k)$ be the processing time of job j on machine i in cycle k. For each job it can be separately determined if it will be processed on machine 2 or 3, let $v_{j,i}(k)$ be equal to zero if job j is to be scheduled for machine i in cycle k where $i \in \{2,3\}$, it is equal to ϵ otherwise. Note that it has to be true that $v_{j,2}(k) \oplus v_{j,3}(k) = 0$ and $v_{j,2}(k) \otimes v_{j,3}(k) = \epsilon$ for all j, k since every job needs to be processed on exactly one machine after being processed on machine 1. Let $x(k)$ be the vector of starting times of all jobs in cycle k, so define

$$x(k) = \begin{bmatrix} x_{1,1}(k) \\ x_{2,1}(k) \\ x_{1,2}(k) \\ x_{2,2}(k) \\ x_{1,3}(k) \\ x_{2,3}(k) \end{bmatrix}$$

where $x_{j,i}(k)$ denotes the starting time of job j on machine i in cycle k. If job j is not processed on machine i in cycle k, $x_{j,i}(k)$ will not denote the starting time but then $x_{j,i}(k) = \epsilon$ is taken. To define the system routing define the matrix $A^r(v(k),k) \in \mathbb{R}_\epsilon^{6\times 6}$ as follows.

$$A^r(v(k),k) = \bigoplus_{j\in\{1,2\}, i\in\{2,3\}} (v_{j,i}(k) \otimes A^r_{j,i}(k)) \tag{A.1}$$

Now to make the routing correct let

$$\begin{aligned} &[A^r_{1,2}(k)]_{3,1} = p_{1,1}(k),\ [A^r_{1,3}(k)]_{5,1} = p_{1,1}(k), \\ &[A^r_{2,2}(k)]_{4,2} = p_{2,1}(k),\ [A^r_{2,3}(k)]_{6,2} = p_{2,1}(k) \end{aligned} \tag{A.2}$$

and $[A^r_{j,i}(k)]_{e_1,e_2} = \epsilon$ for all other values of i, j, e_1, e_2. The next step is to define system matrices that determine the ordering of the jobs. Define the matrix $P_0(v(k),k)$ as in equation (2.24) and establish that it can be taken equal to

$$P_0(v(k),k) = \begin{bmatrix} \epsilon & 0 & \epsilon & \epsilon & \epsilon & \epsilon \\ 0 & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & v_{1,2}(k)\otimes v_{2,2}(k) & \epsilon & \epsilon \\ \epsilon & \epsilon & v_{1,2}(k)\otimes v_{2,2}(k) & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & v_{1,3}(k)\otimes v_{2,3}(k) \\ \epsilon & \epsilon & \epsilon & \epsilon & v_{1,3}(k)\otimes v_{2,3}(k) & \epsilon \end{bmatrix}$$

for this case. When looking at the previous μ-th cycle jobs are always scheduled both on machine 1, and if they are both scheduled on machine 2 or 3 depends on the variables $v_{j,i}(k)$ and $v_{j,i}(k-\mu)$. It can therefore be obtained that for $\mu > 0$ the matrix $P_\mu(v(k),k)$ can be written as in equation (A.3).

$$P_\mu(v(k),k) = \tag{A.3}$$
$$\begin{bmatrix} 0 & 0 & \epsilon & \epsilon & \epsilon & \epsilon \\ 0 & 0 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & v_{1,2}(k) \otimes v_{1,2}(k-\mu) & v_{1,2}(k) \otimes v_{2,2}(k-\mu) & \epsilon & \epsilon \\ \epsilon & \epsilon & v_{2,2}(k) \otimes v_{1,2}(k-\mu) & v_{2,2}(k) \otimes v_{2,2}(k-\mu) & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & v_{1,3}(k) \otimes v_{1,3}(k-\mu) & v_{1,3}(k) \otimes v_{2,3}(k-\mu) \\ \epsilon & \epsilon & \epsilon & \epsilon & v_{1,3}(k) \otimes v_{2,3}(k-\mu) & v_{2,3}(k) \otimes v_{2,3}(k-\mu) \end{bmatrix}$$

Let the binary variable $v_j(k)$ be equal to 0 if job j is the first job of the cycle k that is going to be processed and $v_j(k) = \epsilon$ otherwise. Remember that the orders cannot be swapped later on a different machine. The system matrix that determines the order for all jobs in cycle k is given by

$$Q_0(v(k),k) = \begin{bmatrix} \epsilon & v_2(k) & \epsilon & \epsilon & \epsilon & \epsilon \\ v_1(k) & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & v_2(k) & \epsilon & \epsilon \\ \epsilon & \epsilon & v_1(k) & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & v_2(k) \\ \epsilon & \epsilon & \epsilon & \epsilon & v_1(k) & \epsilon \end{bmatrix}$$

For the cycle $k-\mu$ it follows from the assumptions that all jobs must be processed before the jobs of cycle k so it is obtained for $\mu > 0$ that

$$Q_\mu(v(k),k) = \begin{bmatrix} 0 & 0 & \epsilon & \epsilon & \epsilon & \epsilon \\ 0 & 0 & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & 0 & 0 & \epsilon & \epsilon \\ \epsilon & \epsilon & 0 & 0 & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & 0 & 0 \\ \epsilon & \epsilon & \epsilon & \epsilon & 0 & 0 \end{bmatrix}$$

Finally, the matrices $H_\mu(k)$ determine the separation times between the starting times of the jobs between cycle k and $k-\mu$. It is clear that the matrices $H_\mu(k)$ should give the proper processing times on the correct indices. It can be obtained that:

$$H_0(k) = \begin{bmatrix} \epsilon & p_{2,1}(k) & \epsilon & \epsilon & \epsilon & \epsilon \\ p_{1,1}(k) & \epsilon & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & p_{2,2}(k) & \epsilon & \epsilon \\ \epsilon & \epsilon & p_{1,2}(k) & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & \epsilon & p_{2,3}(k) \\ \epsilon & \epsilon & \epsilon & \epsilon & p_{1,3}(k) & \epsilon \end{bmatrix}$$

$$H_\mu(k) = \begin{bmatrix} p_{1,1}(k-\mu) & p_{2,1}(k-\mu) & \epsilon & \epsilon & \epsilon & \epsilon \\ p_{1,1}(k-\mu) & p_{2,1}(k-\mu) & \epsilon & \epsilon & \epsilon & \epsilon \\ \epsilon & \epsilon & p_{1,2}(k-\mu) & p_{2,2}(k-\mu) & \epsilon & \epsilon \\ \epsilon & \epsilon & p_{1,2}(k-\mu) & p_{2,2}(k-\mu) & \epsilon & \epsilon \\ \epsilon & \epsilon & \epsilon & \epsilon & p_{1,3}(k-\mu) & p_{2,3}(k-\mu) \\ \epsilon & \epsilon & \epsilon & \epsilon & p_{1,3}(k-\mu) & p_{2,3}(k-\mu) \end{bmatrix}$$

where the second matrix is valid for $\mu > 0$. The ordering system matrix can now be determined by equation (A.4).

$$A^o_\mu(v(k),k) = P_\mu(v(k),k) \odot Q_\mu(v(k),k) \odot H_\mu(k) \tag{A.4}$$

The system matrix for the SMPL system can now be determined and is given by equation (A.5).

$$A_0(v(k),k) = A^r(v(k),k) \oplus A^o_0(v(k),k) \tag{A.5}$$
$$A_\mu(v(k),k) = A^o_\mu(v(k),k) \text{ if } \mu > 0$$

Now define $r_j(k)$ as the release date a job j has in cycle k. A job can be processed only after its release dates, and when all jobs in the previous cycle on that machine are completed. Also consider the case for this example where a machine can remain idle, even when a job is available for that machine. There are several ways to model the release dates and machine idle time. Since the release dates of jobs are variable per cycle, they will be modeled outside of the input matrix B. Now let

$$u(k) = \begin{bmatrix} r_1(k) \otimes w_1(k) \\ r_2(k) \otimes w_2(k) \end{bmatrix}, \; B = \begin{bmatrix} 0 & \epsilon \\ \epsilon & 0 \\ \epsilon & \epsilon \\ \epsilon & \epsilon \\ \epsilon & \epsilon \\ \epsilon & \epsilon \end{bmatrix} \tag{A.6}$$

where $w_j(k)$ is a control variable that denotes the least amount of time after time $r_j(k)$ machine 1 is not processing job j in cycle k. The control variable $w_j(k)$ is typically used to let a job wait until another high priority job comes available for processing. The system matrices are now constructed and the final system equation is given by

$$x(k) = \left(\bigoplus_{\mu=0}^{k} A_\mu(v(k),k) \otimes x(k-\mu) \right) \oplus B \otimes u(k) \tag{A.7}$$

The output of the system $y_j(k)$ is defined by the time a job j of cycle k is finished and can be calculated by equation (A.8).

$$y(k) = \begin{bmatrix} y_1(k) \\ y_2(k) \end{bmatrix} = \begin{bmatrix} \epsilon & \epsilon & p_{1,2}(k) & \epsilon & p_{1,3}(k) & \epsilon \\ \epsilon & \epsilon & \epsilon & p_{2,2}(k) & \epsilon & p_{2,3}(k) \end{bmatrix} \otimes x(k) \tag{A.8}$$

The modeling method in this example is different from the example in Appendix A.1. In Appendix A.1 only cycle k and $k-1$ are used to determine the state variables of cycle k while in Appendix A.2 all the previous cycles are required. The difference is the interpretation of $x_{j,i}(k)$ when a job is not processed on a machine. Both methods can be correctly implemented, but the method of example Appendix A.2 will result in a very large amount of variables if one has to compute a large amount of cycles. In a continuous cyclic system this would be a problem. It could be resolved by setting $\mu_{\max}$ fixed and smaller than $k-1$, but large enough such that the jobs before cycle $k-\mu_{\max}$ do not effect jobs in cycle k (if that is even possible).

A.3 Transformation of SMPL to MPC

This example is an extension of the example in section A.2 and will use the same model but with given parameters. The idea is to show how the MPC control problem can be transformed into a MILP. The system equations used are equations (A.7) and (A.8). There are 5 cycles with each 2 jobs which each have a due date. Let $d_j(k)$ be the due date of job j in cycle k. Every cycle has a fixed duration of 6 time units and it can be the case that a job is not finished before the end of its cycle. The initial prediction horizon will be $N_p = 2$. After each optimization, the prediction interval moves up in time the length of a cycle. For optimization only the operations of the current cycle k up to cycle $k + N_p - 1$ are taken into consideration. The data regarding processing times, release dates and due dates is found in table A.1.

		Job 1					Job 2				
		$p_{1,1}$	$p_{1,2}$	$p_{1,3}$	r_1	d_1	$p_{2,1}$	$p_{2,2}$	$p_{2,3}$	r_2	d_2
k	1	3	4	4	0	7	4	5	6	1	8
	2	6	6	4	6	16	2	8	6	6	15
	3	2	7	7	13	20	3	6	6	12	23
	4	1	5	5	20	24	4	4	7	19	26
	5	3	3	4	24	31	3	5	3	25	31

Table A.1: Given data for 5 cycles of the model in Appendix A.2

The objective function for the MPC problem is given by equation (A.9) and should be minimized. It denotes the weighted maximum completion time for the jobs of the last cycle plus the total tardiness of all jobs from cycle k to $k + N_p - 1$. Recall that k is now fixed per optimization problem, k will only increase with value 1 after the optimization stops.

$$J_{N_p}(k) = \delta(k) \max_{j=1,2}\{y_i(k + N_p - 1)\} + \sum_{\tau=0}^{N_p-1} \sum_{j=1,2} \max\{(y_j(k + \tau) - d_j(k + \tau)), 0\} \tag{A.9}$$

Here $\delta(k) = \frac{3}{1+2k}$ is a regression model such that the maximum completion time has a decreasing weight that compensates the high completion times in higher cycles. If $\delta(k)$ was constant for each k the makespan of the system would be less important in the first optimization but become more important in later optimizations because the time has attained higher values. The decreasing $\delta(k)$ makes sure the makespan has a similar weight to the tardiness of the objective function for any given time.
The objective function is not in linear form due to the max operations. Such expressions can be easily transformed to the linear forms by adding a variable. Introduce variable C and let

$$C \geq y_1(k + N_p - 1) \text{ and } C \geq y_2(k + N_p - 1) \tag{A.10}$$

be additional constraints such that C is larger or equal to the maximum completion time. Since the problem is a minimization and $\delta(k) > 0$, it will take C as small as possible and C thus will be equal to the maximum completion time of jobs in cycle $k + N_p - 1$. The same trick can be done for the tardiness of jobs. Introduce variable $T_j(k + \tau)$ which should denote the tardiness of job j in cycle $k + \tau$. The constraints are given by

$$T_j(k + \tau) \geq 0, \; j = 1,2 \; \tau = 0, \ldots, N_p - 1 \tag{A.11}$$

$$y_j(k + \tau) - d_j(k + \tau) \leq T_j(k + \tau), \; j = 1,2 \; \tau = 0, \ldots, N_p - 1 \tag{A.12}$$

The objective function for $N_p = 2$ is now given by

$$J_{N_p}(k) = C + T_1(k) + T_2(k) + T_1(k + 1) + T_2(k + 1)$$

The next step is to define constraints such that the variables behave like the model in section A.2. First, start with the routing constraints. It can be obtained from equations (A.1) and (A.2) that

$$p_{j,1}(k + \tau) + x_{j,1}(k + \tau) + v_{j,i}(k + \tau) \leq x_{j,i}(k + \tau), \; j = 1,2 \; i = 2,3 \; \tau = 0, \ldots, N_p - 1$$

Since the equation is required to be of binary variables in the sense of conventional algebra, the binary transformation is done to obtain routing constraints:

$$p_{j,1}(k+\tau)+x_{j,1}(k+\tau)+\beta(1-v^b_{j,i}(k+\tau)) \leq x_{j,i}(k+\tau),\ j=1,2\ i=2,3\ \tau=0,\ldots,N_p-1 \tag{A.13}$$

The next step is to define the constraints such that the ordering is done according to the model. Recall that binary variables $v_j(k+\tau)$ denote whether job 1 or 2 is processed first in cycle $k+\tau$. Also notice from the ordering matrices of section A.2 that there are different system matrices for $\mu=0$ and $\mu \geq 1$. First, the constraints for $\mu=0$ will be determined. It is important to note that exactly one of the variables $v_1(k)$ and $v_2(k)$ is true and the other is false. In conventional algebra, this can be written as the constraint

$$v^b_1(k)+v^b_2(k)=1 \tag{A.14}$$

Now redefine the starting time of a job j that does not enter machine i in cycle time to $x_{j,i}(k)=0$ instead of ϵ. It will still be clear from the resulting x vector which job will enter which machine when. To make sure this holds add the constraint:

$$x_{j,i}(k+\tau) \leq -\beta v_{j,i}(k+\tau),\ j=1,2,\ i=2,3,\ \tau=0,\ldots,N_p-1 \tag{A.15}$$

For the ordering on machines 1,2 and 3 in cycle k the corresponding constraints are equal to

$$x_{2,1}(k+\tau)+p_{2,1}(k+\tau)+\beta(1-v^b_2(k+\tau)) \leq x_{1,1}(k+\tau) \tag{A.16}$$
$$x_{1,1}(k+\tau)+p_{1,1}(k+\tau)+\beta(1-v^b_1(k+\tau)) \leq x_{2,1}(k+\tau) \tag{A.17}$$
$$p_{2,2}(k+\tau)+x_{2,2}(k+\tau)+\beta(3-v^b_2(k+\tau)-v^b_{1,2}(k+\tau)-v^b_{2,2}(k+\tau)) \leq x_{1,2}(k+\tau) \tag{A.18}$$
$$p_{1,2}(k+\tau)+x_{1,2}(k+\tau)+\beta(3-v^b_1(k+\tau)-v^b_{1,2}(k+\tau)-v^b_{2,2}(k+\tau)) \leq x_{2,2}(k+\tau) \tag{A.19}$$
$$p_{2,3}(k+\tau)+x_{2,3}(k+\tau)+\beta(3-v^b_2(k+\tau)-v^b_{1,3}(k+\tau)-v^b_{2,3}(k+\tau)) \leq x_{1,3}(k+\tau) \tag{A.20}$$
$$p_{1,3}(k+\tau)+x_{1,3}(k+\tau)+\beta(3-v^b_1(k+\tau)-v^b_{1,3}(k+\tau)-v^b_{2,3}(k+\tau)) \leq x_{2,3}(k+\tau) \tag{A.21}$$

for $\tau=0,\ldots,N_p-1$. Notice that there is only one constraint for each variable $x_{j,i}(k)$ since there is only one entry in each row of the matrix $A^o_0(v(k),k)$, this means the maximization operation is absent. Similarly, the constraints for $\mu \geq 1$ can be determined. However, the matrix $A^o_\mu(v(k),k)$ can have multiple entries per row, which results in multiple constraints due to the max operator. For machine 1 the constraints are defined by

$$x_{1,1}(k+\tau-\mu)+p_{1,1}(k+\tau-\mu) \leq x_{j,1}(k+\tau) \tag{A.22}$$
$$x_{2,1}(k+\tau-\mu)+p_{2,1}(k+\tau-\mu) \leq x_{j,1}(k+\tau) \tag{A.23}$$

for $j=1,2$, $\tau=0,\ldots,N_p-1$ and $\mu=1,\ldots,k+\tau$. Notice that it is possible that μ is larger than τ. In this case some variables from a previous cycle is required. These variables are already determined and can be chosen equal to the values they are attained in the past. For machine 2 and 3 one can derive constraints (A.24)-(A.27).

