The University of Nottingham
UNITED KINGDOM · CHINA · MALAYSIA

Groenemeyer, Sven (2012) Novel approaches to cyclic job-shop problems with transportation. PhD thesis, University of Nottingham.

Access from the University of Nottingham repository:
http://eprints.nottingham.ac.uk/13945/1/575075.pdf

Copyright and reuse:

The Nottingham ePrints service makes this work by researchers of the University of Nottingham available open access under the following conditions.

- Copyright and all moral rights to the version of the paper presented here belong to the individual author(s) and/or other copyright owners.
- To the extent reasonable and practicable the material made available in Nottingham ePrints has been checked for eligibility before being made available.
- Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way.
- Quotations or similar reproductions must be sufficiently acknowledged.

Please see our full end user licence at:
http://eprints.nottingham.ac.uk/end_user_agreement.pdf

A note on versions:

The version presented here may differ from the published version or from the version of record. If you wish to cite this item you are advised to consult the publisher's version. Please see the repository url above for details on accessing the published version and note that access may require a subscription.

For more information, please contact eprints@nottingham.ac.uk

Novel Approaches to Cyclic Job-Shop Problems with Transportation

Sven Groenemeyer, Dipl.-Math.

Thesis submitted to the University of Nottingham for the degree of

Doctor of Philosophy (PhD)

July 2012

Abstract

Scheduling problems can be found in almost any field of application in the real world. These problems may not only have different characteristics but they also imply more or less complex requirements. One specific class within this domain is the cyclic job-shop problem. It occurs in various areas reaching from industrial production planning down to the systems architecture of computers. With manufacturers in particular, one can find increasing demand for effective solution methods in order to tackle these scheduling problems efficiently.

This thesis will deal with the *Cyclic Job-Shop Problem with Blocking and Transportation.* It arises in modern manufacturing companies, where the products move automatically between the different workstations, for instance.

The problem itself is not new to the research community, but hardly any work has been done in solving it. Within this thesis we will try to close this gap and present some first approaches, discussing the structure of the problem and how it can be solved. As a result, we will provide three different solution methods, including an integer programming formulation, which is solved with a commercial solver, a branch and bound algorithm and a tabu search heuristic. All algorithms are tested on a range of data sets and compared with each other.

Additionally, we have worked on a polynomial solvable subproblem, which has gained more interest in the literature. As a result, a new polynomial algorithm, that outperforms the existing ones in theory as well as in empirical tests (except for some special cases) is presented.

This thesis concludes with a discussion about ideas of how to improve the presented methods and some other extensions to the investigated problem.

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

To my wife, Eva.

Acknowledgements

I would like to devote these last lines of my thesis to all the people who have supported, inspired and assisted me during the preparation of this work. First of all, I wish to thank my supervisors: Professor Peter Brucker for his experience, creativity, patience and persistence which gave a big contribution to this piece of research, and Professor Edmund Burke for providing me with the funding, advice and flexibility to pursue my own research goals.

During my years in Nottingham, I met a lot of new people and made many new friends. I am very grateful for their support which expressed itself in many different ways. Some of them, deserve a special thanks. Sam Allen, for being a great house mate, teacher and friend, especially. Nick Poxon, for his spontaneity and for providing a brilliant life outside of work. Daniel Soria, for being a really good friend who always makes you feel welcome. Jakub Mareček, for his sustained help and discussions about everything that I did not understand in the first place and of course his inspiring passion for photography.

Furthermore, I would also like to thank the ASAP research group, including the following people in no particular but "ASAP-style" order: Jason Atkin, Daniel Barthel, Elkin Castro, John Drake, Maria Franco, Juan Castro Gutiérrez, Matthew Hyde, Robert Oates, Andrew Parkes, Erwin Pesch, Stefan Ravizza, Pedro Rocha, Raymond Suen and Pawel Widera. It would

have been a lot harder without you all.

Obviously, I also had a lot of support back in Germany. First of all I want to thank my family, for their love and backup not just during my thesis but also life in general. They always supported me the best way they possibly could. Friedhelm Hinsenhofen, for his support, conveyance and challenges during the last 10 years.

My friends, René and Melanie Partmann, Lena and Christian Meyer, Heiner Dierkes and Rique Dierkes-Melzer, Jan Bufe, Stefan Wiebke, Nicole Wiesner, Andrea and Martin Voßeberg and Moni Sicking and Sven Sicking-Röttger for so many things that writing them all down would probably double the size of this work. Cheers guys!

Last but not least I am deeply grateful to my wife, Eva. Especially, for her patience, appreciation, believe in me and love during the last four years. This work would not have been accomplished without her.

BLANK PAGE
IN
ORIGINAL

Chapter 1

Introduction

1.1 Background and Motivation

Scheduling problems have been investigated over more than fifty years. A great interest for the study of these problems is their applicability to real world problems. Important applications of interest are machine scheduling and project planning. At the beginning of research in this area, the problem constraints were kept very simple. Early work in project planning, for instance, only considered scheduling situations with precedence constraints between activities, assuming that sufficient resources (machines, workers, space, material, etc.) to perform the activities were available. Over the years, more constraints have been introduced, which also coincided with the fact that modern computers could solve larger problems in time horizons that were realistic and useful in practice. So, the problem definition developed from a few simple constraints to more sophisticated and more complex problem definitions (cf. Brucker and Knust (2005)). Today, process automatisation and assembly line production are key factors for modern industry. It is, therefore, of great interest to have good, reliable, and flexible methods for supporting production planning. One problem that derives from those manufac-

turing environments is the cyclic job-shop problem with transportation and blocking (CJSPTB).

A practical example of where this problem arises is a modern furniture factory. It produces chairs, tables, wardrobes, and many other things. These objects are the jobs. Every job has to pass different machines through its production process (saws, drills, ploughs and varnishing stations). The processing of a job at one of these machines is called an operation. Obviously, these operations have to be executed in a specific order for every job and might take different times. We assume that the machines can only process one job at a time and that a machine has no space to store a job when it is finished (blocking). The transport of the jobs between the machines is done automatically by a single transport robot which can also only transport one job at a time. Since a factory is usually not producing just one item of each job, we consider a mass production environment, where every job has to be produced over and over again (cyclic). Note that this is just an example of the problem. A precise definition will be given in Chapter 2. As one can imagine, the CJSPTB is not just of great theoretical interest, but it also arises in many practical applications, which is one of the reasons why we have chosen it for this work.

1.2 Aims and Scope

The purpose of this dissertation is to provide a contribution for understanding and solving the cyclic job-shop problem with transportation and blocking. The problem is known to be NP-hard, which might be one of the reasons, why it has not been excessively studied. Also, the blocking situation makes it more difficult to find feasible solutions, since it significantly restricts the total number of them compared to the non-blocking case.

We discuss two main problems in this thesis. Firstly, we consider a description of fea-

sible solutions for the CJSPTB. In addition to that, different algorithms are presented to check if a given solution is feasible and to calculate the objective value. (We will see that this is not as straightforward as it is for other NP-hard problems.) Secondly, we look at different approaches to solve the general problem. Within this, we consider exact as well as heuristical methods. It is important for us not to just present theoretical results. We also apply all developed algorithms or solution methods to explicit problems of different sizes and compare their practical performances.

1.3 Contributions of this Thesis

In the following we summarise the contributions of the different sections within this work.