$$x_{1,i}(k+\tau-\mu)+p_{1,i}(k+\tau-\mu)+\beta(2-v^b_{1,i}(k+\tau)-v^b_{1,i}(k+\tau-\mu)) \leq x_{1,i}(k+\tau) \tag{A.24}$$
$$x_{2,i}(k+\tau-\mu)+p_{2,i}(k+\tau-\mu)+\beta(2-v^b_{1,i}(k+\tau)-v^b_{2,i}(k+\tau-\mu)) \leq x_{1,i}(k+\tau) \tag{A.25}$$
$$x_{1,i}(k+\tau-\mu)+p_{1,i}(k+\tau-\mu)+\beta(2-v^b_{2,i}(k+\tau)-v^b_{1,i}(k+\tau-\mu)) \leq x_{2,i}(k+\tau) \tag{A.26}$$
$$x_{2,i}(k+\tau-\mu)+p_{2,i}(k+\tau-\mu)+\beta(2-v^b_{2,i}(k+\tau)-v^b_{2,i}(k+\tau-\mu)) \leq x_{2,i}(k+\tau) \tag{A.27}$$

Here, take $\tau = 1, \ldots, N_p - 1, \mu = 1, \ldots, k + \tau$ and $i = 1, 2$. To make jobs start no earlier than their release dates, one can simply implement the constraints:

$$r_j(k + \tau) \leq x_{j,1}(k + \tau) \tag{A.28}$$

for $j = 1, 2$ and $\tau = 0, \ldots, N_p - 1$. The next step is to define constraints such that the output comes out correct. This is easily done by the constraints

$$x_{j,2}(k + \tau) + p_{j,2}(k + \tau) \leq y_j(k + \tau) \tag{A.29}$$
$$x_{j,3}(k + \tau) + p_{j,3}(k + \tau) \leq y_j(k + \tau) \tag{A.30}$$

for $\tau = 0, \ldots, N_p - 1$. Finally, every binary variable $v^b_{j,i}(k)$ or $v^b_i(k)$ should be bounded. Let v be the vector containing all binary decision variables, simply denote $0 \leq v_i \leq 1$ for every entry of v. The MILP is now finished and is of the form (B.3) and (B.4) on page 110, even though there is no variable $w_j(k)$ as defined in equation (A.6). For the result this does not matter, since the system equation is changed to a system inequality in the constraints, so jobs are allowed to wait to leave a machine idle.
The work of the controller on the model s simulated in MATLAB. The prediction horizon is $N_p = 2$, and the controller will calculate the optimal schedule for the next N_p cycles and then only use the input variables of the first cycle that has to be scheduled. Figure A.2 shows one step of the MPC controller, namely for cycle $k = 2$. It optimizes the schedule over cycle 2 and 3 (dotted and striped lines) but only uses the schedule of cycle 2 for control. The next step is to let the MPC controller work for cycle 3. When all optimal schedules are computed the controller stops and the final resulting schedule is shown in figure A.3. The grey areas indicate the tardiness of jobs.

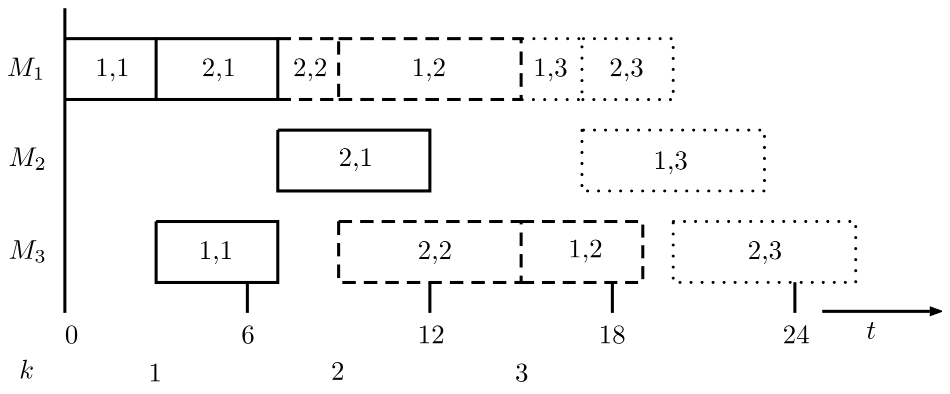


Figure A.2: Single step of the MPC controller. The dotted and striped lines indicate the prediction of jobs, but only the striped are used for control. Jobs covered by full lines are in the past cycles. The numbers in the blocks represent j, k and denote which job j is processed in cycle k. Grey areas indicate the tardiness of jobs.

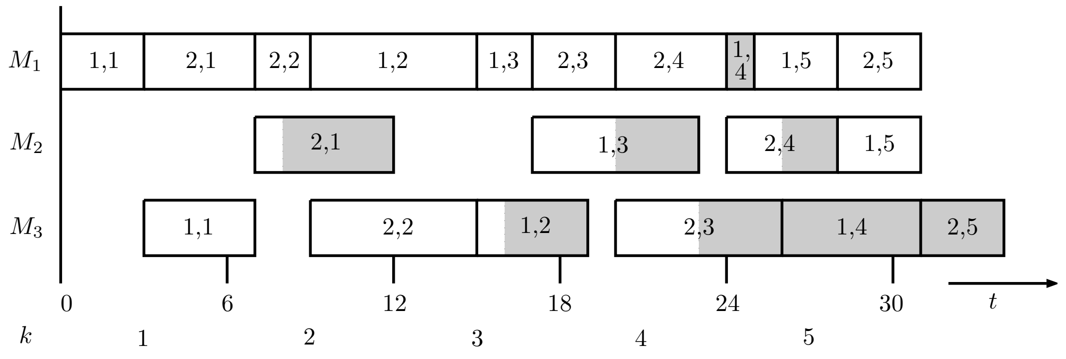


Figure A.3: Optimal schedule for the example of this section. The numbers in the blocks represent j, k and denote which job j is processed in cycle k. Grey areas indicate the tardiness of jobs.

This example is now completed.

B Linear Optimization

The concept of linear programming is a popular method to solve real optimization problems. One uses a model to describe a problem and the type of model will determine what possible algorithms could solve the problem. In this report, optimization problems are restricted to linear optimization problems, since the max-plus system models nicely fit into linear models. However, there are plenty of strategies to discuss that can be applied to linear models. The research of this thesis will use linear optimization to control SMPL systems. In order to decrease the computational workload on the controller, one first needs to understand what a linear program is and why it can be difficult to solve such a problem. Afterwards, this section will explain how the problem is solved and how the solution offers room for improvement.
Before continuing into this chapter, some definitions and properties of linear algebra are given. The following definitions are basics to linear algebra as it is described in Friedberg et al. [22].

Definition B.1. *A finite set of vectors $x_1, \dots, x_k \in \mathbb{R}^n$ is said to be linearly independent if the unique solution to $\sum_{i=1}^{k} \lambda_i x_i = 0,\ \lambda_i \in \mathbb{R}$ is given by $\lambda_i = 0$ for each $i = 1, \dots, k$.*

Theorem B.2. *The following statements are equivalent for $A = [a_1, a_2, \dots, a_n] \in \mathbb{R}^{n \times n}$:*

- *$a_1, a_2, \dots, a_n$ are linearly independent.*
- *A is invertible*
- *$det(A) \neq 0$.*
- *For any $b \in \mathbb{R}^n$ the equation $Ax = b$ has exactly one solution.*

Definition B.3. *A finite set of vectors $x_1, \dots, x_k \in \mathbb{R}^n$ is said to be affinely independent if the unique solution to $\sum_{i=1}^{k} \lambda_i x_i = 0,\ \sum_{i=1}^{k} \lambda_i = 0,\ \lambda_i \in \mathbb{R}$ is given by $\lambda_i = 0$ for each $i = 1, \dots, k$.*

From definition of linear- and affine independency, it can be seen that linear independence implies affine independence. The converse however is not true, as described in Friedberg et al. [22].

Theorem B.4. *The following statements are equivalent:*

- *$x_0, x_1, \dots, x_k$ are affinely independent.*
- *$x_1 - x_0, x_2 - x_0, \dots, x_k - x_0$ are linearly independent.*

B.1 Linear programming

To solve an optimization problem one first has to establish a valid formulation for the optimization problem. There are several type of formulations, a common one is the Linear Program (LP). An LP is defined by the program

$$\max_{x \in \mathbb{R}^n} \quad c^T x \qquad \text{s.t.} \tag{B.1}$$

$$Ax \leq b \tag{B.2}$$

Here $c \in \mathbb{R}^n, b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$ are given model parameters and are called the input of the LP. Expression $c^T x$ is the *objective function* that is supposed to be optimized. $Ax \leq b$ are the *constraints* of the problem, any solution should satisfy these inequalities. Any solution x for

which $Ax \leq b$ holds is called a *feasible* solution. If x is not feasible it is called *infeasible.* The entries of x are said to be the *decision variables* of the problem. Any *bounds* on the entries of x should also be given in the constraints. One can also say that the program optimizes $c^T x$ over $Ax \leq b$.
The best known ways to solve the LP are by the *simplex algorithm* which was found by Dantzig et al. [16] or the *interior point method* described in Kojima et al. [35]. The algorithms are guaranteed to find the optimal solution. It was proven by Karmarker [33] that the interior point method can actually find the optimal solution to the LP in polynomial time. However, in practice, the simplex method and interior point method neither dominate the other in computation time. It is important to notice that a computer cannot store any irrational numbers. Since the simplex algorithm is usually performed by the computer, $\mathbb{R}^n$ in (B.1) is usually restricted to be $\mathbb{Q}^n$ for computational purposes.
One of the disadvantages of the LP model is that it cannot model discrete decisions. For example, let us consider the possibility where one chooses to buy a specific item or not buy that item. One could have $x = 1$ if the item is bought and $x = 0$ if the item is not bought. The variable x is then restricted to be *integer*, say $x \in \mathbb{Z}$. Such modeling properties can be described in a *Mixed-Integer Linear Program* (MILP). The difference with an LP is that the MILP restricts some decision variables to be integer. This gives the following formulation for an MILP:

$$\max_{x \in \mathbb{R}^n, v \in \mathbb{Z}^k} \quad c^T x + f^T v \qquad \text{s.t.} \tag{B.3}$$

$$Ax + Bv \leq b \tag{B.4}$$

Here $c \in \mathbb{R}^n, f \in \mathbb{R}^k, b \in \mathbb{R}^m, A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{m \times k}$ are the model parameters. Notice that a LP is a special instance of the MILP, which makes the MILP harder to solve in general. The LP and MILP can be recasted as minimization problems any time by the transformation $\max\{c^T x\} = -\min\{-c^T x\}$. The minimization versions of the LP and MILP are therefore considered as the same problems.

B.2 Convexity, polyhedra & formulations

To understand how to solve an LP or MILP some mathematical definitions are explained in this section. The definitions in this section of the Appendix can also be found in Wolsey [54].

Definition B.5. *A set X is said to be convex if for any $x_1, x_2 \in X$ and $\lambda \in [0, 1]$ it follows that $\lambda x_1 + (1 - \lambda)x_2 \in X$.*
A function $f : X \to \mathbb{R}$ is said to be convex if for any $x_1, x_2 \in X$ and $\lambda \in [0, 1]$ it follows that $f(\lambda x_1 + (1 - \lambda)x_2) \leq \lambda f(x_1) + (1 - \lambda)f(x_2)$.
Let $x_1, \ldots, x_n \in X$ and non-negative $\lambda_1, \ldots, \lambda_n \in \mathbb{R}$ such that $\lambda_1 + \ldots + \lambda_n = 1$. Now

$$x = \sum_{i=1}^{n} \lambda_i x_i$$

is said to be a convex combination of $x_1, \ldots, x_n$.
An optimization problem is said to be convex if the objective function is convex and the set of feasible solutions is convex.

Convexity plays an important role in optimization. When optimizing a convex function over a convex set any local optimum met is sure to be a global optimum. Barvinok [8] described the concept of convex optimization and why a local optimum equals a global optimum in convex optimization.

Theorem B.6. (Barvinok [8]) *Let $\mathcal{F}$ be a family of convex sets. Then*

$$X = \bigcap_{A \in \mathcal{F}} A \tag{B.5}$$

is a convex set.

Proof. Let $x_1, x_2 \in X$ where X is defined as in equation (B.5) and let $\lambda \in [0, 1]$. By definition of the intersection operator it follows that x_1, x_2 are in A for every $A \in \mathcal{F}$. Since $A \in \mathcal{F}$ is convex it follows that $\lambda x_1 + (1 - \lambda)x_2 \in A$. Since this holds for every $A \in \mathcal{F}$, it can directly be concluded that

$$\lambda x_1 + (1 - \lambda)x_2 \in \bigcap_{A \in \mathcal{F}} A = X \tag{B.6}$$

which completes the proof. □

Since any linear half-space of the form $a^T x \leq b$, for $a \in \mathbb{R}^n$, $b \in \mathbb{R}$ is a convex set it follows directly from Theorem B.6 that the set generated by the matrix inequalities (B.2) is a convex set. Since objective function (B.1) is a convex function it can be concluded that the LP is a convex optimization problem.

Definition B.7. *A subset $P \subset \mathbb{R}^n$ is called an polyhedron if there exist a $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ for some $m \in \mathbb{N}$ such that $P = \{x \in \mathbb{R}^n \ : \ Ax \leq b\}$. P is called a polytope if P is a polyhedron and it is bounded. The dimension of a polyhedron P, denoted by $dim(P)$, is the maximum number of affinely independent elements in P minus one.*

Notice that if a_i are the rows of A, it can be obtained that

$$\{x \in \mathbb{R}^n \ : \ Ax \leq b\} = \bigcap_{i=1}^{m} \{x \in \mathbb{R}^n \ : a_i x \leq b_i\} \tag{B.7}$$

Since for any $a \in \mathbb{R}^{1 \times n}$ and $b \in \mathbb{R}$ the set $\{x \in \mathbb{R}^n \ : ax \leq b\}$ is convex, it can be concluded from equation (B.7) and Theorem B.6 that any polyhedron is convex. There are multiple ways to describe a polytope. One can for example also take the convex hull of a set of elements to describe a polytope.

Definition B.8. *The convex hull of a set X denoted by* conv(X) *is the set of all convex combinations of elements of X.*

Definition B.9. *A point x of a polyhedron P is said to be an extreme point of P if whenever for given $x_1, x_2 \in P$ and $\lambda \in [0, 1]$ with the property that $x = \lambda x_1 + (1 - \lambda)x_2$ it directly follows that $x = x_1 = x_2$.*

Theorem B.10. *Any polytope is the convex hull of its extreme points.*

It turns out that the LP is always optimal in at least one extreme point (Barvinok [8]) of the polyhedron. Dantzig's simplex algorithm [16] calculates an extreme point of the polyhedron described by (B.2). It determines whether the objective function is optimal on that extreme point. If not, the algorithm traverses on the boundary of the polyhedron to find a new extreme point. This process is iterated until the global optimum is found and this is done in finite time.
The MILP can in general not be solved by the simplex algorithm (it is possible in some instances). The simplex algorithm calculates the optimal extreme point of the given polytope, if y of formulation (B.3) & (B.4) does not attain integer values at this extreme point, it is not feasible for the MILP and thus does the solution not solve the MILP.

Definition B.11. *Let X be a subset of $\mathbb{R}^n \times \mathbb{Z}^k$. A polyhedron $P \subset \mathbb{R}^n \times \mathbb{R}^k$ is a formulation for X if $X = P \cap (\mathbb{R}^n \times \mathbb{Z}^k)$.*

Notice that a formulation for a subset of $\mathbb{R}^n \times \mathbb{Z}^k$ is not unique. It is important to note that whenever a formulation P satisfies $P = \text{conv}(X)$ the extreme points of P are feasible solutions of X, what means that the simplex algorithm solves the MILP over X. With the right formulation, the MILP can be solved by the simplex algorithm which works well in practice. However, in practice, for many problems it turns out that it can be very difficult to find a formulation for the MILP that equals the convex hull of X. There are multiple ways to solve the general MILP, this will be explained in Section B.4.

Example B.12. Consider the following subset of $\mathbb{Z}^2$.

$$X = \{(0,1),(0,2),(0,3),(0,4),(1,1),(1,2),(1,3),(1,4),(2,2),(2,3),(2,4),(3,3)\}$$

A valid formulation for X is

$$P_1 = \{(v_1, v_2) \ : \ (v_1, v_2) \in \mathbb{R}^2, 0 \leq v_1 \leq 3, v_2 \geq 1, 1.2v_2 - v_1 \geq 0.1, v_1 + 2.6v_2 \leq 12.5\}$$

since $P_1 \cap \mathbb{Z}^2 = X$. However, notice that $(3, \frac{31}{12})$ is an extreme point of P_1 but not an element of X. The formulation

$$P_2 = \{(v_1, v_2) \ : \ (v_1, v_2) \in \mathbb{R}^2, 0 \leq v_1, 1 \leq v_2 \leq 4, v_1 \leq v_2, v_1 + v_2 \leq 6\}$$

for X has however only integer extreme points. This means that $P_2 = \text{conv}(X)$.

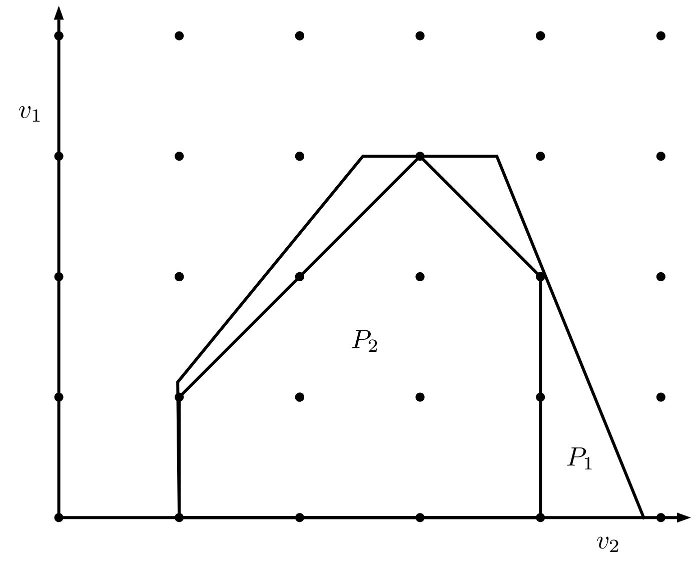


Figure B.1: Formulations P_1 and P_2 of example B.12.

In order to obtain a strong formulation of a feasible region X one can add constraints to a formulation P until it is considered good enough. However, adding many constraints is often not a good idea, because it raises the complexity of the simplex algorithm. One should therefore aim to improve the formulation for a region X while keeping the number of the constraints low. A constraint from polytope P is said to be *redundant* if P does not change after removing that constraint. To keep the complexity of the simplex algorithm optimal, one should always remove all redundant constraints from a formulation. Figure B.2 shows the graphical interpretation of a redundant constraint.

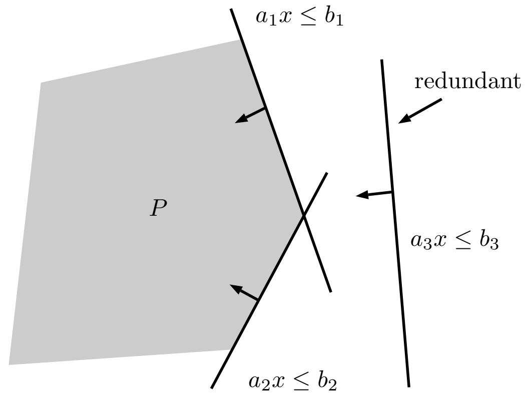


Figure B.2: Graphical interpretation of a redundant constraint, which is $a_3x \le b_3$

Definition B.13. *A cutting plane (a_1, a_2, b) with $a_1 \in \mathbb{R}^n$, $a_2 \in \mathbb{R}^k$, $b \in \mathbb{R}$ is valid for $X \subseteq \mathbb{R}^n \times \mathbb{Z}^k$ if $\forall (x, v) \in X$ it holds that $a_1^T x + a_2^T v \le b$. It is said that (a_1, a_2, b) separates $(\hat{x}, \hat{v})$ from X if $a_1^T \hat{x} + a_2^T \hat{v} > b$.*

Example B.14. Suppose an MILP has to be solved where $X = \{(x, v) \in \mathbb{R} \times \mathbb{Z} \ : \ x \le 3, x + 2v \ge 5, -7x + 10v \le 10\}$ and the objective is to minimize x over X. The actual optimal to this MILP is found in the point $(1, 2)$ with objective value 1. A valid formulation for X is given by $P_1 = \{(x, v) \in \mathbb{R}^2 \ : \ x \le 3, x + 2v \ge 5, -7x + 10v \le 10\}$. Now the relaxation of the MILP is to minimize x over P_1 and has the solution $(x_1^\star, v_1^\star) = (\frac{1}{2}, \frac{9}{4})$ with objective value $\frac{1}{2}$, so the relaxed solution is indeed a lower bound to the MILP. Now consider the cutting plane given by $(-14, 12, 15)$ and add it to P_1 to obtain P_2, which is still a formulation for X. Now $(x_1^\star, v_1^\star)$ is not feasible for P_2 since $-14 \cdot \frac{1}{2} + 12 \cdot \frac{9}{4} = 20 > 15$, so $(-14, 12, 15)$ separates $(x_1^\star, v_1^\star)$ from X. The new relaxed optimal given over P_2 is given by $(x_2^\star, v_2^\star) = (\frac{3}{4}, \frac{17}{8})$ with objective value $\frac{3}{4}$, which is still a lower bound to the MILP solution, but tighter. An illustration of this example is displayed in figure B.3.

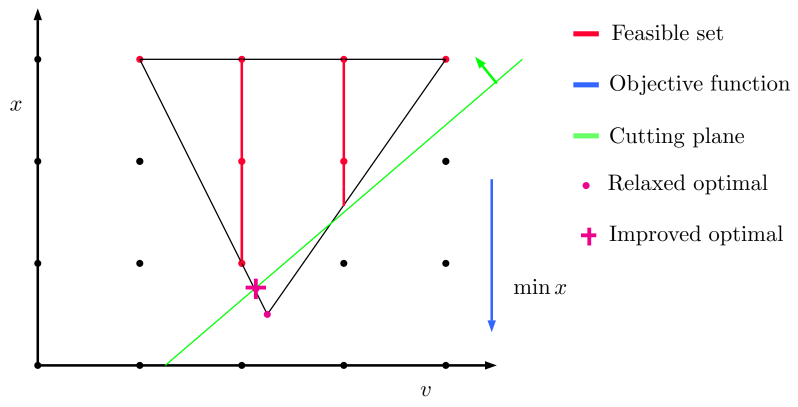


Figure B.3: Improving the relaxation with a cutting plane (Example B.14).

To improve a formulation P of a set X, one can find a valid cutting plane for X that separates elements $\hat{x} \in P \backslash \text{conv}(X)$. For optimization of the MILP it would be ideal to have the optimal solution at an extreme point of the given formulation.

Definition B.15. *A non-empty set F is a face of polyhedron $P = \{x : Ax \le b\}$, if there is an inequality $a_i x \le b_i$ that is valid for P such that $F = P \cap \{x : a_i x = b_i\}$. A set F is a facet of P if it is a face of P and $dim(F) = dim(P) - 1$. If an inequality $a_i x \le b_i$ is such that $P \cap \{x : a_i x = b_i\}$ is a facet of P, then $a_i x \le b_i$ is a facet-defining inequality* (FDI) *for P.*

In order to obtain the smallest formulation for $\mathrm{conv}(X)$ where $X \subseteq \mathbb{R}^n \times \mathbb{R}^k$, it is required to find all FDI's of $\mathrm{conv}(X)$. Depending on the problem specification, it can be hard to find all FDI's. Sometimes this is because the problem description is too complex, or sometimes because there are simply too many FDI's to compute.

B.3 Complexity theory

In computational mathematics and computer science it has always been a challenge to write the fastest algorithm. Besides deciding how to quantify the speed of an algorithm, it can be very challenging to decrease the workload of an algorithm, especially when the running time is not deterministic. At some point, one might consider if it is even possible to construct a faster algorithm. If one cannot find a better algorithm than an existing one, an important question in computational science is whether the algorithm is to blame, or is the problem to blame?
There are several ways to measure the running time of an algorithm, but the running time is usually dependent on the input size of the problem. In practice it is of course desired to have the running time of an algorithm in the least amount of real time seconds. Measuring the running time in real time is however *instance dependent.* It is possible (and usually the case) that an algorithm has different running times in seconds for different instances of the same size. A more interesting but theoretical approach of measuring the running time of algorithms is to measure the growth rate of the number of steps an algorithm has to compute in order to solve a problem in the *worst case scenario* with respect to the input size of the problem. This approach is described by Arora & Barak [5]. A summary of the basics of complexity theory will be discussed in this section.

Definition B.16. *Let $f, g : \mathbb{N} \to \mathbb{R}$ be non-negative functions. Define $\mathcal{O}(\cdot)$ to be a class of functions, where $f(n)$ is a function of $\mathcal{O}(g(n))$ (denote $f(n) = \mathcal{O}(g(n))$) if there exists a positive $c \in \mathbb{R}$ and $n_0 \in \mathbb{N}$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$. The definition of $\mathcal{O}$ is usually referred as "the big O-notation".*

It is common to use the smallest possible function for $g(n)$ in the notation of definition B.16. For example, clearly $3n^2 = \mathcal{O}(n^3)$, but it common to write $3n^2 = \mathcal{O}(n^2)$ instead. An algorithm with input size n that has at most $\mathcal{O}(n^k)$ computational steps for some finite $k \in \mathbb{N}$ it is said to have an *polynomial* running time. If the largest number of computational steps is a function of $\mathcal{O}(p^n)$ with $p > 1$ the algorithm has an *exponential* running time.
To qualify an algorithm it is often a good idea to determine to what *complexity class* the problem one is trying to solve belongs. Complexity classes are sets of problems that describe the complexity characteristics of the problems. Arora & Barak [5] gives a good overview of the most important complexity classes. The discussion around the theory of complexity classes is mostly focused on the class $\mathcal{NP}$. A *decision problem* is a problem which can be answered by a "yes" or a "no". A solution to the decision problem is information that does not answer the decision problem with a "yes" or a "no", but from this information one can derive whether the decision problem can be answered with a "yes" or a "no". For example, if the decision problem is whether it is possible to find a feasible $x \in X$ with objective value $c(x) \leq C$, and a solution $y \in X$ is given that shows to be feasible with $c(y) \leq C$, then the solution y leads to a "yes".
A decision problem belongs to $\mathcal{NP}$ if giving a solution leading to a "yes" or a "no" is of polynomial size with respect to the input size and it can verified whether the given solution leads to a "yes" or a "no" in polynomial steps with respect to input size.
A subset of $\mathcal{NP}$, called $\mathcal{P}$ is the class of decision problems that given a "no" solution, a solution can be obtained in polynomial time. It is left unanswered if $\mathcal{NP}$ is a subset of $\mathcal{P}$, this question is one of the so called millennium problems, which is a list of mathematical problems that are

stated by the Clay Mathematics Institute which are considered some of the most difficult and intriguing problems in mathematics.
Another subset of $\mathcal{NP}$ is the class of $\mathcal{NP}-$complete problems. This is the subset of problems of $\mathcal{NP}$ for which no polynomial time solving algorithm is known. It is however possible that an algorithm for such a problem exists, but as explained in Arora & Barak [5] that must mean that such an algorithm exists for all $\mathcal{NP}-$complete problems.
Most optimization problems are not decision problems. Optimization problems often optimize an objective function. Such an optimization problem can be recasted into a decision problem by considering a certain threshold. If there exists a feasible solution to the optimization problem such that the objective value is equal or better than the threshold, the decision version of the problem can be answered by a "yes". If this is not the case the decision variant can be answered with a "no".
The class of $\mathcal{NP}-$hard problems is the class op optimization problems from which their decision variant is an $\mathcal{NP}-$complete problem. Figure B.4 illustrates the set construction for complexity classes in the case that $\mathcal{P} \neq \mathcal{NP}$.

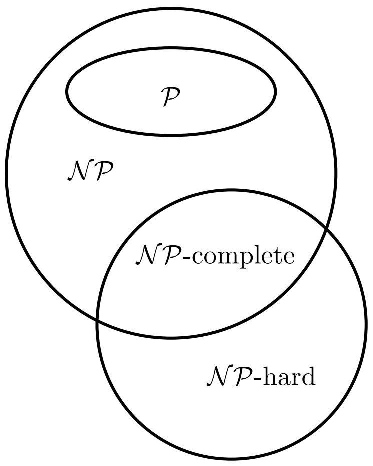


Figure B.4: The composition of the complexity classes in the case that $\mathcal{P} \neq \mathcal{NP}$.

By definition, there are no known polynomial running time algorithms that solve $\mathcal{NP}-$hard problems, unless they can all be solved in polynomial time. The search for polynomial time algorithms for $\mathcal{NP}-$hard problems has been going on for more than half a century. Since no one has ever since found a polynomial running time algorithm for a single $\mathcal{NP}-$hard problem, mathematicians consider it is unlikely that such an algorithm exists. It is therefore often assumed that $\mathcal{NP}$ is not a subset of $\mathcal{P}$, and the answer to the millennium problem is therefore likely that $\mathcal{NP} \neq \mathcal{P}$. It is therefore believed by many that the search for polynomial time algorithms for $\mathcal{NP}-$hard problems is a waste of time.

If one desires to solve a large instance of such a problem the worst case scenario running time can grow so large that it is considered not to be solved in a realistic amount of time. In order to solve such large instances one has to find a creative way to decrease the running time. Luckily, the theory for these complexity classes are all based on worst case scenarios. For some $\mathcal{NP}-$hard problems people have written algorithms that work really well in practice. These algorithms still have an exponential worst-case scenario running time but do most of the time the work in a reasonable amount of time. For example, CONCORDE is an algorithm written by Cook [15] which is very effective in solving the traveling salesman problem (Applegate [4]). The traveling salesman problem is $\mathcal{NP}-$hard but CONCORDE solves most large instances in seconds.

B.4 The branch-and-bound approach

The general method to solve an integer linear program is by the *branch-and-bound algorithm* described in Wolsey [54]. The branch and bound algorithm relies on solving the linear *relaxation* of the MILP. Let X be the set of feasible solutions of the MILP. The relaxation of an instance of the MILP is defined as the LP defined by the same constraints of the MILP but where the v vector is allowed to be non-integer. Note that if the the relaxation is optimized over P it has to hold that $X = P \cap (\mathbb{R}^n \times \mathbb{Z}^k)$, but P is certainly not unique. Let $L := \{P\}$ the initial set of remaining polytopes to use for optimization. The polytopes in L are referred as the *nodes* of the branch-and-bound algorithm. From now on, consider the MILP to be of the minimization form. Now choose a polytope from L (initially P) and optimize the objective function over the relaxation of this polytope. Optimizing the relaxation is of the form of an LP so this can be done by the simplex algorithm (Dantzig et al. [16]) or interior point method (Kojima et al. [35]). Let $(\hat{x}, \hat{v})$ be the optimal solution to the relaxation. Since $X \subseteq P$ it holds that P allows more solutions, so the objective value to the relaxed optimal solution is at least as good as the integer optimal. This means that the solution to the relaxation has a lower objective value than the solution of the MILP. Now if (this is usually unlikely) the $\hat{v}$ is integer this solution is also an element of X and the MILP is solved. If this is not the case, the branch-and-bound algorithm chooses an index i of $\hat{v}$ that is not integer and splits the problem to optimize over $P_1 := P \cap \{(x, v) \in (\mathbb{R}^n \times \mathbb{R}^k) : v_i \leq \lfloor \hat{v}_i \rfloor\}$ and $P_2 := P \cap \{(x, v) \in (\mathbb{R}^n \times \mathbb{R}^k) : v_i \geq \lceil \hat{v}_i \rceil\}$. In short, in this case P is removed from L and P_1 and P_2 are added tot L. Then, one of the remaining problem is chosen and the process is repeated. Which problem of L is chosen is a strategy on its own, and can be defined by the algorithm designer. The result is that the problem splits into a tree of nodes, this is referred to as *branching*. Each node represents a relaxed sub-problem of the MILP, and the *leaves* of the tree are exactly the elements in L. The initial polytope is called the *root*. The branching of a node will not occur in one of the following cases:

- The problem in a node is infeasible.
- The found solution to the relaxation has integer values for v.
- The found solution to the relaxation is worse than an earlier found integer solution.

Only for these three cases a node is removed from L and a node from L is chosen for optimization. This process is called *pruning*. To determine whether a solution is better or worse than the current best integer solution an upper bound u is used. Initially, u is set equal to ∞ and whenever an integer solution $(\hat{x}, \hat{v})$ is found that is better than all previous integer solutions set $u := c^T \hat{x} + f^T \hat{v}$. If L becomes empty the process is complete. The best solution of the relaxation of a sub-problem with integer v values is the optimal solution the the MILP.