- Chapter 3
  - Section 3.1: An overview about the heights of different cyclic scheduling models, especially their interpretations and the theory of overlapping operations.
  - Section 3.2: The theory of blocking-feasible robotic cycles and a simple method to verify this.
  - Section 3.5: A new algorithm to solve the CJSPTB for a given robotic cycle, that (except for special cases) outperforms the existing ones in the literature theoretically (cf. Lemma 3.5.2) and based on experimental results.
  - Section 3.6: A computational comparison of three algorithms (including our own) solving the cyclic job-shop problem with transportation for a given robotic cycle on various data sets of different sizes.
- Chapter 4

- Section 4.1.1: An adaptation of a mixed integer programming model to our problem formulation of the CJSPTB.
- Section 4.1.2: A new mixed integer programming formulation for the CJSPTB based on overlapping operations.
- Section 4.2: An efficient method to construct feasible robotic cycles and an integration into a branch and bound procedure.
- Section 4.3: A new neighbourhood for the CJSPTB embedded in a tabu search heuristic.
- Section 4.4: A problem instance generator to create data sets for the CJSPTB of different sizes and properties.
- Section 4.5: A comparison of the experimental results on various data sets using all previously presented solution methods.

1.3.1 Publications Produced and Presentations Given based on the Work in this Thesis

The following work has been produced during the creation of this thesis. Although material from all of them has been assimilated into the thesis as a whole, they can be assigned as belonging particularly to the following specific following chapters.

Journal Publications and Submissions

- Section 3.5
  - Peter Brucker, Edmund K. Burke, and Sven Groenemeyer. A novel graph theoretical approach for cyclic job-shop problems. Submitted to *Annals of Operations Research*, September 2010.
- Section 4.1.2

  - Peter Brucker, Edmund K. Burke, and Sven Groenemeyer. A mixed integer programming model for the cyclic job-shop problem with transportation. *Discrete Applied Mathematics*, 2012.

- Section 4.2

  - Peter Brucker, Edmund K. Burke, and Sven Groenemeyer. A branch and bound algorithm for the cyclic job-shop problem with transportation. *Computers & Operations Research*, 2012.

Conference Publications and Talks

- Section 2.2

  - Peter Brucker, Edmund K. Burke, and Sven Groenemeyer. Cyclic job-shop problems with transport robots. *Student Conference on Operational Research, SCOR 2009, Lancaster*, March 2009.

- Section 3.5

  - Peter Brucker, Edmund K. Burke, and Sven Groenemeyer. A fast algorithm for the cyclic job-shop problem with one transport robot and blocking. *Student Conference on Operational Research, SCOR 2010, Nottingham*, April 2010.

- Section 4.2

  - Peter Brucker, Edmund K. Burke, and Sven Groenemeyer. A fast algorithm for the cyclic job-shop problem with one transport robot, blocking, setup times and a fixed robotic cycle. *The 12th International Workshop on Project Management and Scheduling, PMS 2010, Tours - Loire Valley, France*, April 2010.

Technical Reports and Manuals

- Section 4.4
  - Peter Brucker, Edmund K. Burke, and Sven Groenemeyer. Problem Generator: Generating problem instances for cyclic job-shop problem with transportation. University of Nottingham, November 2008, manual.

1.4 Structure of the Thesis

The main body of the thesis is separated into 4 chapters. Chapter 2 starts with a problem definition of the classical non-cyclic job-shop problem and some additional constraints. This should help everyone who does not have a broad background in job-shop scheduling to get a general impression and basic understanding about this area. From there, it builds up defining the main problem of the thesis. The cyclic job-shop problem with transportation and blocking (CJSPTB). Furthermore, we give a literature review and description of the previous work undertaken in the field.

In Chapter 3, we consider a specific polynomially solvable subproblem of the CJSPTB, where the robotic cycle is given in advance. We discuss different algorithms to solve this problem and present theoretical as well as empirical results.

Approaches to solve the general CJSPTB are presented in Chapter 4. Two mathematical programming models, a branch and bound procedure and a tabu search heuristic are discussed and experimental results are shown.

Finally, the thesis is concluded with some discussion in Chapter 5. All notations that are used within this work is summarised in a glossary at the end of this dissertation (cf. page 211).

Chapter 2

Problem Definition and Literature Review

Introduction

The classical job-shop problem is a well known combinatorial optimisation problem which (including its variations) has been widely studied by numerous authors over the last few decades. Within this chapter, we start describing the classical job-shop problem, followed by introducing additional constraints (time windows, blocking) that are also very common in the literature. Furthermore, a transport robot is added to the problem and leads to additional constraints. The first part of this chapter concludes with a literature review.

Even if the classical job-shop problem is not the major part of this thesis, it still builds the foundation to it. For that reason, we included it in this work.

In our opinion, it makes it easier to understand the second part (Section 2.2), in which we present the cyclic version of the job-shop problem. Especially readers with less expertise in cyclic (or even non-cyclic) scheduling problems should get a good impression and basic understanding about the problem.

The cyclic job-shop problem has many practical applications in the real world. We introduce different models from the literature and illustrate them with small examples. As in the non-cyclic case before, a transport robot is introduced to formulate the main problem of this thesis. The cyclic job-shop problem with transportation and blocking (cf. Section 2.2.5). We end the chapter with a literature review about cyclic scheduling problems and its variation.

For the reader's convenience, a glossary about all symbols, signs and variable names used within this work is added to the end of this thesis.

2.1 The Classical Job-Shop Problem

2.1.1 The Basic Model

The problem can be formulated as follows. We are given a set of N jobs $J_1, J_2, \ldots, J_N$. Each Job J_j consists of a set of n_j operations. There are different notations for the operations in the literature. One is that the i-th operation of job J_j is denoted by O_{ij} with $i \in \{1, \ldots, n_j\}$. Another simplified way, which will mostly be used in this work, is to consecutively number the operations in the form i with $i \in \{1, 2, \ldots, n\}$ and $n = \sum_{j=1}^{N} n_j$. This means that the operations $1, \ldots, n_1$ belong to job J_1, $n_1+1, \ldots, n_1+n_2$ belong to J_2, and so on. (We will occasionally refer to the first notation, if needed, in the case of ambiguity.) The set of all operations is denoted by Ω. Furthermore, let $J(i) \in \{J_1, \ldots, J_N\}$ be the job operation i belongs to. The operations of each job have to be processed in ascending order. For instance job J_1 has the processing order $1 \rightarrow 2 \rightarrow \ldots \rightarrow n_1$, J_2 has $n_1 + 1 \rightarrow n_1 + 2 \rightarrow \ldots \rightarrow n_2$ and so on. In general, it can be stated that the processing order of job J_j is $O_{1j} \rightarrow O_{2j} \rightarrow \ldots \rightarrow O_{n_j,j}$. These processing orders are called *precedence constraints*. Considering this order, let $pre(i)$ (respectively $suc(i)$) be the preceding (respectively succeeding) operation of i, in case it exists. Every operation has to be processed on one specified machine out of m machines $M_1, \ldots, M_m$. The machine, operation $i \in \Omega$ will be processed upon, is denoted by $M(i)$. If two succeeding operations of the same job are processed on the same machine, we simply combine those two operations into one. Each machine can only process one job at a time. Every operation i has a given processing time $p_i \geq 0$ for which it has to stay at machine $M(i)$ and we assume that preemption is not allowed (that means a job cannot stop its processing on one machine and continue later). For each operation $i \in \Omega$ let S_i be the starting time of operation i. Thus, for an operation

i and its direct successor $suc(i)$ the following has to hold:

$$S_i + p_i \leq S_{suc(i)} \tag{2.1}$$

for all $i \in \Omega$ where $suc(i) \in \Omega$ exists. These restrictions are called *precedence constraints.* In this first part, we assume that there is sufficient buffer space between the machines to temporarily store a job if it is finished on one machine and the succeeding machine is still occupied by another job. (In the next subsection we will introduce the so called blocking constraints, where machines do not have such a buffer. There is also the possibility to consider scenarios with limited buffers. However, the problem constraints will become a lot more complex and would exceed the scope of this thesis.) The time for a job to move from one machine to another or to a buffer zone is assumed to be included in the processing time. Thus, a job can continue its processing on the next machine immediately after finishing on the current machine. For two operations $i, j \in \Omega$ that have to be executed by the same machine $M(i) = M(j)$, one of the following machine constraints has to hold:

$$\begin{aligned} &\text{either } S_i + p_i \leq S_j \\ &\text{or } S_j + p_j \leq S_i. \end{aligned} \tag{2.2}$$

We call these constraints *machine constraints.* Note that for reentrant jobs, which means that the same job can visit the same machine several times, the machine constraints are dominated by the precedence constraints (2.1).