Example B.17. Suppose one wants to solve the following program with the branch-and-bound algorithm.

$$\min_{v \in \mathbb{Z}^2} f^T v = v_1 + v_2 \text{ s.t.}$$

$$v \in P = \{v \in \mathbb{Z}^2 \ : \ 0 \leq v_1 \leq 2, 2v_1 \leq 3v_2, 5v_1 + 2v_2 \geq 5, 0 \leq v_2 \leq 3\}$$

A visualization of P is shown in figure B.5. The steps of the simplex algorithm where a relaxation is solved for non-integer values is not showed in this example to keep it short but still make the principles of branch and bound clear.

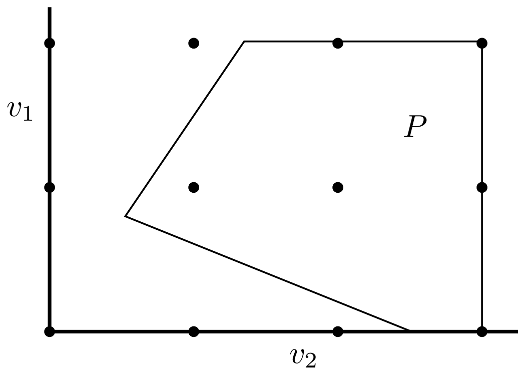


Figure B.5: The polytope formulated by P and the integer values as dots.

Let P_1 be the relaxation of P such that it has the same constraints as P but allows v to be an element of $\mathbb{R}^2$. Now let $u := \infty$ be the current upper bound and set $L := \{P_1\}$ and solve the relaxation over P_1 with the simplex algorithm. The solution is given by $\hat{v} = (\frac{15}{19}, \frac{10}{19})$ with objective value $f^T\hat{v} = \frac{25}{19}$. $\hat{v}$ is certainly no element of $\mathbb{Z}^2$, so a branching is required. Branch by removing P_1 from L and by adding $P_2 = P_1 \cap \{v \in \mathbb{R}^2 : v_2 \leq \lfloor\frac{10}{19}\rfloor = 0\}$ and $P_3 = P_1 \cap \{v \in \mathbb{R}^2 : v_2 \geq \lceil\frac{10}{19}\rceil = 1\}$ to L. By evaluation of the simplex algorithm over P_2 it can be concluded that P_2 is empty. So P_2 can be pruned and thus be removed from L. By evaluating the simplex algorithm over P_3 the problem is solved by optimal solution $\hat{v} = (\frac{3}{5}, 1)$ with value $f^T\hat{v} = \frac{8}{5}$. This is still no integer solution so branching is required. Remove P_3 from L and add $P_4 = P_3 \cap \{v \in \mathbb{R}^2 : v_1 \geq \lceil\frac{3}{5}\rceil = 1\}$ and $P_5 = P_3 \cap \{v \in \mathbb{R}^2 : v_1 \leq \lfloor\frac{3}{5}\rfloor = 0\}$ to L. The relaxation over P_4 is solved in $\hat{v} = (1, 1)$ with objective value $f^T\hat{v} = 2$. This is an integer solution, so P_4 can be pruned and the new upper bound to the problem becomes $u := 2$. The simplex algorithm solution over P_5 is $\hat{v} = (0, \frac{5}{2})$ with objective value $f^T\hat{v} = \frac{5}{2}$. Notice that $f^T\hat{v} > u$ so any integer solutions from branches of P_5 will result in solutions worse than $v = (1, 1)$. P_5 is therefore pruned and can be removed from L. The branch-and-bound algorithm has reached the point where L is empty and thus it stops. It is concluded that $v = (1, 1)$ is the optimal solution with objective value $f^Tv = 2$. An illustration of how the algorithm branches and prunes as in this example is shown in figure B.6.

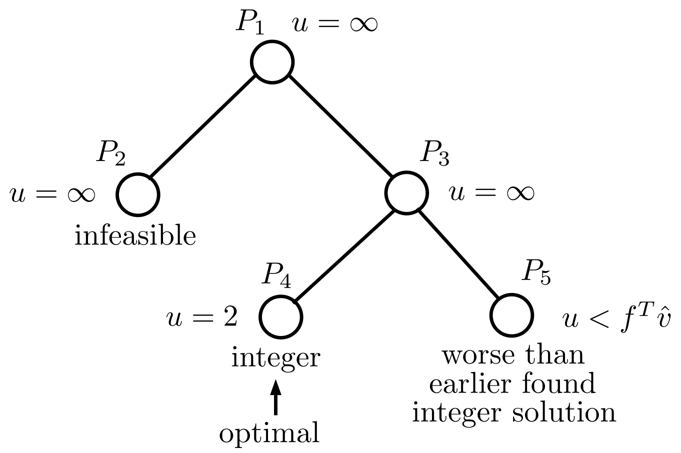


Figure B.6: The branch-and-bound search tree of example B.17.

The branch-and-bound algorithm is an exponential running time algorithm and can therefore take a long time to compute for large instances. However, there is room for improvement on the branch and bound algorithm. As mentioned earlier in appendix B.2 page 112, a formulation P for X is in general not unique. The choice of P can affect the number of branching processes in the

algorithm. Since the number of nodes grows exponentially with respect to the number of integer variables from the branching (Wolsey [54]), less branching results in a smaller computational time. When P and conv(X) are close it will be more likely that an integer solution will be found for the relaxation, so the initial formulation P is important.
A strong formulation of P does also improve the dual bound to the sub-problems. With a strong dual bound it is more likely that the relaxed value in a node exceeds global primal bound which results in pruning by the third criterion of pruning. The dual bound can also be improved by combinatorial bounds.
Another way to get more pruning in the branching tree is to find a good primal bound. Primal bounds are mainly found by heuristic solutions. With a stronger primal bound it is more likely that the relaxed optimal of a node exceeds the global optimum. Primal bounds can also be found by integer solutions from the branching tree.
Note that there are different strategies in choosing the order in which the sub-problems are solved, leading to different computational results. Figure B.7 illustrates the branch-and-bound algorithm in a flow chart. The striped squares are optional steps or steps where a certain strategy can be applied in order to improve the algorithm.

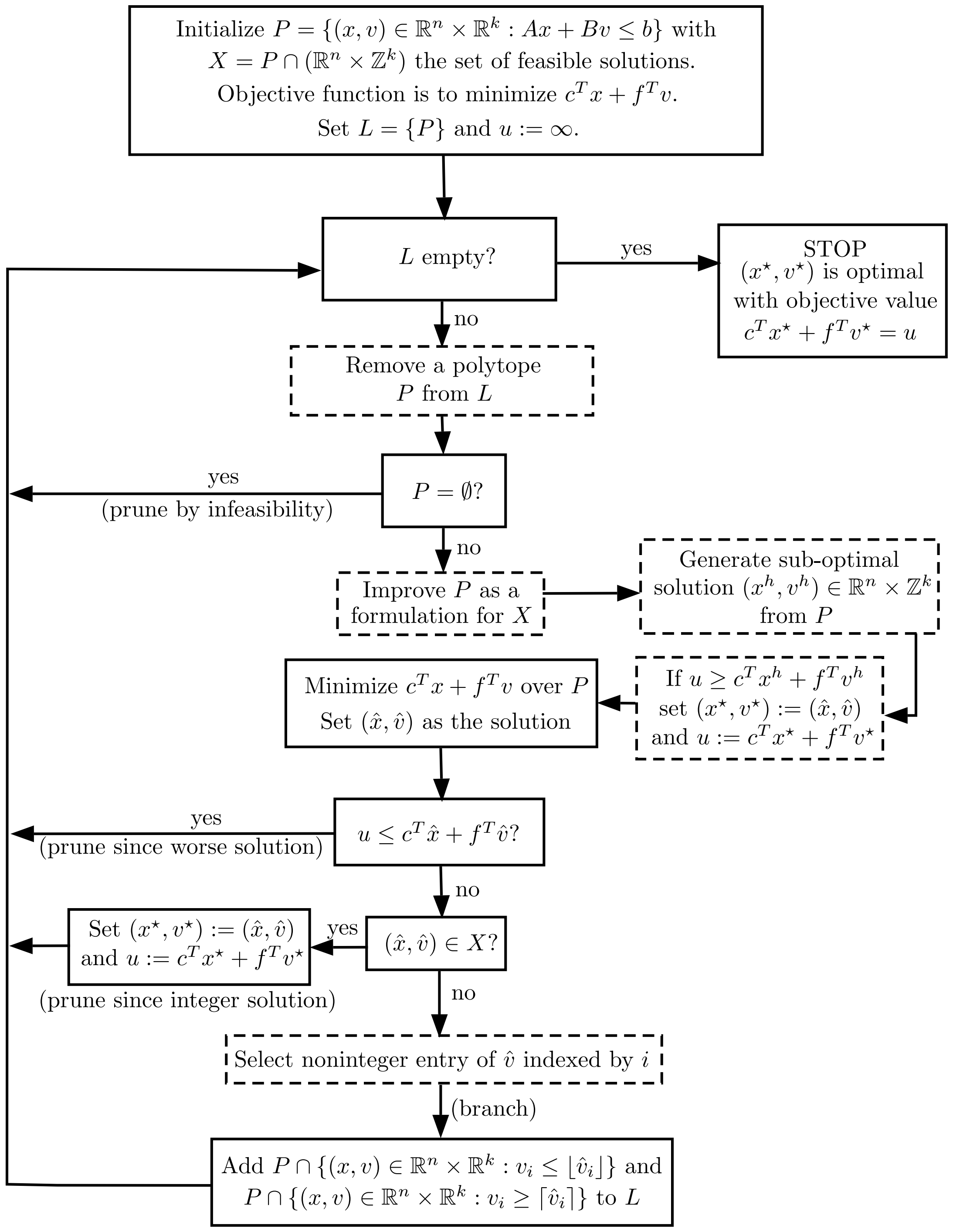

Initialize $P = \{(x,v) \in \mathbb{R}^n \times \mathbb{R}^k : Ax + Bv \leq b\}$ with $X = P \cap (\mathbb{R}^n \times \mathbb{Z}^k)$ the set of feasible solutions. Objective function is to minimize $c^T x + f^T v$. Set $L = \{P\}$ and $u := \infty$.
L empty?
yes
STOP $(x^\star, v^\star)$ is optimal with objective value $c^T x^\star + f^T v^\star = u$
no
Remove a polytope P from L
$P = \emptyset$?
yes
(prune by infeasibility)
no
Improve P as a formulation for X
Generate sub-optimal solution $(x^h, v^h) \in \mathbb{R}^n \times \mathbb{Z}^k$ from P
If $u \geq c^T x^h + f^T v^h$ set $(x^\star, v^\star) := (\hat{x}, \hat{v})$ and $u := c^T x^\star + f^T v^\star$
Minimize $c^T x + f^T v$ over P Set $(\hat{x}, \hat{v})$ as the solution
$u \leq c^T \hat{x} + f^T \hat{v}$?
yes
(prune since worse solution)
no
$(\hat{x}, \hat{v}) \in X$?
yes
Set $(x^\star, v^\star) := (\hat{x}, \hat{v})$ and $u := c^T x^\star + f^T v^\star$
(prune since integer solution)
no
Select noninteger entry of $\hat{v}$ indexed by i
(branch)
Add $P \cap \{(x,v) \in \mathbb{R}^n \times \mathbb{R}^k : v_i \leq \lfloor \hat{v}_i \rfloor\}$ and $P \cap \{(x,v) \in \mathbb{R}^n \times \mathbb{R}^k : v_i \geq \lceil \hat{v}_i \rceil\}$ to L

Figure B.7: Flow chart of the branch-and-bound algorithm with optional improvements in striped blocks.

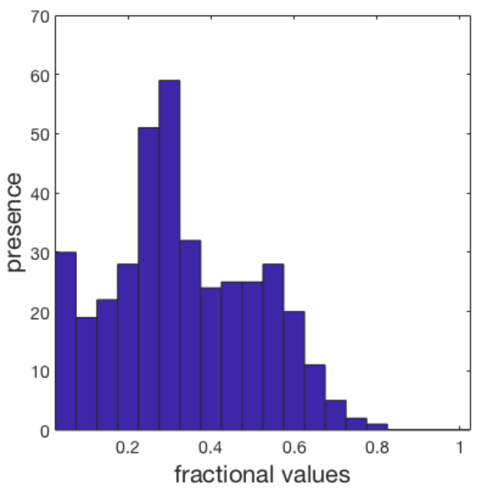
A good outline of strategies that can decrease the computational time of the branch-and-bound algorithm is given by Wolsey [54]. Combining these strategies has shown to be one of the current best methods to solve integer programs in practice. However, many problems that fit into the MILP framework are $\mathcal{NP}$-hard (Arora & Barak [5], Wolsey [54]), which means there might still be instances where the general strategy in this section can result in a very large computation time for some instances.

B.5 Improving the branch-and-bound algorithm

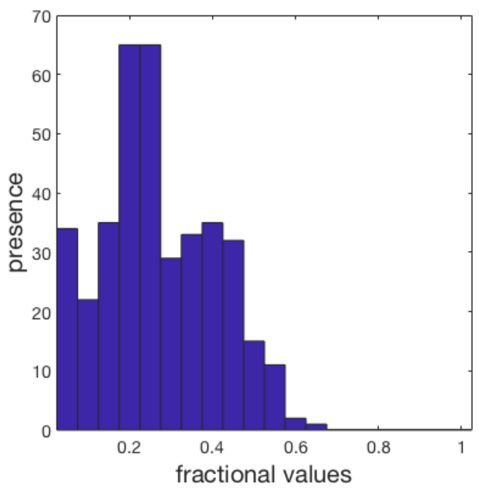
The branch-and-bound algorithm of section B.4 is a guaranteed way to find the global optimum of the MPC-SMPL problem. It is however very slow for large instances due to the complexity of the problem. Fortunately, there are plenty of ways to improve the practical running time of the branch and bound algorithm. As explained in section B.4, the most important improvements are the addition primal and dual bounds as explained in Wolsey [54]. Primal bounds are mainly found via heuristics, and dual bounds can be found by either improving the relaxation or by finding combinatorial lower bounds. How to find better bounds for the MPC-SMPL problem is explained in sections 7 and 8.
This section, will focus on other methods to improve the running time of the algorithm besides the improvement on bounds. The decision on which node of the list is to be evaluated first has an impact on the running time of the algorithm. It is also important to choose a good variable to branch on in the case a node is not pruned. The way an algorithm designer makes these decisions has a huge impact on the bounds of the global problem, and thus an effect on running time of the algorithm.

Branching rules

It is very likely that in the first iteration of the branch-and-bound algorithm the current node is to be branched instead of pruned (otherwise the problem is directly solved). It is also likely that many relaxed integer variables are fractional in the relaxed solution. This makes it hard to choose a good variable to branch on. It is usually a good idea to branch on the variable that is the most fractional, i.e. the variable v_i with the highest value $\min(\lceil v_i \rceil - v_i, v_i - \lfloor v_i \rfloor)$. The work of Gupta & Ravindran [28] is one among many to point out the benefits of this branching method. In the MPC-SMPL problem all variables are binary, so choosing the most fractional variable is equivalent to choosing the variable closest to 0.5. Intuitively, this feels like a good idea since because it represents splitting the problem on the most indecisive variable.
The solution to the relaxation of the MPC-SMPL problem is dependent on the value β, which might affect the work of the branching rule. Take for example an instance with $n = m = 5$, $N_p = 3$, $\mu_{\max} = 2$, $\lambda = 20$ and a single route for each job. The histograms of the relaxed values of the integer variables are shown in figure B.8.

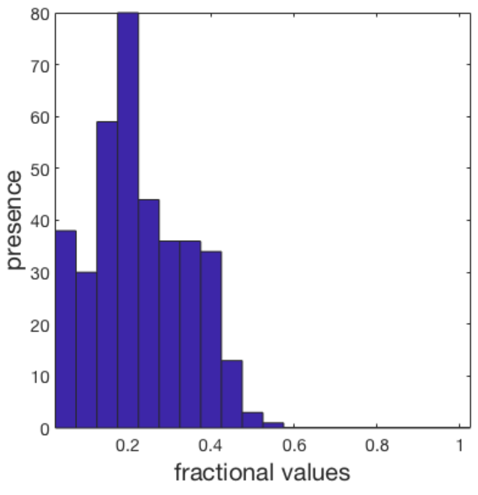


(a) $\beta = -80$

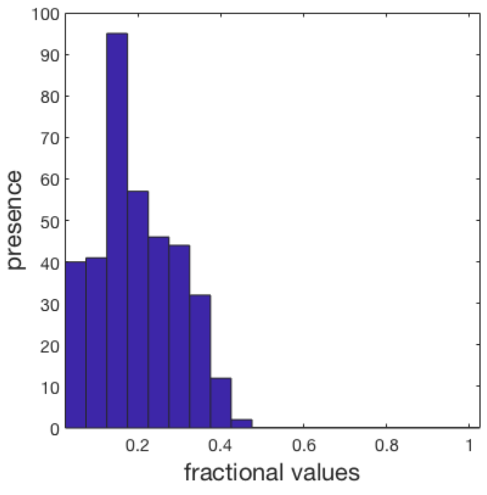


(b) $\beta = -100$

Figure B.8: Presence of fractional values from the relaxed optimal for 4 different values for β.



(c) $\beta = -120$



(d) $\beta = -140$

Figure B.8: Presence of fractional values from the relaxed optimal for 4 different values for β.

It can be seen from figure B.8 that the most fractional variable differs per value of β, while the feasibility of the problem does not change if β large enough in absolute value. It can be seen that most values are actually closer to 0 than to 1, and become even closer if β becomes larger in absolute value. Hence, it is not really clear which is the most indecisive variable. Although branching on the most fractional relaxed variable is usually a good strategy, for the MPC-SMPL problem it seems not a good choice.

A suggested alternative branching method is *strong branching* (Achterberg [2] et al.). Strong branching selects a set of variables and computes the relaxed optimum for both children for all these variable. Then, the variable with a child maximizing the relaxed optimal objective value is selected for branching. Computing many relaxed optimums costs time, so the set of evaluated integer variable candidates must be of a limited size. Suggested by Brucker et al. [12] is to evaluate branching on all ordering variables that belong to a single processor and cycle to keep the computational expenses low.

Search strategies

Another important aspect of the branch-and-bound algorithm is the search strategy. The search strategy determines which leaf of the branching tree are evaluated first. Linderoth [39] provides an overview of different search strategies. Most well known are *breadth-first* and *depth-first* searches. Breadth-first always selects a leaf closest to the root in the search tree, while depth-first selects a leaf with a maximum branches distant from the root problem. Since the MPC-SMPL problem has to be solved in a limited amount of time, finding good feasible solution early in the optimization process is essential. The depth-first strategy will therefore be a good choice. However, as stated in Linderoth & Savelsbergh [39], there are disadvantages for any strategies. It is best to choose a combination of strategies, and take advantage from the problem structure to come up with a good combined search strategy. This thesis focuses on solving larger instances of the MPC-SMPL problem by heuristics. Hence, the research for a good search strategy on the MPC-SMPL problem is left out, but could be an interesting topic for further research.

Advanced Pruning

In section 5.2 it is shown with the help of Theorem 5.4 on page 42 that only $\mathcal{O}(p^3)$ constraints are required to eliminate the circuits of ordering operations. Although this is not an exponential number of constraints, it is still quite a lot for large instances that can slow down the optimization process. Notice that the optimization process will never choose a solution containing a circuit, since in that case events would wait for each other for infinite time, and the objective function is certainly not optimal in that case.
It seems a good idea to prevent the algorithm from getting into branches that certainly contain circuits in ordering. To achieve this, suppose for each processor i the algorithm keeps track of a directed graph $G_i = (V_i, D_i)$ for each node in the branching tree. The vertices of V are the operations on processor i. There is an arc $(e_1, e_2) \in$ in D_i whenever the branches leading to the current node have set the ordering variables such that e_2 is processed after e_1 on i.

Example B.18. Let $O = \{1, 2, 3, 4\}$ be the set of operations that is to be scheduled on only one processor. Consider the model to be parameterized by the first method of parameterization for ordering variables. The algorithm keeps track of the graph $G = (O, D)$. The evolution of the graph according to the branch-and-bound search tree is illustrated in figure B.9.

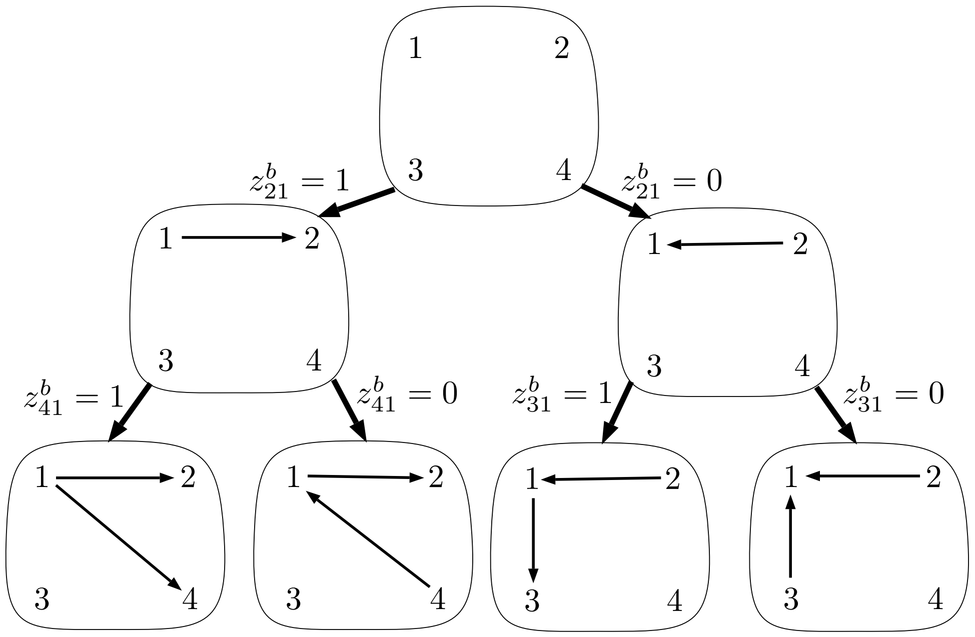


Figure B.9: The ordering graph in the branch-and-bound tree.

In order to prevent the branching from getting into situations with a circuit, determine for the current sub-problem if there are two arcs where the end vertex of one arc is the beginning vertex of the other arc. So look for arcs of the form (e_1, e_2) and (e_2, e_3). If it is the case that there are such two arcs, it can be immediately seen that e_3 has to be processed after e_1. In this case it can immediately be said that the binary variable that sets e_3 after e_1 can be determined in order to prevent a circuit in G_i. In addition, G_i can be updated with the arc (e_1, e_3). In example B.18 such a case occurs in the two middle sub-problems of the lowest layer. The problem update is shown in figure B.10.

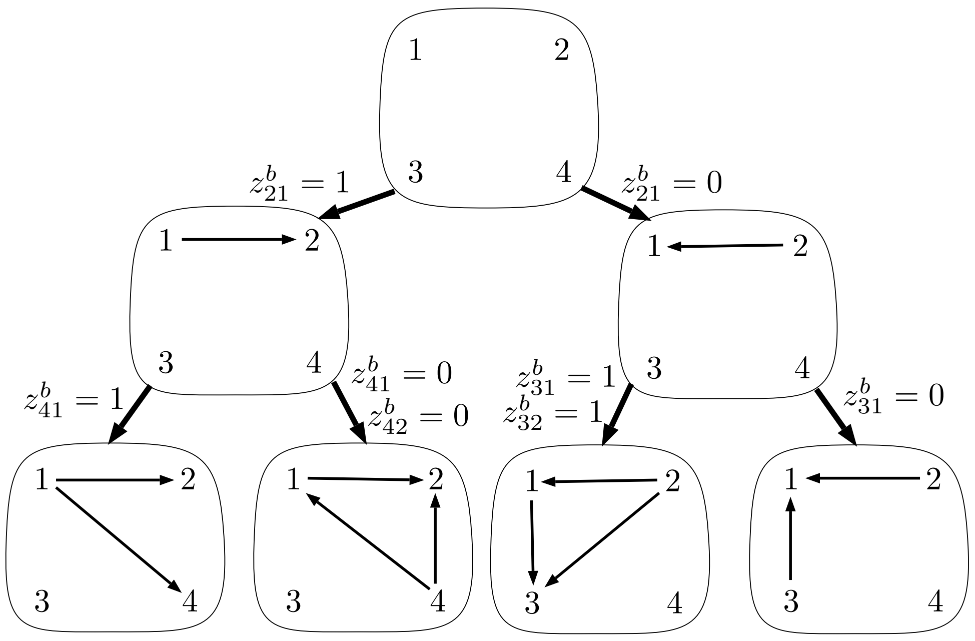


Figure B.10: The updated ordering graph in the branch-and-bound tree from example B.18

With this method only the necessary branching is computed. What is required is a method that finds any pair of successive arcs in G_i in a very small amount of time. It is not the most time efficient to simply test adjacency for all arcs in G_i. It is a better choice to keep track of all the new arcs in a sub-problem. Then for every new arc (e_1, e_2) test if e_1 has some incoming arc (e_3, e_1). If this is the case add (e_3, e_2) if it was not present already and update the binary ordering variables. The same goes for e_2, if (e_2, e_3) for some e_3 is an arc of G_i, add (e_1, e_3) and update the binary values.

C Terminology

Appendix C contains a few terms and definitions used throughout the report.

C.1 Definitions

Definition C.1. (Koopman & Sportiche [36]) *Let $f : X \to Y$ be a function.*

- *If $\forall x_1, x_2 \in X$ with $f(x_1) = f(x_2)$ follows that $x_1 = x_2$ then f is called injective.*
- *If $\forall y \in Y\ \exists x \in X$ such that $f(x) = y$ then f is called surjective.*
- *Function f is called bijective if it is injective and surjective.*

A bijection is also often referred as a *one-to-one mapping*, which means that if a function is bijective it is possible to take the inverse of that function.

Definition C.2. (Rice [47]) *The correlation coefficient of two stochastic variables X and Y is defined by*

$$\rho_{XY} := \frac{\mathrm{Cov}(X, Y)}{\sigma_X \sigma_Y}$$

where $\mathrm{Cov}(X, Y)$ is the co-variance between X and Y and σ_X is the standard deviation of X.

Let x_i and y_i for $i = 1, \ldots, n_e$ be n_e samples for stochastic variables respectively X and Y. Let $\bar{x} = \frac{1}{n_e} \sum_{i=1}^{n_e} x_i$. The following unbiased estimate for ρ_{XY} is given by Rice [47].

$$\hat{\rho}_{XY} = \frac{\sum_{i=1}^{n_e}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n_e}(x_i - \bar{x})^2 \sum_{i=1}^{n_e}(y_i - \bar{y})^2}}$$

Definition C.3. (Rice [47]) *A continuous random variable X is exponentially distributed with parameter $\theta > 0$ (denoted $X \sim Exp(\theta)$) if the density function is given by*

$$f_X(x) = \begin{cases} \theta e^{-\theta x} & \textit{if } x > 0 \\ 0 & \textit{otherwise} \end{cases}$$

C.2 Dispatching rules

A Dispatching rule decides which operation to schedule from the available jobs at a given moment in time. It moves up in time until a processor becomes idle or a job available to do the decision making in a continuous process. Dispatching rules barely cost time but rarely result in an optimal solution. The results of dispatching rules can often be improved using heuristics. There are several dispatching rules, each with advantages and disadvantages. Below one finds an overview of common dispatching rules which are only a small amount of dispatching rules as they are described in Pinedo [44], Chiang & Fu [14], Rajendran & Holthaus [46], Kaban et al. [32] and Dominic et al. [21].

FIFO (First in first out) The first job that came available will be scheduled first.

LPT (Longest processing time) The event with the longest processing time will be selected for scheduling.

SPT (Shortest processing time) The event with the shortest processing time will be selected for scheduling.

LTPT (Longest total processing time) The event from the job with the highest uncompleted total processing time is selected.

STPT (Shortest total processing time) The event from the job with the lowest uncompleted total processing time is selected.

LTRPOP (Longest remaining processing time on other processors) The event is scheduled that belongs to the job with the highest sum of processing times on processors that it did not complete yet.

STRPOP (Songest remaining processing time on other processors) The event is scheduled that belongs to the job with the lowest sum of processing times on processors that it did not complete yet.

EDD (Earliest due date) The event with the earliest due date is selected for scheduling.

SQNE (Shortest Queue at next event) Schedules the event that has the minimum processing time of available jobs on a processor that can be the next processor after completion of the selected even.

MS (Minimum slack) Selects the event belonging to the job j with the lowest value of $\max(d_j - t_j - \sum_i \text{remaining}\, p_{j,i}, 0)$.

ERD (Earliest release date) Schedule the job with the earliest release date first. WSPT (Weighted shortest processing time) Selects the event with the lowest value of $\frac{p_{j,i}}{\kappa_{j,i}}$

D Appendix for the Railway Network

Figure D.1 shows the interconnections of the railway network for trains from Amsterdam in the direction of The Hague. Table D.1 contains the corresponding platforms.

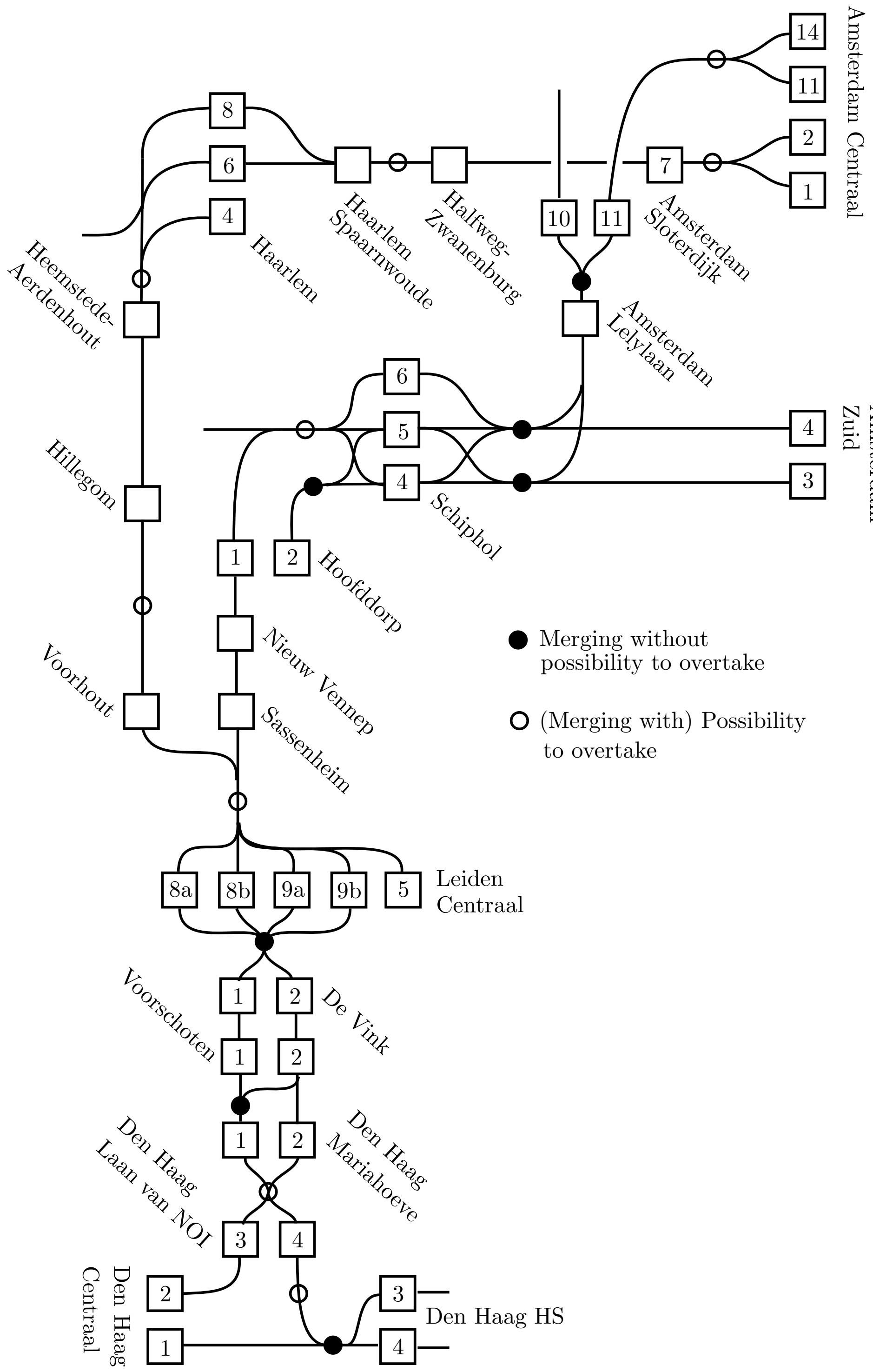


Figure D.1: An overview of the Railway Network for trains in the direction from Amsterdam to The Hague.

i	station (platform)	i	sation (platform)
1	Amsterdam Centraal (1)	22	Hoofddorp (1)
2	Amsterdam Centraal (2)	23	Hoofddorp (2)
3	Amsterdam Lelylaan	24	Nieuw Vennep
4	Amsterdam Centraal (11)	25	Sassenheim
5	Amsterdam Centraal (14)	26	Leiden Centraal (5)
6	Amsterdam Sloterdijk (7)	27	Leiden Centraal (8a)
7	Amsterdam Sloterdijk (10)	28	Leiden Centraal (8b)
8	Amsterdam Sloterdijk (11)	29	Leiden Centraal (9a)
9	Amsterdam Zuid (3)	30	Leiden Centraal (9b)
10	Amsterdam Zuid (4)	31	De Vink (1)
11	Halfweg-Zwanenburg	32	De Vink (2)
12	Haarlem Spaarnwoude	33	Voorschoten (1)
13	Haarlem (4)	34	Voorschoten (2)
14	Haarlem (6)	35	Den Haag Mariahoeve (1)
15	Haarlem (8)	36	Den Haag Mariahoeve (2)
16	Heemstede-Aerdenhout	37	Den Haag Laan van NOI (1)
17	Hillegom	38	Den Haag Laan van NOI (2)
18	Voorhout	39	Den Haag Centraal (1)
19	Schiphol (4)	40	Den Haag Centraal (2)
20	Schiphol (5)	41	Den Haag HS (3)
21	Schiphol (6)	42	Den Haag HS (4)

Table D.1: All the stations and platforms and their processor index.