A common way of representing those problem is the *Disjunctive Graph Model* and this has been introduced in connection with the *Program Evaluation and Review Technique* (PERT). In this representation, we consider a graph $G = (V, E \cup D)$ where V is the set of nodes and E, D are sets of conjunctive and disjunctive arcs connecting the nodes. For the job-shop problem, there is a node for every operation and every arc in E

(respectively D) represents a precedence constraint (respectively machine constraint). Note that the set E consists of directed arcs and the set D of directed arc pairs. Formally, we define:

$$E = \{(i,j) \mid J(i) = J(j) \text{ and } suc(i) = j\},$$
$$D = \{(i,j),(j,i) \mid M(i) = M(j)\}.$$

To every arc $(i,j) \in E \cup D$, we assign a *length* which is equivalent to the processing time p_i of operation i. Additionally, there are two special *dummy operations* 0 and $\star$ indicating the start and the end of the schedule. The processing times of those operations is set to $p_0 = p_\star = 0$. To the graph, we add a directed arc of length 0 from the dummy start node to any other node without a predecessor (O_{1j} for $j = 1,\ldots,N$). From every node i without successor ($O_{n_j,j}$ for $j = 1,\ldots,N$), we set $suc(i) = \star$ and add a directed arc of length p_i to the dummy end node $\star$.

The basic scheduling decision now is to define an ordering of the operations processed on the same machines. This can be done by choosing an arc from each pair in D. Such a set of directed disjunctive arcs is called a selection $A \subseteq D$. A selection A is called complete if A contains exactly one arc of each pair out of D. Note that choosing one of every disjunctive arc pair is equivalent to fixing the order of the jobs processed on the same machine. The disjunctive graph model as a representation for the job shop problem was first proposed by Roy and Sussman (1964). A complete selection can be used to determine a feasible solution for the job-shop problem. Thereby, a solution is defined by a schedule S consisting of the starting times of all operations S_i with $i \in \Omega$. For a graph $G(A)$ representing a solution of a job-shop problem, the starting time S_i of each operation i is equivalent to the longest path from 0 to i. We call a schedule $S = (S_i)_{i=1}^n$ *feasible* if and only if constraints (2.1) and (2.2) hold and the corresponding graph $G(A)$ contains no cycle. We always assume that the starting time

of the dummy start operation is $S_0 = 0$. The objective of the job-shop problem is to find a feasible schedule that minimises the makespan $C_{\max} = \max_{i=1}^{n} C_i$, where $C_i = S_i + p_i$ is the completion time of operation i. In a given graph with a feasible selection, the makespan is equivalent to the length of the longest path from 0 to $\star$. For further explanations of and other results, see Shtub et al. (1994). The following example shows a problem and a corresponding feasible schedule.

Example 2.1.1. *Consider a job-shop problem with $N = 2$ jobs and $m = 3$ machines. Both jobs J_1 and J_2 consist of $n_1 = n_2 = 3$ operations and J_2 is a reentrant job. The processing times and the machine allocation are given in the following table.*

Job	J_1			J_2		
Operation	1	2	3	4	5	6
Processing time	3	5	2	3	8	2
Machine	M_2	M_3	M_1	M_1	M_2	M_1

It follows that the precedence constraints are

$$S_1 + 3 \leq S_2,\ S_2 + 5 \leq S_3,\ S_4 + 3 \leq S_5 \text{ and } S_5 + 8 \leq S_6. \tag{2.3}$$

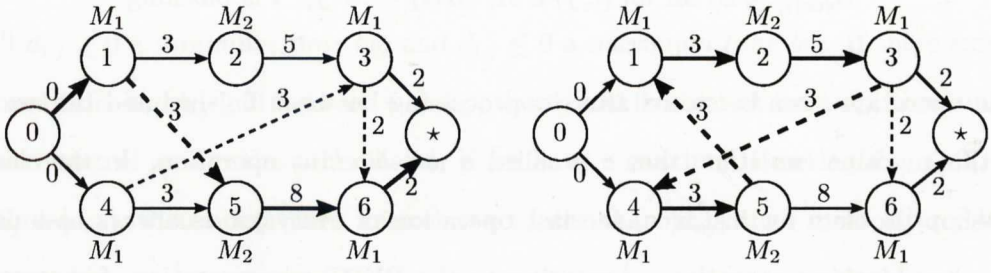
Constraints (2.2) *are of the following form*

$$M_1 : \begin{cases} \textit{either} & S_3 + 2 \leq S_4 & \textit{and} & S_4 + 3 \leq S_6 & \textit{and} & S_3 + 2 \leq S_6 \\ \textit{or} & S_4 + 3 \leq S_3 & \textit{and} & S_3 + 2 \leq S_6 & \textit{and} & S_4 + 3 \leq S_6 \\ \textit{or} & S_4 + 3 \leq S_6 & \textit{and} & S_6 + 2 \leq S_3 & \textit{and} & S_4 + 3 \leq S_3 \end{cases}$$

$$M_2 : \quad \textit{either} \quad S_1 + 3 \leq S_5 \quad \textit{or} \quad S_5 + 8 \leq S_1$$

Note that there are also constraints for the reentrant job J_2 in which the processing of operation 6 starts before the processing of operation 4 (e.g. $S_6 + 2 \leq S_4$) for machine M_1. However, those constraints are contradictory to the precedence constraints and therefore we exclude them in advance. Figure 2.1(a) *respectively* 2.1(b) *show a feasible*

respectively infeasible selection of the machine constraints. In the infeasible selection, there is a cycle $1 \to 2 \to 3 \to 4 \to 5 \to 1$ *with length* 21 *that is highlighted by bold arcs. For choosing* $S_4 + 3 \leq S_3$, $S_3 + 2 \leq S_6$ *and* $S_1 + 3 \leq S_5$ *a feasible schedule is presented*



(a) Feasible machine sequence

(b) Infeasible machine sequence

Figure 2.1: Directed graphs for Example 2.1.1

in Figure 2.2. *It is optimal for this problem with* $C_{\max} = 13$.

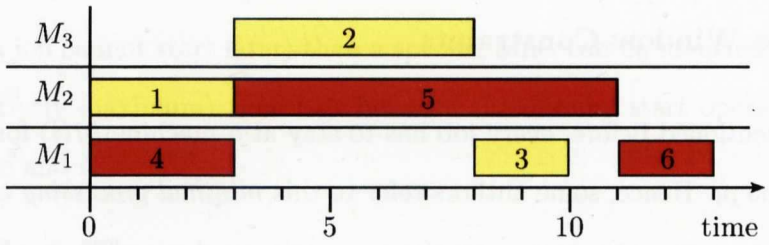


Figure 2.2: Schedule for Example 2.1.1

2.1.2 Blocking Constraints

Assume that there is no buffer to store a job after it has finished its processing on a machine and the next machine is still occupied by another job. Then the job has to remain on the current machine and blocks it until the next machine becomes available. We call such an operation of this job a *blocking operation.* Obviously, blocking operations may delay the start of successive operations on the same machine. Let us consider two operations i, j which have to be processed on the same machine and assume that i is a blocking operation. If operation i is going to be processed first on $M(i)$, then

$suc(i)$ has to start at the same time or before j can start. (Otherwise $M(i)$ would still be blocked by i.) Hence, we get the relation:

$$S_{suc(i)} \leq S_j \text{ for all } (i,j) \in A,\ M(i) = M(j),\ i \text{ is blocking.} \tag{2.4}$$

If an operation i can be stored after its processing has been finished and the next job on this machine can start, then i is called a *non-blocking* operation. In the classical job-shop problem for instance, the last operation of every job is always assumed to be a non-blocking operation. According to the PERT representation, for every two blocking operations i, j that have to be processed on the same machine $M(i) = M(j)$ we introduce two directed *alternative* arcs $(suc(i), j)$ and $(suc(j), i)$. In a feasible solution, one of these arcs has to be chosen, such that no cycle exists in the corresponding graph.

2.1.3 Time Window Constraints

As we have mentioned before, every job has to stay at a machine $M(i)$ for at least its processing time p_i. Hence, some authors refer to this *minimal processing time* as $p_i^{\min}$. Furthermore, one can introduce a *maximum processing time* $p_i^{\max} \geq p_i^{\min}$, defining the time job $J(i)$ can stay at longest at machine $M(i)$. Note that this maximum processing time only makes sense, if there is no or only a limited buffer at machine $M(i)$ to store operations that have finished their processing at this machine. In practice, those minimum and maximum processing times are used, for instance, in circuit board printing, where the job has to pass through different acid baths. It needs to stay in a bath for a minimum time, but cannot exceed a given duration, because this could ruin the board.

Those *processing time windows* can be generalised to *minimal and maximal time lags* between the starting times of two operations. Therefore, let $i, j \in \{1, ..., n\}$ be two operations with $i \neq j$ and S_i respectively S_j their starting times. We have a *time lag*

$d_{i,j}$ between S_i and S_j if

$$S_i + d_{i,j} \leq S_j. \tag{2.5}$$

We call $d_{i,j} \geq 0$ a *minimum time lag* and $d_{i,j} \leq 0$ a *maximum time lag.* If there exists a minimal time lag $d_{i,j} \geq 0$ between i and j, then j has to start at least $d_{i,j}$ time units after the start of i. On the other hand, if we have a maximal time lag $d_{i,j} \leq 0$, i.e $S_i \leq S_j + |d_{i,j}|$, then operation i cannot start later than $|d_{i,j}|$ time units after j has been started.

By using minimum and maximum time lags, several restrictions can be added to the general job-shop problem, for example:

1. Release times, where an operation i cannot start earlier (respectively deadlines where a job cannot start later) than a specific time, can be modeled by minimum (respectively maximum) time lags between the dummy start operation 0 of the schedule and i.

2. If operation i has to start exactly at time T, we introduce minimum and maximum time lags such that $d_{0,j} = -d_{j,0} = T$.

3. The so called *no-wait constraint* implies that operation j has to be carried out directly and without delay after activity i, so $S_j - S_i = p_i$. Therefore we introduce minimum and maximum time lags such that $d_{i,j} = -d_{j,i} = p_i$.

4. As mentioned before, if an operation i has a minimum and a maximum duration of processing, then the processing time of i is bounded by a time window, i.e. $p_i \in [p_i^{\min}, p_i^{\max}]$. That means, the earliest finishing point in time of i is $S_i + p_i^{\min}$ and the latest is $S_i + p_i^{\max}$. Hence, for an operation j which follows i we have the time lags $d_{i,j} := p_i^{\min}$ and $d_{j,i} := -p_i^{\max}$.

Those time lags can be represented in our current PERT graph as follows. As before, the nodes V in Graph $G = (V, E \cup A)$ are representing the start of an activity. For every minimum time lag $d_{i,j} \geq 0$, we introduce an arc (i, j) of length $d_{i,j}$. On the other hand, for every maximum time lag $d_{i,j} \leq 0$ we introduce an arc (j, i) of lengths $-d_{i,j}$. Note that, if in (2.5) we set $d_{i,j} = p_i$ we get the same constraints as in (2.1) and (2.2).

2.1.4 Transportation Constraints

So far, we have assumed that, at the time a job has finished its processing on a machine, it moves automatically and without any delay directly to the next machine (in the blocking case, only if this machine is free). However, in many industrial production lines this move is done by a *transport robot* (or automated guided vehicle). This means that after a job has finished its processing on a machine, a robot unloads the job, transports it to the next machine and loads the job onto that machine. After a job has been transported to, and loaded onto the machine, the robot either stays at this machine until the job is finished and unloads it, or moves empty to another machine. Of course, the robot needs some time to perform these tasks, which has to be taken into account. Within this work, we restrict ourselves to a *single* transport robot. Scenarios where multiple robots are allowed have additional problem constraints. For instance, physical collisions of two robots needs to be avoided. Especially for practical purposes this would be an interesting point. To provide a fundamental understanding and a variety of different approaches for a problem within this thesis, we chose the single robot scenario. However, the multiple robot problem is definitely an interesting topic and as we will see in the literature review, the problem has already been investigated.

For an operation $i \in \Omega$ the single robot has to perform the following tasks:

- *unloading* job $J(i)$ from machine $M(pre(i))$ (if the processing is not finished the robot will wait);

- *transporting* the job from $M(pre(i))$ to $M(i)$;
- *loading* job $J(i)$ onto $M(i)$.

These operations are called the *transport move* of operation i which is denoted by τ_i and the time it takes to perform such a move by t_i. We will refer to t_i as the *transportation time* of operation i. After a job has been loaded onto a machine the robot either stays at this machine or drives empty to another one. An unloaded drive is called an *empty move*. The time needed by the robot to move from machine $M(i)$ to $M(j)$ is denoted by e_{ij}. Note that a sequence of transport moves indirectly induces necessary empty moves after loading a machine and unloading the succeeding one. Hence if, for example, in a sequence of robot moves τ_j follows τ_i then after loading $J(i)$ on $M(i)$ the robot drives empty to machine $M(pre(j))$ (or stays at $M(i)$ if $M(i) = M(pre(j))$).

Since the robot also needs to transport the finished jobs away from their last machine to the output station, we add a dummy end activity $\star^j$ for every job $J_j \in \{J_1, \ldots, J_N\}$ and set the direct successor of the last operation of $O_{n_j,j}$ of J_j to $\star^j$. Furthermore, we define $\Omega^\star := \Omega \cup \{\star^1, \ldots, \star^N\}$ as the set of all operations including the dummy end operations and set $p_i = 0$ for all $i \in \{\star^1, \ldots, \star^N\}$.

We make the reasonable assumption that the triangle inequality $e_{ij} + e_{jk} \geq e_{ik}$ holds for the empty moving times between any three machines $M(i), M(j)$ and $M(k)$. We assume that, after an empty move, a transport operation always follows, and that for all empty moves the triangle inequality

$$e_{ik} + e_{kl} \geq e_{il} \text{ for all } i, k, \Omega^\star \cup \{0\} \tag{2.6}$$

holds. This means that the direct way between two machines is at least as short as the detour through a third machine. Otherwise, the robot always takes the shorter way through the third machine and we set $e_{il} = e_{ik} + e_{kl}$. We also assume that $e_{ij} = 0$ for

$M(i) = M(j)$ and the empty moving time from $M(i)$ to $M(j)$ is the same as from $M(j)$ to $M(i)$, so $e_{ij} = e_{ji}$. Note that a transport move and an empty move between the same two machines do not need to have the same duration. Since a transport time t_i also includes the loading and unloading process of the job it usually holds that $t_i > e_{pre(i),i}$.