Since the order on some railway tracks must be the same due to the absence of the possibility to overtake, some processors have common ordering variables. The processors are placed into the following groups which contain the same ordering variables, shown in table D.2.

group	processors	group	processors	group	processors	group	processors
1	1	9	16, 17	17	22	24	35
2	2	10	18	18	24, 25	25	36
3	3	11	7	19	23	26	37, 40
4	4	12	8	20	26	27	38
5	5	13	9	21	27, 28, 29, 30	28	39
6	6, 11	14	10	22	31, 33	29	41
7	12, 14, 15	15	19, 20	23	32, 34	30	42
8	13	16	21				

Table D.2: The groups of processors that share ordering variables.

D.1 Variable routing for the railway network

In this part of the Appendix the cycle value k has been left out for all variables for simplicity.

Haarlem

Platforms 6 and 8 at Haarlem correspond to respectively processors 14 and 15 in the model. The trains come from Haarlem Spaarnwoude, which is processor 12, and the trains depart to either Heemstede Aerdenhout (processor 16) or leave the network. The constraints for routing here are

$$a_{j,14} \geq d_{j,12} + p_{j,12,14} + \beta v^r_{j,r_1} \tag{D.1}$$
$$a_{j,15} \geq d_{j,12} + p_{j,12,15} + \beta(1 - v^r_{j,r_1}) \tag{D.2}$$
$$d_{j,14} \geq a_{j,14} + p_{j,14} + \beta v^r_{j,r_1} \tag{D.3}$$
$$d_{j,15} \geq a_{j,15} + p_{j,15} + \beta(1 - v^r_{j,r_1}) \tag{D.4}$$
$$a_{j,16} \geq d_{j,14} + p_{j,14,16} + \beta v^r_{j,r_1} \tag{D.5}$$
$$a_{j,16} \geq d_{j,15} + p_{j,15,16} + \beta(1 - v^r_{j,r_1}) \tag{D.6}$$

Constraints (D.5) and (D.6) can be left out if the train leaves the network. Not that Heemstede Aerdenhout is a stop for all trains and thus should there be no passing time variable for processor 16. Note that the given train stops at platform 6 if $v^r_{j,r_1} = 0$ and stops at plotform 8 otherwise.

Schiphol

Platforms 4, 5 and 6 on Schiphold correspond to respectively processors 19, 20 and 21 in the model. In the case that a train comes from Amsterdam Zuid platform 3, it can only stop at platforms 4 and 5. In this case constraints of the form inequalities (D.1)-(D.6) can be implemented with the alternative processors. If not coming from Amsterdam Zuid platform 3, but coming from platform i a train can also stop at platform 6. In this case, the routing constraints are

$$a_{j,19} \geq d_{j,i} + p_{j,i,19} + \beta(v^r_{j,r_1} + v^r_{j,r_2}) \tag{D.7}$$
$$a_{j,20} \geq d_{j,i} + p_{j,i,20} + \beta(1 - v^r_{j,r_1}) \tag{D.8}$$
$$a_{j,21} \geq d_{j,i} + p_{j,i,21} + \beta(1 - v^r_{j,r_2}) \tag{D.9}$$
$$d_{j,19} \geq a_{j,19} + p_{j,19} + \beta(v^r_{j,r_1} + v^r_{j,r_2}) \tag{D.10}$$
$$d_{j,20} \geq a_{j,20} + p_{j,20} + \beta(1 - v^r_{j,r_1}) \tag{D.11}$$
$$d_{j,21} \geq a_{j,21} + p_{j,21} + \beta(1 - v^r_{j,r_2}) \tag{D.12}$$

If the train is an ICD it leaves the network after schiphol and no further constraints have to be added. If the destination is Hoofddorp platform 2, the train can only arrive at that destination from Schiphol platform 4 and 5. Luckily, all the trains that do go there are coming from Amsterdam Zuid platform 3 so they always are on the right platforms at Schiphol. Again, for these trains the construction of constraints is similar to the constraints of routing via Haarlem in section D.1. In the other case, the trains travel via Hoofddorp platform 1, which is processor 22. If the train is an IC the remaining constraints are

$$t_{j,22} \geq d_{j,19} + p_{j,19,22} + \beta(v^r_{j,r_1} + v^r_{j,r_2}) \tag{D.13}$$
$$t_{j,22} \geq d_{j,20} + p_{j,20,22} + \beta(1 - v^r_{j,r_1}) \tag{D.14}$$
$$t_{j,22} \geq d_{j,21} + p_{j,21,22} + \beta(1 - v^r_{j,r_2}) \tag{D.15}$$

If the train is an SP the constraints are

$$a_{j,22} \geq d_{j,19} + p_{j,19,22} + \beta(v^r_{j,r_1} + v^r_{j,r_2}) \tag{D.16}$$
$$a_{j,22} \geq d_{j,20} + p_{j,20,22} + \beta(1 - v^r_{j,r_1}) \tag{D.17}$$
$$a_{j,22} \geq d_{j,21} + p_{j,21,22} + \beta(1 - v^r_{j,r_2}) \tag{D.18}$$

At Schiphol, the train now stops at platform 4 if $v^r_{j,r_1} = v^r_{j,r_2} = 0$, at platform 5 if $v^r_{j,r_1} = 1$ and at platform 6 if $v^r_{j,r_2} = 1$. The solution $v^r_{j,r_1} = v^r_{j,r_2} = 1$ must be excluded by setting

$$v^r_{j,r_1} + v^r_{j,r_2} \leq 1$$

Leiden Centraal

At Leiden Centraal, there are 4 platforms a train can stop if it is not the final destination. These platforms are 8a, 8b, 9a and 9b which are processors 27, 28, 29 and 30 respectively. Let i be the platform preceding Leiden Centraal. The routing constraints for a stop at Leiden Centraal are

$$a_{j,27} \geq d_{j,i} + p_{j,i,27} + \beta(v^r_{j,r_1} + v^r_{j,r_2}) \tag{D.19}$$
$$a_{j,28} \geq d_{j,i} + p_{j,i,28} + \beta(1 - v^r_{j,r_1} + v^r_{j,r_2}) \tag{D.20}$$
$$a_{j,29} \geq d_{j,i} + p_{j,i,29} + \beta(1 + v^r_{j,r_1} - v^r_{j,r_2}) \tag{D.21}$$
$$a_{j,30} \geq d_{j,i} + p_{j,i,30} + \beta(2 - v^r_{j,r_1} - v^r_{j,r_2}) \tag{D.22}$$
$$d_{j,27} \geq a_{j,27} + p_{j,27} + \beta(v^r_{j,r_1} + v^r_{j,r_2}) \tag{D.23}$$
$$d_{j,28} \geq a_{j,28} + p_{j,28} + \beta(1 - v^r_{j,r_1} + v^r_{j,r_2}) \tag{D.24}$$
$$d_{j,29} \geq a_{j,29} + p_{j,29} + \beta(1 + v^r_{j,r_1} - v^r_{j,r_2}) \tag{D.25}$$
$$d_{j,30} \geq a_{j,30} + p_{j,30} + \beta(2 - v^r_{j,r_1} - v^r_{j,r_2}) \tag{D.26}$$

It is now the case that in every binary combination of v^r_{j,r_1} and v^r_{j,r_2} there is exactly one platform where the constraints become not overwhelmed by β such that the constraint does not apply anymore. The further routing after Leiden Centraal is a bit more complicated, since it can choose multiple routes to follow directly after Leiden. The routing constraints for this part are given in in the following subsection.

Leiden - Den Haag

The section of the railway network from Leiden Centraal to Den Haag Laan van NOI contains 3 routes. On every station in between there are 2 tracks to follow. The train travels via platforms 1 or platforms 2 of the intermediate train stations. However, if traveling via the second platforms, the train has the option to switch to the trail of first platforms between Voorschoten en Den Haag Mariahoeve. To model routing for this part of the network, introduce binary variables v^r_{j,r_3} and v^r_{j,r_4}. A schematic drawing of the railway network between Leiden en Den Haag is shown in figure D.2. The values along the tracks denote the binary combinations of $(v^r_{j,r_3}, v^r_{j,r_4})$ that attain that part of the route. The presence of $\sim$ means it does not matter which variable is chosen.

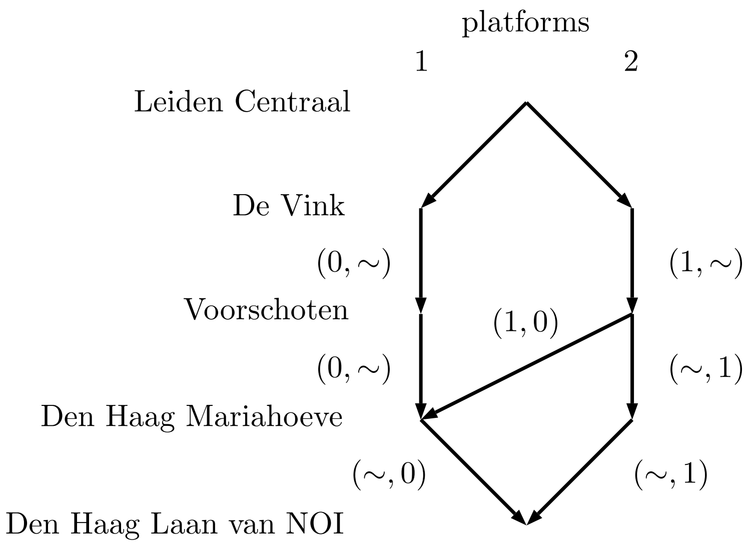


Figure D.2: The routes between Leiden Centraal and Den Haag Laan van NOI.

As mentioned, there are only 3 routes for this section of the network. Hence, there is one binary combination for $(v^r_{j,r_3}, v^r_{j,r_4})$ that has to be excluded, which is $(0, 1)$. This can be done by adding the following constraint.

$$v^r_{j,r_4} - v^r_{j,r_3} \leq 0$$

Let v^r_{j,r_1} and v^r_{j,r_2} still be the routing variables for the stop at Leiden Centraal. In case of a IC train, the routing part from Leiden Centraal to De Vink is modelled by constraints (D.27)-(D.34). The processor indices of the station platforms can be found in table D.1.

$$t_{j,31} \geq d_{j,27} + p_{j,27,31} + \beta(v^r_{j,r_1} + v^r_{j,r_2}) + \beta v^r_{j,r_3} \tag{D.27}$$
$$t_{j,31} \geq d_{j,28} + p_{j,28,31} + \beta(1 - v^r_{j,r_1} + v^r_{j,r_2}) + \beta v^r_{j,r_3} \tag{D.28}$$
$$t_{j,31} \geq d_{j,29} + p_{j,29,31} + \beta(1 + v^r_{j,r_1} - v^r_{j,r_2}) + \beta v^r_{j,r_3} \tag{D.29}$$
$$t_{j,31} \geq d_{j,30} + p_{j,30,31} + \beta(2 - v^r_{j,r_1} - v^r_{j,r_2}) + \beta v^r_{j,r_3} \tag{D.30}$$
$$t_{j,32} \geq d_{j,27} + p_{j,27,32} + \beta(v^r_{j,r_1} + v^r_{j,r_2}) + \beta(1 - v^r_{j,r_3}) \tag{D.31}$$
$$t_{j,32} \geq d_{j,28} + p_{j,28,32} + \beta(1 - v^r_{j,r_1} + v^r_{j,r_2}) + \beta(1 - v^r_{j,r_3}) \tag{D.32}$$
$$t_{j,32} \geq d_{j,29} + p_{j,29,32} + \beta(1 + v^r_{j,r_1} - v^r_{j,r_2}) + \beta(1 - v^r_{j,r_3}) \tag{D.33}$$
$$t_{j,32} \geq d_{j,30} + p_{j,30,32} + \beta(2 - v^r_{j,r_1} - v^r_{j,r_2}) + \beta(1 - v^r_{j,r_3}) \tag{D.34}$$

For the SP type of train, the routing constraints for this part of the railway network are given by (D.35)-(D.44).

$$a_{j,31} \geq d_{j,27} + p_{j,27,31} + \beta(v^r_{j,r_1} + v^r_{j,r_2}) + \beta v^r_{j,r_3} \tag{D.35}$$
$$a_{j,31} \geq d_{j,28} + p_{j,28,31} + \beta(1 - v^r_{j,r_1} + v^r_{j,r_2}) + \beta v^r_{j,r_3} \tag{D.36}$$
$$a_{j,31} \geq d_{j,29} + p_{j,29,31} + \beta(1 + v^r_{j,r_1} - v^r_{j,r_2}) + \beta v^r_{j,r_3} \tag{D.37}$$
$$a_{j,31} \geq d_{j,30} + p_{j,30,31} + \beta(2 - v^r_{j,r_1} - v^r_{j,r_2}) + \beta v^r_{j,r_3} \tag{D.38}$$
$$a_{j,32} \geq d_{j,27} + p_{j,27,32} + \beta(v^r_{j,r_1} + v^r_{j,r_2}) + \beta(1 - v^r_{j,r_3}) \tag{D.39}$$
$$a_{j,32} \geq d_{j,28} + p_{j,28,32} + \beta(1 - v^r_{j,r_1} + v^r_{j,r_2}) + \beta(1 - v^r_{j,r_3}) \tag{D.40}$$
$$a_{j,32} \geq d_{j,29} + p_{j,29,32} + \beta(1 + v^r_{j,r_1} - v^r_{j,r_2}) + \beta(1 - v^r_{j,r_3}) \tag{D.41}$$
$$a_{j,32} \geq d_{j,30} + p_{j,30,32} + \beta(2 - v^r_{j,r_1} - v^r_{j,r_2}) + \beta(1 - v^r_{j,r_3}) \tag{D.42}$$
$$d_{j,31} \geq a_{j,31} + p_{j,31} + \beta v^r_{j,r_3} \tag{D.43}$$
$$d_{j,32} \geq a_{j,32} + p_{j,32} + \beta(1 - v^r_{j,r_3}) \tag{D.44}$$

Now for the part from De Vink to Den Haag let i be the destination platform at Den Haag Laan van NOI. in the case of an IC train the constraints are given by (D.45)-(D.51).

$$t_{j,33} \geq t_{j,31} + p_{j,31,33} + \beta v^r_{j,r_3} \tag{D.45}$$
$$t_{j,34} \geq t_{j,32} + p_{j,32,34} + \beta(1 - v^r_{j,r_3}) \tag{D.46}$$
$$t_{j,35} \geq t_{j,33} + p_{j,33,35} + \beta v^r_{j,r_3} \tag{D.47}$$
$$t_{j,36} \geq t_{j,34} + p_{j,34,36} + \beta(1 - v^r_{j,r_4}) \tag{D.48}$$
$$t_{j,35} \geq t_{j,34} + p_{j,34,35} + \beta(1 - v^r_{j,r_3} + v^r_{j,r_4}) \tag{D.49}$$
$$a_{j,i} \geq t_{j,35} + p_{j,35,i} + \beta v^r_{j,r_4} \tag{D.50}$$
$$a_{j,i} \geq t_{j,36} + p_{j,36,i} + \beta(1 - v^r_{j,r_4}) \tag{D.51}$$

In case that the train type is SP, the constraints are given by (D.52)-(D.62).