Furthermore, we define a sequence of all necessary transport moves τ_i as a *robot route* R. That means a robot route is of the form

$$R = (\tau_{\sigma(1)}, \tau_{\sigma(2)}, \ldots, \tau_{\sigma(n+N)}), \tag{2.7}$$

where $\sigma : \Omega^\star \to \Omega^\star$ is permutation of the set of all operations. It is also required that in a robot route R all transport moves concerning the same job are in ascending order. This means that τ_i appears before $\tau_{suc(i)}$ in R. (Otherwise, the precedence constraints would be violated.)

In the following, we update the previous model and include those robot operations. We distinguish between two different cases: in the first case we investigate the problem without blocking constraints. We assume that a machine M_i has a buffer B_i that is large enough to store jobs that have either finished their processing, or have been delivered by the robot. This means that, on the one hand, the jobs that the robot loads onto a machine are stored in this buffer, and the machine can take a job automatically without any delay from this buffer. On the other hand, if the processing of a job is finished, then the job will be stored without any delay in the buffer, and the robot unloads the jobs from there. After a job has been stored in the buffer, the machine can start the processing of the next job.

In the second case, we assume that a machine has no buffer. This means, that after finishing its processing on the machine a job has to stay there, until it is unloaded by the robot. During this stay, the machine is blocked and not available for processing any other job.

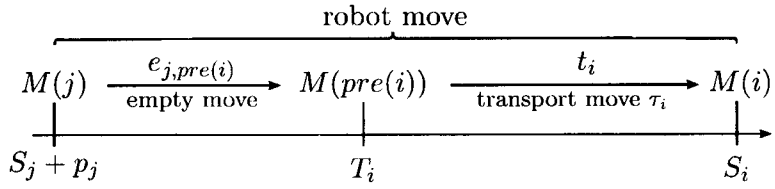


Figure 2.3: Illustration of a robot move

Problems without Blocking

In addition to the starting time S_i of operation i, we denote the staring time of the transport move τ_i with T_i, which is the point in time when the robot starts unloading job $J(i)$ off machine $M(pre(i))$. Figure 2.3 provides an illustration of a general robot move.

For a feasible schedule $S = (S_i, T_j)_{i,j \in \Omega^\star}$ there are several constraints which have to be satisfied. First, we have to guarantee that operation i cannot be unloaded from machine $M(i)$ until its processing time p_i is over:

$$S_i + p_i \leq T_{suc(i)} \text{ for all } i \in \Omega. \tag{2.8}$$

The next constraint ensures that operation i cannot be processed on $M(i)$ before the robot has unloaded $J(i)$ from the previous machine $M(pre(i))$, transported it to $M(i)$ and loaded $J(i)$ onto $M(i)$:

$$T_i + t_i \leq S_i \text{ for all } i \in \Omega^\star. \tag{2.9}$$

After unloading a job from a machine, the robot must have sufficient time to transport the job to the next machine, load it onto that machine and travel empty to another machine, before it can start unloading the job there. For two operations $i, j \in \Omega^\star$ with $i \neq j$ one of the following constraints has to be satisfied, depending on which transport

move will be processed first:

$$\begin{aligned} T_i + t_i + e_{i,pre(j)} &\leq T_j \\ \text{or } T_j + t_j + e_{j,pre(i)} &\leq T_i. \end{aligned} \tag{2.10}$$

For two jobs which have to be processed on the same machine we have to determine an order in which the jobs will be processed. Hence, one of the following constraints must hold.

$$\begin{aligned} S_i + p_i &\leq S_j \\ \text{or } S_j + p_j &\leq S_i, \end{aligned} \tag{2.11}$$

for $i, j \in \Omega$ with $i \neq j$ and $M(i) = M(j)$.

In the next step, we will introduce a graph representation of this model. Let $G = (V, E \cup A)$ be a directed graph where V is the set of nodes and $E \cup A$ is the set of arcs. As in our previous model, for every $i \in \Omega^\star$ we add a node to V, which indicates the start of processing operation i. Furthermore, for every $i \in \Omega^\star$ we add a node τ_i to V which denotes the start of a transport move. For every precedence constraint defined by (2.8) we get an arc from i to $\tau_{suc(i)}$ of length p_i. Additionally, for every constraint given by (2.9), we get an arc from τ_i to i of length t_i. These arcs together build the set E. The constraints defined by (2.10) are dependent on the robot route. If the robot has loaded a job onto a machine, then it either has to wait for this job at the machine or drive empty to another machine. In the first case, the next operation would be the transport move of the same job after its processing has finished on the machine, while in the second case, the robot would perform an empty move. For each empty move according to (2.10), we introduce two disjunctive arcs. One leads from τ_i to τ_j of length $t_i + e_{i,pre(j)}$ and the other one from τ_j to τ_i of length $t_j + e_{j,pre(i)}$. Furthermore, there are

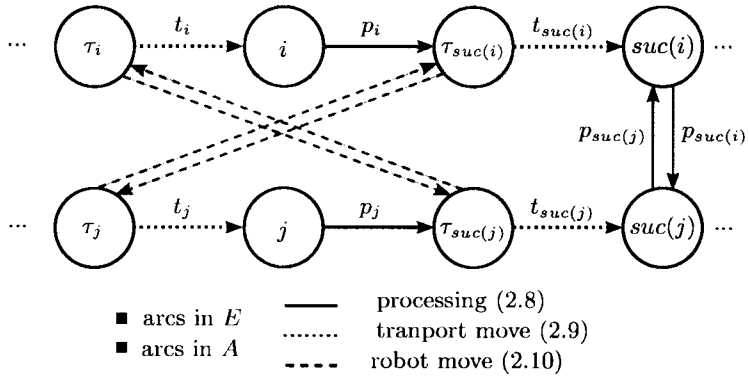


Figure 2.4: Example for disjunctive arcs

alternative arcs between the operations that have to be processed on the same machine. For each pair $i, j \in \Omega$ with $i \neq j$ and $M(i) = M(j)$, we add an arc from i to j of length p_i and an arc from j to i of length p_j. In a feasible solution, one arc of each disjunctive arc pair has to be chosen. These arcs build the set A.

Figure 2.4 represents an example of a partial graph with $M(suc(i)) = M(suc(j))$ and $M(i) \neq M(j)$.

Additionally, we add one general source node 0 and sink node $\star$. An arc leads from the source node to every first transport operation of each job of length 0 and from the last operation $\star^j$ of each job J_j there exists an arc to the sink node $\star$ of length 0.

Example 2.1.2. *Consider the following data of a job-shop problem with three jobs and three machine.*

Job	J_1			J_2		J_3	
Operation	1	2	3	4	5	6	7
Processing time	13	7	4	4	9	3	9
Machine	M_1	M_2	M_3	M_1	M_3	M_1	M_2

Additionally, there is one transport robot which transports the jobs between the machines. The transportation time the robot needs to transport a job from one machine to another is given by $t_i = 2$ *for all* $i = 1, \ldots, 7, \star^1, \star^2, \star^3$. *We assume that the empty*

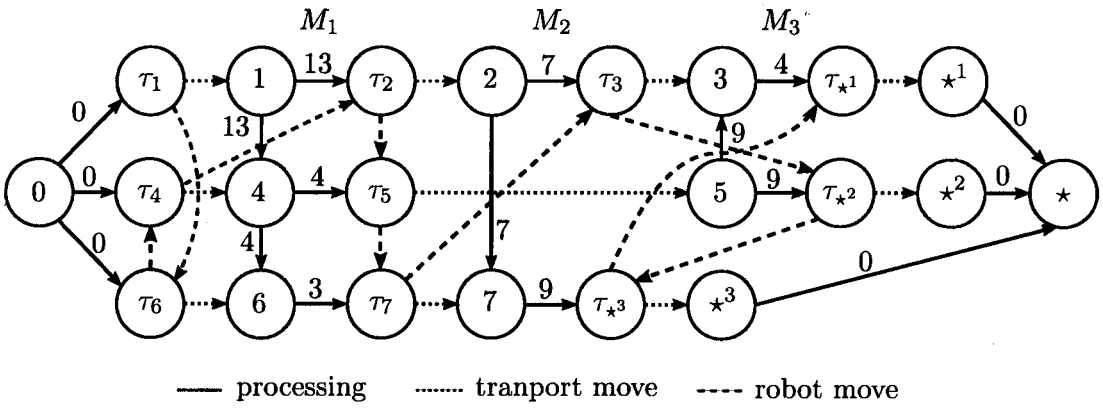


Figure 2.5: Graph for Example 2.1.2

moving time between any two machine $M(k), M(l)$ *is* $e_{kl} = 1$. *We choose the following job sequences on the machines:* $1 \to 4 \to 6$ *for* M_1, $2 \to 7$ *for* M_2 *and* $5 \to 3$ *for* M_3. *Furthermore, a robot route is given by* $R = \tau_1, \tau_6, \tau_4, \tau_2, \tau_5, \tau_7, \tau_3, \tau_{\star^2}, \tau_{\star^3}, \tau_{\star^1}$. *Figure* 2.5 *represents the graph for this job-shop problem.*

The Gantt-chart in Figure 2.6 *shows an optimal solution for this problem with* $C_{\max} = 38$. *Moreover, the route taken by the robot and the buffers at each machine are included. The robot starts at the input station* M_0, *unloads the first job* J_1 *and transports it to* M_1. *Immediately after loading, the processing of operation* 1 *starts and the robot moves back empty to the input station and unloads job* J_3. *It transports the job and loads it onto* M_1 *where* J_3 *now has to stay in the buffer* B_1, *because the machine is still busy with operation* 1. *The robot repeats this procedure with job* J_2, *which also has to stay in the buffer* B_1 *for now. Afterwards it waits at the machine until operation* 1 *has finished and unloads the job after the processing. Due to the fact that we assume that a job, having finished its processing, is moved automatically and without any delay to the buffer or has been unloaded by the robot (if the robot is already waiting), operation* 4 *can start its processing immediately after operation* 1 *has finished. After transporting* J_1 *to machine* M_2, *the robot drives back empty to machine* M_1, *waits and unloads job* J_2, *which also means operation* 6 *can start. When the last operation* 3 *in the schedule*

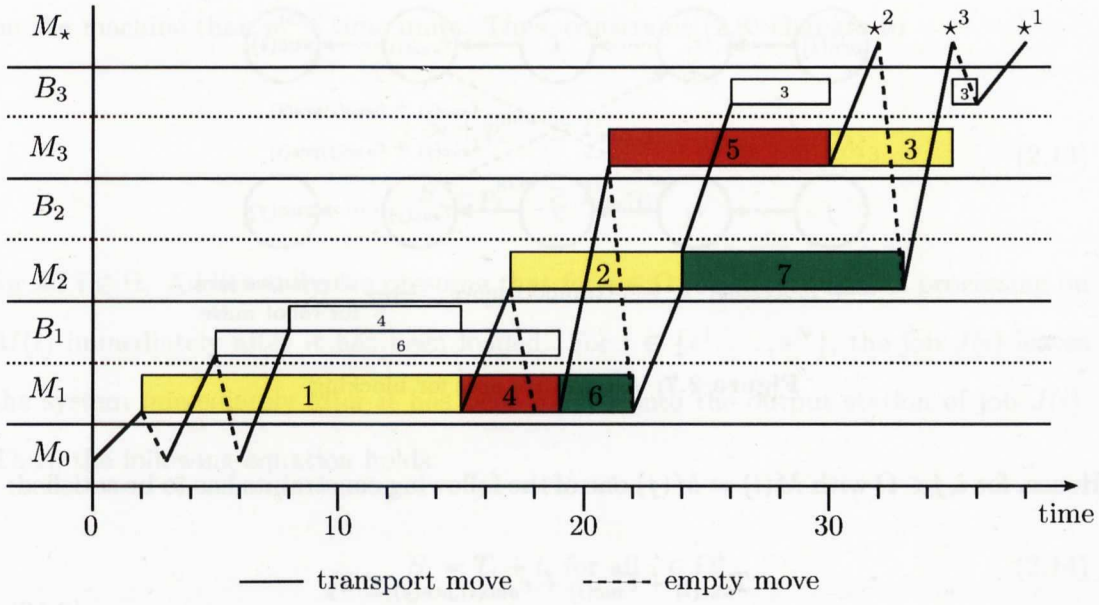


Figure 2.6: Schedule for Example 2.1.2

has finished it will be loaded in the buffer again, since the robot is currently at the output station. The robot continues until the last finished job has been transported to the output station $M_\star$.

Problems with Blocking

In this part, we assume that there is no buffer to store jobs at any machine (apart from the input and output machines M_0 and $M_\star$, which have a sufficiently large buffer) while the machine is still occupied by another job. In that case, the robot first has to unload the machine before it can transport another job to it.

We consider two operations $i, j \in \Omega$ that have to be processed on the same machine $M(i) = M(j)$. Assume that i will be processed before j. Before we can start the transport move τ_j to machine $M(j)$, we have to guarantee that $M(j)$ is already empty. This implies that job $J(i)$ has already been transported to its next machine $M(suc(i))$. Otherwise, the machine would be blocked and the robot could not load $J(j)$ onto it.

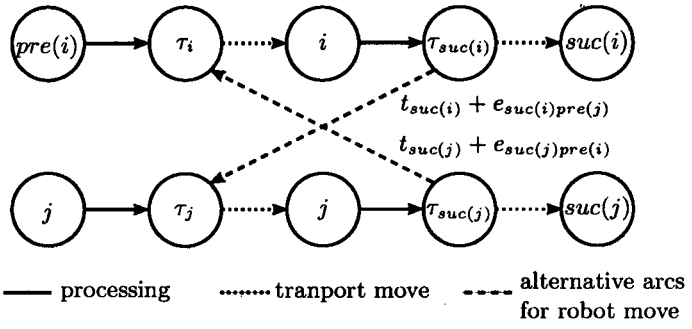


Figure 2.7: Alternative arcs for blocking

Hence, for $i, j \in \Omega$ with $M(i) = M(j)$ one of the following constraints has to be satisfied:

$$\begin{aligned} T_{suc(i)} + t_{suc(i)} + e_{suc(i),pre(j)} &\leq T_j \\ \text{or } T_{suc(j)} + t_{suc(j)} + e_{suc(j),pre(i)} &\leq T_i. \end{aligned} \tag{2.12}$$

In the graph these constraints lead to two alternative arcs. One goes from $\tau_{suc(i)}$ to τ_j and the other one from $\tau_{suc(j)}$ to τ_i. We always have to choose one of these arcs. Otherwise, we would get a cycle with positive length in the graph (cf. Figure 2.7).