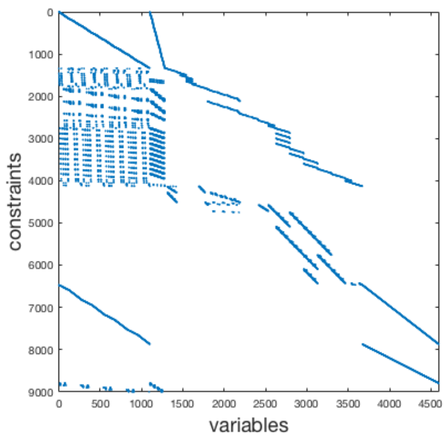
$$a_{j,33} \geq d_{j,31} + p_{j,31,33} + \beta v^r_{j,r_3} \tag{D.52}$$
$$a_{j,34} \geq d_{j,32} + p_{j,32,34} + \beta(1 - v^r_{j,r_3}) \tag{D.53}$$
$$d_{j,33} \geq a_{j,33} + p_{j,33} + \beta v^r_{j,r_3} \tag{D.54}$$
$$d_{j,34} \geq a_{j,34} + p_{j,34} + \beta(1 - v^r_{j,r_3}) \tag{D.55}$$
$$a_{j,35} \geq d_{j,33} + p_{j,33,35} + \beta v^r_{j,r_3} \tag{D.56}$$
$$a_{j,36} \geq d_{j,34} + p_{j,34,36} + \beta(1 - v^r_{j,r_4}) \tag{D.57}$$
$$a_{j,35} \geq d_{j,34} + p_{j,34,35} + \beta(1 - v^r_{j,r_3} + v^r_{j,r_4}) \tag{D.58}$$
$$d_{j,35} \geq a_{j,35} + p_{j,35} + \beta v^r_{j,r_4} \tag{D.59}$$
$$d_{j,36} \geq a_{j,36} + p_{j,36} + \beta(1 - v^r_{j,r_4}) \tag{D.60}$$
$$a_{j,i} \geq d_{j,35} + p_{j,35,i} + \beta v^r_{j,r_4} \tag{D.61}$$
$$a_{j,i} \geq d_{j,36} + p_{j,36,i} + \beta(1 - v^r_{j,r_4}) \tag{D.62}$$

Den Haag HS

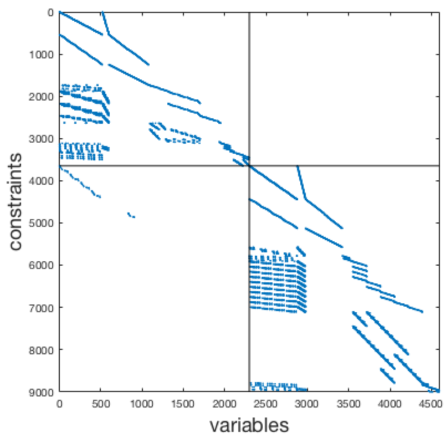
The routing model on station Den Haag HS is similar to the model of Haarlem, since there are 2 platforms. Since every train stopping at Den Haag HS leaves the network afterwards, constraints of the form of (D.5) and (D.6) can be left out.

D.2 Railway network partitions

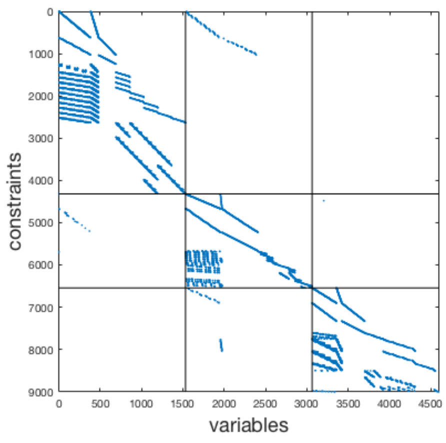
Figure D.3 shows the partitioned constraint matrices of the railway network modeled in section 4.1.



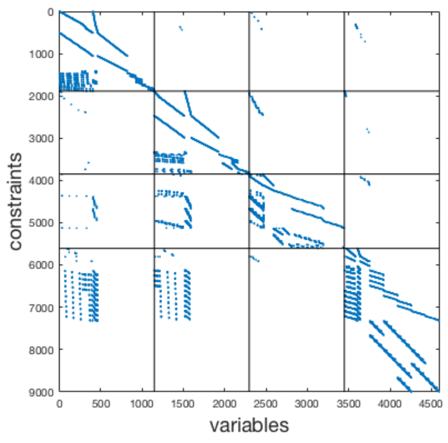
(a) no partition



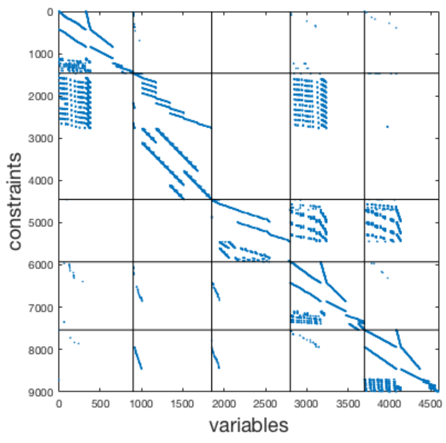
(b) 2 segments



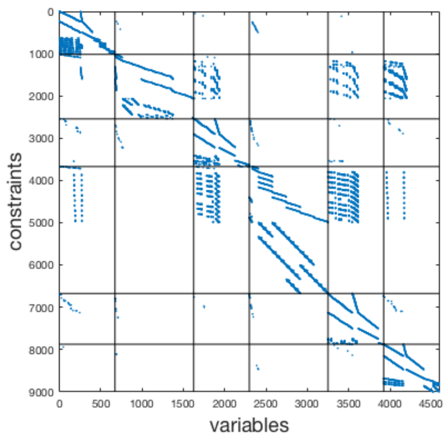
(c) 3 segments



(d) 4 segments



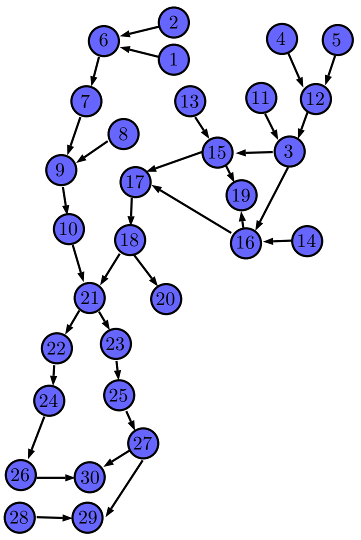
(e) 5 segments

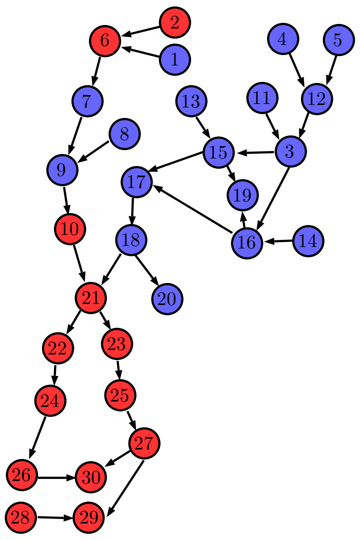


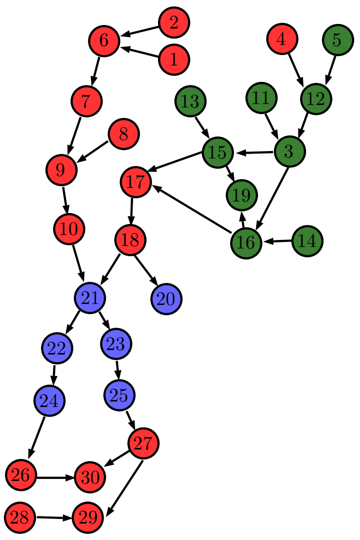
(f) 6 segments

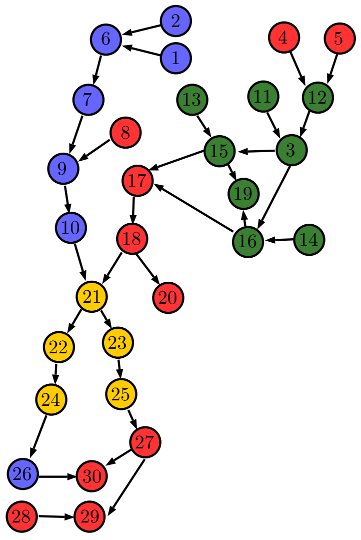
Figure D.3: The partitioned constraint matrices for the railway network.

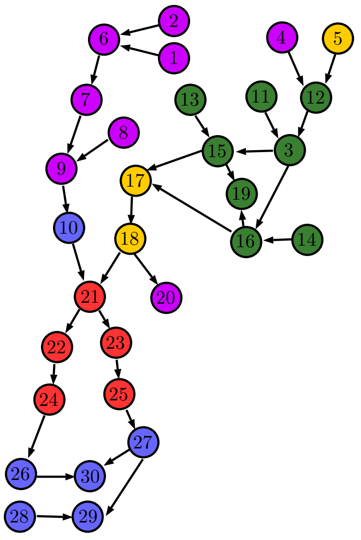
Notice that some of the diagonal blocks are not of equal size. As one can see in section 6.1, the partitioning algorithm prioritizes the minimization of the maximum difference in integer variables per segment. The number of constraints are allowed to differ quite a bit per segment. In integer optimization, the constraints have less effect on the computation time than the number of variables. Most desired is to split the integer variables equally over the segments. It is however very unlikely that this will result in equally sized segments with minimal interconnections.
In order to obtain minimal interconnections between the segments, the resulting partitioned solution will almost always have all the variables of one processor in the same segments.


(a) no partition


(b) 2 segments


(c) 3 segments


(d) 4 segments


(e) 5 segments

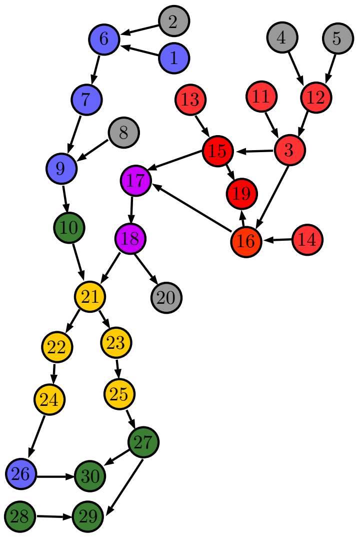

(f) 6 segments

Figure D.4: The partitioned graph of sets of ordering variables.

Figure D.4 shows the groups of processors from table D.2 partitioned into different partitions. Because there is not everywhere the possibility to overtake, some groups have always to be put into the same segment. Otherwise some integer variables end up outside the block-diagonal structure. Notice that therefore groups 3, 11, 12, 13, 14, 15, 16, 19 must always be partitioned to the same segment.

E Appendix for Results

The following figures represent the correlation between objective values of the starting points and the algorithm solution explained in section 10.1 page 90. The estimated correlation coefficients are present in table 10.1 on that same page.

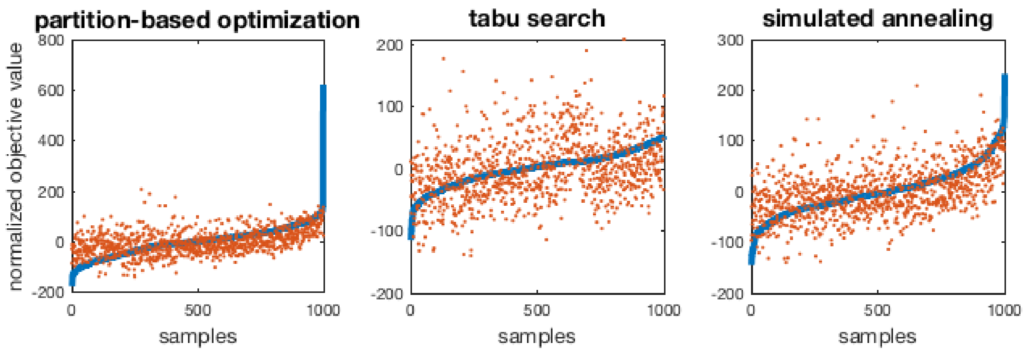


Figure E.1: Normalized objective value for starting points and algorithm outcome for $|L_j| = 1$.

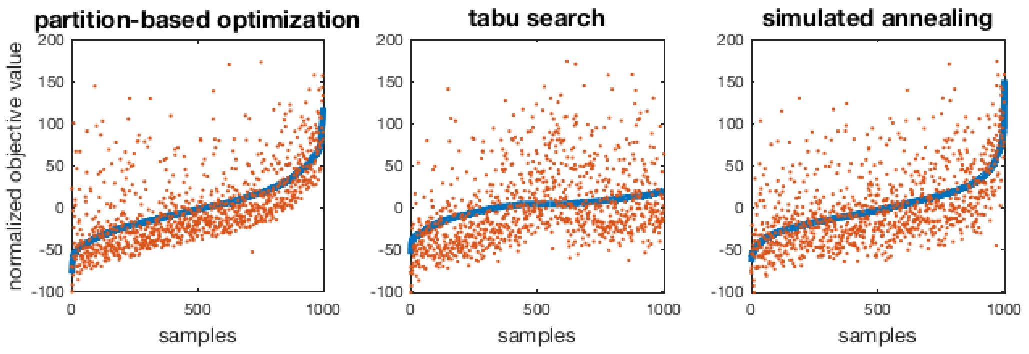


Figure E.2: Normalized objective value for starting points and algorithm outcome for $|L_j| = 3$.

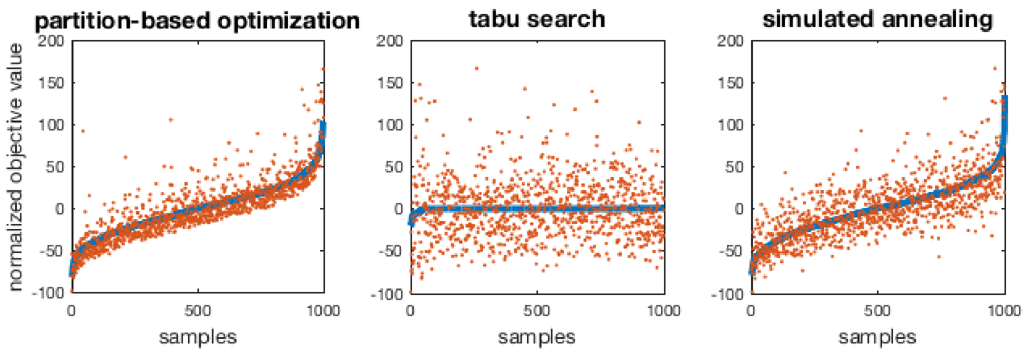


Figure E.3: Normalized objective value for starting points and algorithm outcome for $|L_j| = 5$.

E.1 Algorithm parameters

The following tables contain model and algorithm parameters for the results used in Part IV.

model	n	m	$\|R_j\|$	$\|L_j\|$	N_p	$\mu_{\max}$	λ	β
parameters	6	6	5	3	3	1	20	-50
partitioning	$p = 3$, $\alpha = 3$, $\theta = 0.5$							

Table E.1: Parameters for the partition of figure 9.4 page 87.

model	N_p	$\mu_{\max}$
parameters	3	1
partitioning	$\alpha = 3$, $\theta = 0.5$	

Table E.2: Parameters for the partition of figure 9.5 page 88.

model	$\|R_j\|$	$\|L_j\|$	N_p	$\mu_{\max}$	λ	β
parameters	m	3	2	1	$4.5n$	$-(\mu_{\max} + 1)\lambda$

Table E.3: Parameters of dual bound strength results in figure 9.6 page 89.

model	n	m	$\|R_j\|$	N_p	$\mu_{\max}$	λ	β
parameters	8	8	5,6,7	3	1	40	-100
partition-based optimization	$p = 4$, $\alpha = 3$, $\theta = 0.5$, $I_{\text{lim}} = 10^3$						
tabu search	$N_{\text{tabu}} = 50$, $I_{\text{bt}} = 15$						
simulated annealing	$T = 10$, $r = 0.9$, $I_Q = 10$						

Table E.4: Parameters for starting point correlation results in table 10.1 page 90 and figures E.1-E.3.

model	n	m	$\|R_j\|$	$\|L_j\|$	N_p	$\mu_{\max}$	λ	β
parameters	8	8	6,7	1	3	1	40	-100
partition-based optimization	$p = 4$, $\alpha = 3$, $\theta = 0.5$, $I_{\text{lim}} = 10^3$							
tabu search	$N_{\text{tabu}} = 50$, $I_{\text{bt}} = 15$, $I_{\text{re}} = 60$							
simulated annealing	$T = 10$, $r = 0.9$, $I_Q = 10$, $I_{\text{re}} = 30$							

Table E.5: Parameters of the main result in figure 10.1 page 91.

model	N_p	$\mu_{\max}$	β
parameters	4	1	-90
partitioning	$\alpha = 10$, $\theta = 0.5$		

Table E.6: Parameters for partitioning the railway network as in figures D.3 and D.4 pages 132 and 133.

model parameters	n	N_s	N_a	N_q	N_p	$\mu_{\max}$	λ	β
	12	2	4	4	2	1	25	-50
partition-based optimization	$p = 4$, $\alpha = 300$, $\theta = 0.5$, $I_{\text{lim}} = 10^3$							
tabu search	$N_{\text{tabu}} = 50$, $I_{\text{bt}} = 4$, $I_{\text{re}} = 7$							
simulated annealing	$T = 3$, $r = 0.75$, $I_Q = 10$, $I_{\text{re}} = 10$							

Table E.7: Parameters of the results for the container terminal in section 10.2 page 94.