For job-shop problems with transportation and blocking, a common real world constraint is represented by processing time windows (cf. Section 2.1.3). This means that a job has to be processed a minimum time at the machine, but is not allowed to stay longer as a given maximum time. Therefore, let again $p_i^{\min}$ (respectively $p_i^{\max}$) be the minimal (respectively maximal) duration of stay of Job $J(i)$ on machine $M(i)$. This means, that $J(i)$ has to stay at least $p_i^{\min}$ time units on machine $M(i)$ and has to be unloaded not later than $p_i^{\max}$ time units after the processing has started. Note that introducing a maximum processing time only makes sense for the blocking case or at least for a limited buffer at the machines. Otherwise, a job will automatically be moved into the buffer after it has been processed for its minimal time. For a feasible schedule, the following constraints must be satisfied. First of all we have to ensure, that job $J(i)$ will be processed for at least $p_i^{\min}$ time units on machine $M(i)$ and will not stay longer

on the machine than $p_i^{\max}$ time units. Thus, constraint (2.8) changes to

$$\begin{aligned} S_i + p_i^{\min} &\leq T_{suc(i)}, \\ S_i + p_i^{\max} &\geq T_{suc(i)}, \end{aligned} \tag{2.13}$$

for all $i \in \Omega$. Additionally, we presume that for $i \in \Omega$ job $J(i)$ starts its processing on $M(i)$ immediately after it has been loaded. For $i \in \{\star^1, \ldots, \star^N\}$, the job $J(i)$ leaves the system immediately after it has been loaded onto the output station of job $J(i)$. Thus, the following equation holds:

$$S_i = T_i + t_i \text{ for all } i \in \Omega^\star, \tag{2.14}$$

$$\Leftrightarrow \quad T_i = S_i - t_i \text{ for all } i \in \Omega^\star. \tag{2.15}$$

If we substitute T_i according to (2.15) in all constraints (2.9) -(2.13) then these constraints change as follows. Constraint (2.9) changes to $S_i \leq S_i$ and, therefore, becomes redundant. For constraint (2.10), we get

$$\begin{aligned} S_i + e_{i,pre(j)} + t_j &\leq S_j \\ \text{or } S_j + e_{j,pre(i)} + t_i &\leq S_i. \end{aligned} \tag{2.16}$$

for all $i, j \in \Omega^\star$. This simply means that an operation i can only start after the empty robot has picked it up from its previous machine $M(pre(i))$ and delivered it to $M(i)$. For two jobs processed on the same machine, we now have to change the constraint modeling the machine order (cf. (2.17)). Since we are dealing with a bufferless environment, a machine has to be empty before another job can be unloaded at it. We know, that job $J(i)$ has left machine $M(i)$ when $suc(i)$ has started its processing on $M(suc(i))$.

Therefore, one of the following constraints must hold.

$$\begin{aligned} S_{suc(i)} + e_{suc(i)pre(j)} + t_j &\leq S_j \\ \text{or } S_{suc(j)} + e_{suc(j)pre(i)} + t_i &\leq S_i, \end{aligned} \tag{2.17}$$

for $i, j \in \Omega$ with $i \neq j$ and $M(i) = M(j)$. The processing time windows in (2.13) can be updated to

$$\begin{aligned} S_i + p_i^{\min} + t_{suc(i)} &\leq S_{suc(i)}, \\ S_i + p_i^{\max} + t_{suc(i)} &\geq S_{suc(i)} \end{aligned} \tag{2.18}$$

for all $i \in \Omega$. Compared to the non-blocking case, now a robot route also induces the order of the jobs on the machines. More precisely, we consider two operations i, j which have to be processed on the same machine $M(i) = M(j)$. If for a given robot route R, the transport move τ_i is executed before τ_j, then operation i will be processed before operation j. That means, if we choose $S_{suc(i)} + e_{suc(i),pre(j)} + t_j \leq S_j$ in (2.17) then $S_i \leq S_j$ is induced by (2.18). The job-shop problem with one transport robot and blocking can be designed as finding an earliest start schedule that fulfills constraints (2.14)-(2.18). After the calculation of optimal S_i-values, the corresponding T_i-values are given by (2.15).

Example 2.1.3. *We consider the same data as in Example* 2.1.2. *For the time window constraints we set* $p_i^{\min} = p_i$ *and* $p_i^{\max} = p_i^{\min} + 10$ *for all* $i \in \Omega$. *A feasible (and also optimal) robot route for this problem is* $R = (\tau_4, \tau_5, \tau_1, \tau_{\star^2}, \tau_2, \tau_6, \tau_3, \tau_7, \tau_{\star^1}, \tau_{\star^3})$. *The robot starts at the input station* M_0, *unloads job* J_2, *transports and loads it onto* M_1. *As machine* M_1 *is now blocked and the first operations of both* J_2 *and* J_3 *also have to be processed on* M_1, *the robot has to wait until operation* 4 *is finished. After that, it can unload* J_2 *and transport it to its next machine* M_3. *Since* M_1 *is not blocked anymore, the robot drives empty to the input station, picks up* J_1 *and continuous according to*

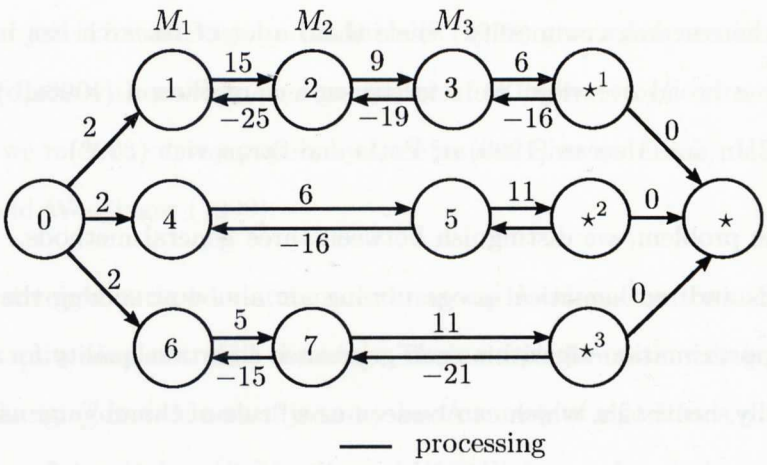


Figure 2.8: Graph for Example 2.1.3 with $R = \tau_4, \tau_5, \tau_1, \tau_{\star^2}, \tau_2, \tau_6, \tau_3, \tau_7, \tau_{\star^1}, \tau_{\star^3}$

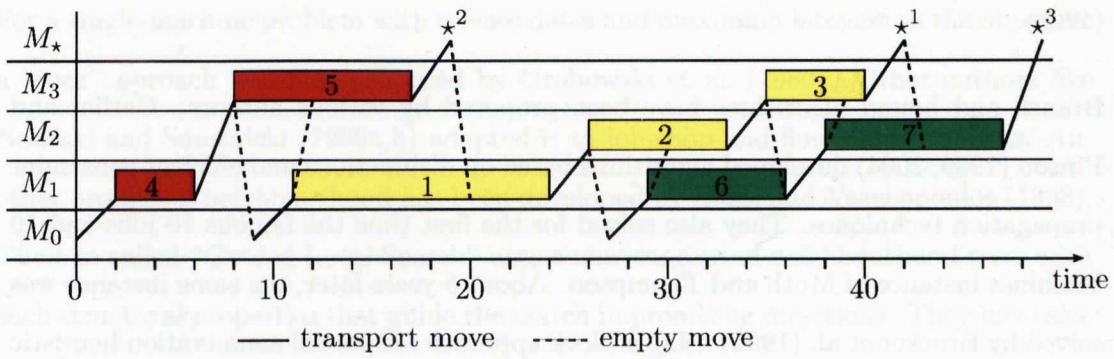


Figure 2.9: Schedule for Example 2.1.3

R. The graph for this problem is shown in Figure 2.8. *The arcs representing the robot route have been omitted. The corresponding and also optimal schedule with* $C_{\max} = 49$ *can be found in Figure* 2.9.