F MATLAB Files

RoutingSearchTree.m

```
    function [I] = RoutingSearchTree(j1,j2,p,E,I)
    %   j1 and j2 denote the events, p the max event index.
    %   E is the set of numbers that has already been computed
    %   initialize by setting I=[], E=[];

    if j1>p || j2>p
        error('j1 and j2 must be smaller or equal to p')
    elseif j1==j2
        error('j1 cannot be equal to j2')
    elseif min(j1,j2)<1
        error('j1 and j2 must be positive integer numbers')
    elseif ((round(j1)==j1)&&(round(j2)==j2))==0
        error('j1 and j2 should be integer numbers')
    end

    if length(E)<p-1

        k=length(E);
        i=0;
        for h=1:k
            X=setdiff(1:p,E(1:(h-1)));
            x=find(X==E(h));
            i=i+factorial(p-h)*(x-1);
        end
        i=i:(i+factorial(p-k)-1);
        S=1:p;
        S=setdiff(S,E);
        n1=find(S==j1);
        n2=find(S==j2);
        In=i(factorial(p-k-1)*(n2-1)+1+factorial(p-k-2)*(n1-1):(factorial(p-k-1)
            *(n2-1))+factorial(p-k-2)*(n1));

        I=[I, In];
        for l=setdiff(S,[j1 j2])
            I=RoutingSearchTree(j1,j2,p,[E,l],I);
        end
    end

end
```

OrderSearchTree.m

```
    function [I] = OrderSearchTree(j1,j2,p,E,I)
    %   j1 and j2 denote the events, p the max event index.
    %   E is the set of numbers that has already been computed
    %   initialize by setting I=[], E=[];

    if j1>p || j2>p
        error('j1 and j2 must be smaller or equal to p')
    elseif j1==j2
        error('j1 cannot be equal to j2')
    elseif min(j1,j2)<1
        error('j1 and j2 must be positive integer numbers')
    elseif ((round(j1)==j1)&&(round(j2)==j2))==0
        error('j1 and j2 should be integer numbers')
    end

    if length(E)<p-1

        k=length(E);
        i=0;
        for h=1:k
            X=setdiff(1:p,E(1:(h-1)));
            x=find(X==E(h));
            i=i+factorial(p-h)*(x-1);
        end
        i=i:(i+factorial(p-k)-1);
        S=1:p;
        S=setdiff(S,E);
        n1=find(S==j1);
        n2=find(S==j2);
        In=i(factorial(p-k-1)*(n2-1)+1:(factorial(p-k-1)*(n2)));
        I=[I, In];
        for l=setdiff(S,[j1 j2])
            I=OrderSearchTree(j1,j2,p,[E,l],I);
        end
    end

end
```

Mathematical Notations

$\mathbb{R}$	The set of real numbers
$\mathbb{R}_+$	The set of positive real numbers
$\mathbb{N}$	The set of natural numbers
$\mathbb{N}_+$	$\mathbb{N}\backslash\{0\}$
$\mathbb{Z}$	The set of integer numbers
$\mathbb{Q}$	The set of rational numbers
$\mathbb{B}$	The set $\{0, 1\}$
∞	Infinity
$\cap$	Intersection of sets
$\cup$	Union of sets
$\backslash$	Relative set complement (for sets), or the boolean operator "exclude" (for boolean variables)
S^c	Complement of S
$\subseteq$	Subset or equal to
$\subset$	Subset but not equal to
$\times$	Cartesian product (for sets) or the cross product (for vectors)
$\emptyset$	The empty set
$\underline{n}$	The set $\{1, 2, \ldots, n\}$
$\mathbb{P}(S)$	The power set of S, the set of all subsets of S
$\forall$	For all
$\exists$	There exists
ϵ	Element $-\infty$
β	A very negative number explained on page 3.1 (in MILP) or the scheduling constraints (in scheduling)
$\mathbb{R}_\epsilon$	The set $\mathbb{R} \cup \{\epsilon\} = \mathbb{R} \cup \{-\infty\}$
$\mathbb{B}_\epsilon$	The set $\{\epsilon, 0\} = \{-\infty, 0\}$
$[A]_{ij}$	The element on the i-th row and j-th column of matrix A
A^T	Transpose of A
$\oplus$	The max-plus addition operator (section 1 page 7)
$\otimes$	The max-plus multiplication operator (section 1 page 7)
$\odot$	The max-plus Schur-product (section 1 page 8)
$\mathbb{R}_{\max}$	The semiring structure $(\mathbb{R}_\epsilon, \oplus, \otimes, \epsilon, 0)$ (page 7)
E_n	The $n \times n$ matrix with 0 on the diagonal and ϵ elsewhere
$\mathcal{E}_{nm}$	The $n \times m$ matrix with entries ϵ everywhere
$A^{\otimes n}$	The max-plus n-th power of A (section 1)
A^+	The max-plus power series $\bigoplus_{k\geq 1} A^{\otimes k}$ (section 1)
$A^\star$	The max-plus power series $\bigoplus_{k\geq 0} A^{\otimes k}$ (Section 1)
$\lambda(A)$	Eigenvalue(s) or cycle time of A
$\mathcal{G}(A)$	The communication graph of A (section 1 page 8)
$V(G)$	The set of vertices of graph G
$D(G)$	The set of arcs of graph G
$\log_n(x)$	Logarithm of x with base n
$\lfloor x \rfloor$	Largest integer value smaller or equal to x
$\lceil x \rceil$	Smallest integer value greater or equal to x
$\underline{0}$	A vector only containing zeros

$\mathbf{e}_i$	Unit vector with value 1 on i-th entry and 0 elsewhere
I	Conventional identity matrix
$\mathbb{1}^S$	Incidence vector of S
$\mathbb{2}_N(\cdot)$	Binary conversion from set to number (section 5 page 36)
$\text{conv}(X)$	The convex hull of X (Definition B.8 page 111)
$\dim(P)$	The dimension of polyhedron P (Definition B.7 page 111).
$\mathcal{O}(\cdot)$	The big O-notation (Definition B.16 page 114)
$\mathcal{NP}$	A complexity class explained in Appendix B.3
$\mathcal{P}$	A complexity class explained in Appendix B.3
$\bar{v}$	The max-plus binary complement
$\sim$	The boolean complement operator or "not"
$\vee$	The boolean operator "or"
$\wedge$	The boolean operator "and"
$\Rightarrow$	Implies statement
$\Leftrightarrow$	Equivalence
$\triangle$	Boolean operator for exclusive disjunction
N_p	The MPC prediction horizon (section 3.1 page 22)
$:=$	Assign value
$\mathcal{C}_j$	Completion time of job j. (section 2.3 page 20)
$\mathcal{C}_{\max}$	System makespan (section 2.3 page 20)
$\mathcal{L}_j$	Lateness of job j. (section 2.3 page 20)
$\mathcal{L}_{\max}$	Maximum lateness (section 2.3 page 20)
$\mathcal{T}_j$	Tardiness of job j. (section 2.3 page 20)
$\mathcal{T}_{\max}$	Maximum tardiness (section 2.3 page 20)
$E[\cdot]$	Expectation of stochastic value (Rice [47])

List of Acronyms

SMPL	switching max-plus linear (system) (section 2.1 page 15)
MPC	model predictive control (section 3.1 page 22)
MPC-SMPL	model predictive control for switching max-plus linear systems (Section 3.3 page 25)
LP	linear program (section B.1 page 109)
MILP	mixed-integer linear program (section B.1 page 110)
MBP	model-based partitioning (section 6)
FDI	facet defining inequality (Definition B.15 page 113)
TSP	traveling salesman problem (Applegate et al. [4])
CONCORDE	CONCORDE TSP solver (Cook [15])
GUROBI	GUROBI optimization software [29]
MATLAB	the MATLAB programming language
IC	intercity type train (section 4.1 page 26)
ICD	intercuty direct type train (section 4.1 page 26)
SP	sprinter type train (section 4.1 page 26)
QM	the Quinn-McClusky method (section 5.3)
ESPRESSO	The Espresso heuristic (Brayton [11])
R1	routing parameterization 1 (section 5.1 page 5.2)
R1QM	routing parameterization 1 (section 5.1 page 5.2) with Quine-McClusky (section 5.3)
R2	routing parameterization 2 (section 5.1 page 37)
R2QM	routing parameterization 2 (section 5.1 page 37) with Quine-McClusky (section 5.3)
R3	routing parameterization 3 (section 5.1 page 5.7)
R3QM	routing parameterization 3 (section 5.1 page 5.7) with Quine-McClusky (section 5.3)
O1	ordering parameterization 1 (section 5.2 page 42)
O2	ordering parameterization 2 (section 5.2 page 44)
O2QM	ordering parameterization 2 (section 5.2 page 44) with Quine-McClusky (section 5.3)

References

[1] Achterberg, T., Bixby, R.E., Gu, Z., Rothberg, E., Weninger, D. (2016). Presolve Reductions in Mixed Integer Programming. Zuse Institute Berlin Takustr. 7. D-14195 Berlin. ZIB-Report ISSN 1438-0064.

[2] Achterberg, T., Koch, T., Martin, A. (2005). Branching Rules Revisited. Operations Research Letters Volume 33, Issue 1.

[3] Akers, S. B. (1978). Binary Decision Diagrams. IEEE Transactions on Computers.

[4] Applegate, D.L., Bixby, R.E., Chvatal, V., Cook, W. J. (2011). The Traveling Salesman Problem: *A Computational Study.*

[5] Arora, S., Barak, B. (2007). Computational Complexity: *A Modern Approach.* Princeton University.

[6] Bai, D. Tang, L. (2013). Open Shop Scheduling Problem to Minimize Makespan with Release Dates.

[7] Baccelli, F., Cohen, G., Olsder, G. J., Quadrat, J. P. (1992). Synchronization and Linearity. Wiley, New York.

[8] Barvinok, A. (2002). A Course in Convexity, *Graduate Studies in Mathematics 54.* American Mathematical Society, Providence.

[9] Bolotashvili, G., Kovalev, M., Girlich, E. (1985). New Facets of the Linear Order Polytope.

[10] Bousfield, A.K. (1979). The Boolean Algebra of Spectra. Commentarii Mathematici Helvetici, Springer.

[11] Brayton, R.K., Hachtel, G.D., McMullen, C.T., Sangiovanni-Vincentelli, A.L. (1997). Logic Minimization Algorithms for VLSI Synthesis. Kluwer Academic Publishes.

[12] Brucker, P., Jurisch, B., Sievers, B. (1994). A Fast Branch & Bound Algorithm for the Job-shop Scheduling Problem. Discrete Appl. Math. 49, 107-127.

[13] Cassandras, C.G., Lafortune S. (2008). Introduction to Discrete event systems, *second edition.* Springer Science+Buisiness Media, LLC.

[14] Chiang, T.C., Fu, L.C. (2007). Using dispatching rules for job shop scheduling with due date-based objectives. International Journal of Production Research, Volume 45, Issue 14.

[15] Cook, W.J. (2018). Concorde TSP solver. <http://www.math.uwaterloo.ca/tsp/concorde/index.html>

[16] Dantzig, G.B., Orden, A., Wolfe, P. (1955). The Generalized Simplex Method For Minimizing a linear Form under Linear Inequality Restraints. Pacific Journal of Mathematics, Vol. 5, no. 2.

[17] De Schutter, B., De Moor, B. (1995). The Extended Linear Complementarity problem. Mathematical Programming, vol. 71, no. 3, pp. 289–325.

[18] De Schutter, B., De Moor, B. (1998). On the boolean minimal realization problem in the max-plus algebra. Proceedings of the 4th International Workshop on Discrete Event Systems (WODES'98). Cagliari, Italy, pp. 231–236.

[19] De Schutter, B. van den Boom A. J. J. (2000). Model predictive control for max-plus-linear discrete-event systems: *Extended report & Addendum.* Technical report bds:99-10a.

[20] Doignon, J.P., Rexhep, S. (2015). Primary Facets of Order Polytopes.

[21] Dominic, P.D.D., Kaliyamoorthy, S., Saravana Kumar, M. (2004) Efficient dispatching rules for dynamic job shop scheduling. Springer, Volume 24, Issue 1–2.

[22] Friedberg, H., Insel, A.J., Spence, L.E. (2014). Linear Algebra.

[23] Garcia, C.E., Prett, D.M., Morari, M. (1989). Model Predictive Control: *Theory and Practice - A Survey.* Automatica, Vol. 25, No. 3, pp. 335-348.

[24] Glover, F.W. (1989). Tabu Search Part I. ORSA Journal on Computing.

[25] Glover, F.W. (1990). Tabu Search Part II. ORSA Journal on Computing.

[26] Glover, F.W., Kochenberg, G.A. (2006). Handbook of Metaheuristics.

[27] Gonzalez, T., Sahni, T. (1976). Open Shop Scheduling to Minimize Finish Time. Journal of the ACM, Volume 23 Issue 4.

[28] Gupta, O.K., Ravindran, A. (1985). Branch and Bound Experiments in Convex Nonlinear Integer Programming. Management Science 31(12):1533-1546.

[29] Gurobi Optimization, LLC (2018). Gurobi Optimizer Reference Manual. <http://www.gurobi.com>.

[30] Heidergott, B., Olsder, G.J., van der Woude, J. (2006). Max Plus at Work, *Modeling and Analysis of Synchronized Systems: A Course on Max-Plus Algebra and its Applications.* Princeton Series in Applied Mathematics.

[31] Horn, R.A., (1990) The Hadamard Product. Proceedings of Symposia in Applied Mathematics, Volume 40.

[32] Kaban, A.K., Othman, Z., Rohmah, D. S. (2012). Comparison of Dispatching Rules in Job-shop Scheduling Problem Using Simulation: *A case study.* International Journal of Simulation Modelling.

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

[34] Kersbergen, B. (2015). Modeling and Control of Switching Max-Plus-Linear Systems, *Rescheduling of railway traffic and changing gaits in legged locomotion.*

[35] Kojima, M., Mizuno, S., Yoshise, A. (1989). A primal-dual interior point algorithm for linear programming. In N. Megiddo, editor, Progress in Mathematical Programming: *Interior Point and Related Methods*, pp. 29–47. Springer Verlag, New York.

[36] Koopman, H., Sportiche, D. (2009). Variables and the Bijection Principle.

[37] Leigh, J.R. (2004). Control Theory.

[38] Liaw, C.F. (1999). A tabu search algorithm for the open shop scheduling problem. Computers & Operations Research.

[39] Linderoth, J.T., Savelsbergh, M.W.P., (1999). A Computational Study of Search Strategies for MixedInteger Programming. INFORMS Journal on Computing, Vol. 11, No. 2.

[40] McCluskey, E.J. (1956). Minimization of Boolean Function. Bell Systems Technical Jour-nal, 35 (5):1417-1444.

[41] Nazareth, L., (1982). Numerical Behavior of LP Algorithms Based upon the Decomposition Principle. International Institute for Applied Systems Analysis A-2361 Laxenburg, Austria.

[42] Nederlandse Spoorwegen, (2019). Nederlandse Spoorwegen. <https://www.ns.nl/>.

[43] Petter (2017). minTruthtable <https://nl.mathworks.com/matlabcentral/fileexchange/37118-mintruthtable-tt-flags>, MATLAB Central File Exchange. Retrieved October 24, 2019.

[44] Pinedo, M.L. (2008). Scheduling, *Theory, Algorithms and Systems.* Springer.

[45] Quine, W.V. (1955). A Way to Simplify Truth Function. American Mathematical Monthly, 62(9):627-631.

[46] Rajendran, C., Holthaus, O. (1999). A comparative study of dispatching rules in dynamic flowshops and jobshops. European Journal of Operational Research. Volume 116, Issue 1.

[47] Rice, J.A. (2007). Mathematical Statistics and Data Analysis.

[48] Sporenplan, (2001). Sporenplan Online. <http://www.sporenplan.nl/ >.

[49] Van den Boom, A.J.J., De Schutter, B. (2016). MPC of implicit switching max-plus-linear discrete event systems – *Timing aspects.* Proceedings of the 8th International Work-shop on Discrete Event Systems (WODES'06), Ann Arbor, Michigan, pp. 457–462.

[50] Van den Boom, A.J.J., van den Muijsenberg, M., De Schutter, B. (2019). Model Predictive Scheduling of Semi-cyclic Discrete-event Systems using Switching Max-plus Linear Models and Dynamic Graphs.

[51] Van Laarhoven P.J.M., Aarts E.H.L. (1987). Simulated Annealing: *Theory and Applications.* Mathematics and Its Applications, vol 37. Springer, Dordrecht

[52] Williamson, D.F., Parker, R.A., Kendrik, J.S. (1989). The Box Plot: *A Simple Visual Method to Interpret Data.*

[53] Williamson, P.D., Hall, L.A., Hoogeveen, J.A., Hurkens, C.A.J., Lenstra, J.K., Sevasti-anov, S.V., Shmoys, D.B. (1997). Short shop schedules. Operations Research, 45:288–294.

[54] Wolsey, L.A. (1998). Integer Programming. Wiley-Interscience in Discrete Mathematics and Optimization.