2.1.5 Literature Review

The classical job-shop problem is well known in the literature and its studies date back to 1960. A benchmark instance with 10 jobs and 10 machines was introduced in Muth and Thompson (1963) which could first be solved 25 years later. Garey et al. (1976) proved that the problem is NP-hard. The complexity of the problem makes it a tough

challenge for the research community. Since then, a lot of research has been done in this area. For a broad overview, we refer to surveys of Pinson (1995a,b), Blazewicz et al. (1996), Jain and Meeran (1999) or Potts and Strusevich (2009).

For solving the problem, we distinguish between three general methods. Branch and bound methods and mathematical programming are aimed at solving the problem to optimality. Approximation algorithms can guarantee a certain quality for the reached solution. Finally, heuristics, which can be seen as a "rule of thumb" are usually pretty fast, but cannot give any guarantee on the quality of the solution. A good overview about various techniques for solving the job-shop problem is given in Jones and Rabelo (1998).

Branch and bound algorithms have been proposed by various authors. Carlier and Pinson (1989, 2004) developed algorithms based on disjunctions and efficient constraint propagation techniques. They also solved for the first time the famous 10 jobs and 10 machines instance of Muth and Thompson. About 5 years later, the same instance was solved by Brucker et al. (1994) using a block approach. A partial enumeration heuristic based on the branch-and-bound method by Carlier and Pinson (1989) has been presented by Applegate and Cook (1991). They developed an approximation method, the shuffle algorithm, where one or more machine orderings are fixed and the remaining ones are optimally completed using a branch-and-bound method. Martin and Shmoys (1996) developed an enumerative procedure which is based on time-oriented branching schemes.

Approximations algorithms are polynomial time algorithms which are guaranteed to find a feasible solution that is at most ρ times the optimal value. The value ρ is called a "worst-case ratio bound". In Jansen et al. (2001) a polynomial time approximation scheme is presented where the number of machines and as well as the number of

operations per job is fixed. Feige and Scheideler (1998) give a polynomial time approximation algorithms with performance guarantee of $O(\log(mn)\log(\log(mn)))$. For more information, we refer to survey papers by Lenstra and Shmoys (1995), Hall (1997) and Schuurman and Woeginger (1999).

One of the first neighbourhood search procedures was developed by Irwin and Wilkerson (1971) and is quite similar to hill climbing. Their simple heuristic rule is based on the idea of interchanging certain non-adjacent jobs. A survey of algorithms with an emphasis on local search is presented in Aarts et al. (1994). In van Laarhoven et al. (1992) a neighbourhood based on critical arcs was first used in a simulated annealing algorithm. For a single-machine problem with release dates and maximum lateness as the objective a block approach was first proposed by Grabowski et al. (1986). Other authors like Nowicki and Smutnicki (1996a,b) adapted it to job-shop and flow-shop problems. Another promising neighbourhood has been developed by Balas and Vazacopoulos (1998). Their so called "Guided Local Search" uses a new concept of neighbourhood trees with such structural properties that guide the search in promising directions. They have also embedded their method in a shifting bottleneck framework to create a hybrid procedure that takes advantages of the differences between the two neighbourhood structures.

Also simulated annealing (often in combination with other methods) has been applied effectively to the job-shop problem. In van Laarhoven et al. (1992) an algorithm is presented that is proved to converge asymptotically to the global minimum. El-Bouria et al. (2007) combined with their heuristic a simulated annealing module and two short-term memories. A hybrid optimisation strategy based on a combination of tabu search and simulated annealing is presented by Zhang et al. (2008).

Among the local search algorithms discussed above, the tabu search algorithm of Nowicki and Smutnicki (1996b) and the guided local search algorithm employing the shifting

bottleneck procedure of Balas and Vazacopoulos (1998) are the most effective.

Vaessens et al. showed in Glover (1997) that tabu search methods in specific scheduling cases are superior over other approaches such as simulated annealing, genetic algorithms and neural networks. For the job-shop problem with flexible machines, Hurink et al. (1994) developed a tabu search algorithm based on a block approach. Mastrolilli and Gambardella (2000) proposed a tabu search based on the reduced neighbourhood for the same problem. For changing an operation in a machine sequence they reduce the set of possible neighbours to a subset that always contains the optimal sequence for this swap. An advanced tabu search algorithm is presented by Nowicki and Smutnicki (2005) which evaluates neighbour solutions in an efficient way.

Different models of buffers for job-shop problems are investigated in Brucker et al. (2006). They proved that pairwise buffers, job-dependent buffers and input- or output buffers have the property that, if schedules are represented by machine sequences, then the corresponding optimal schedules can be calculated in polynomial time. A more detailed overview about job-shop problems with limited buffers can be found in Heitmann (2007).

For the job-shop with no buffers and blocking constraints, Mascisa and Pacciarelli (2002) formulated the problem with alternative graphs and introduced dispatching rule based heuristics to solve the problem. Furthermore, they have developed a branch and bound method and even solved several large (10×10) instances to optimality. Meloni et al. (2004) presented a constructive approach (a rollout metaheuristic) based on an alternative graph formulation that, that iteratively extends a partial schedule by fixing some alternative arcs. All possible next candidates are evaluated using a scoring function and the arc providing the best score is added to the partial selection. In Gröflin and Klinkert (2009) a neighbourhood based on a disjunctive graph is presented. Their

approach is based on the exchange of critical alternative arcs that guarantees to get a new feasible solution providing a key theorem of "short cycles". Their neighbourhood has been integrated in a tabu search and promising computational results are presented.

Today, most researchers concentrate their studies on variations of the classical job-shop problem. A possible reason for this could be the applicability of their developed solutions to problems in the real world. This includes the problems with transportation as presented in Section 2.1.4.

Several modern environments require transport robots, so generalising the classical job-shop problem leads to more realistic models. As many authors investigate special cases of the problem, only a few results are available for the general case, and most of them also only consider problems without blocking. Kise (1991) proved that minimising the makespan in a two-machine flow-shop with constant transportation time and a single robot is already $\mathcal{NP}$-hard. Different aspects of job-shop problems with an unlimited or limited number of transport robots and without blocking is considered in Knust (1999). In Bilge and Ulusoy (1995), a heuristic for simultaneously scheduling the machines and robots in a flexible manufacturing system with job-shop structure is proposed. Brucker and Knust (2002) studied constraint propagation techniques for the job-shop problem with one transport robot. One-stage and two-stage tabu search algorithms for the job-shop problem with one transport robot were proposed by Hurink and Knust (2005). Lee and Strusevich (2005) considered the two-machine problem, where the robot is allowed to transport an arbitrary number of jobs at a time. Several complexity results for unlimited buffer flow-shops where either the processing times or the transportation times are constant are presented by Hurink and Knust (2001).

General problems with blocking are rarely discussed in the literature. Lacomme and Tchernev (2006) presented a memetic algorithm where limited or even no buffers can be included in the problem definition. However, many authors instead study flow-shop

problems with transportation times and blocking constraints. Panwalkar (1991) and Levner et al. (1995) considered a two-machine case without an output buffer behind the first machine. They showed that this problem is solvable in polynomial time and both developed an algorithm for the problem. The two-machine case with an additional no-wait constraint has been studied by Stern and Vitner (1990).

2.2 Cyclic Job-Shop Problems

There are many situations where it happens that a job is not going to be produced only once but several times. That means, many instances of the same job have to be processed. In large scale production, for instance, there usually is a fixed set of jobs that have to be processed indefinitely often. If we model this as a classical job-shop problem we have to include different repetitions of the same job in our schedule. This could be a huge number of individual jobs and, therefore, very hard to solve. A slightly different idea would be to define a minimal part set (MPS) of all jobs, which basically is the ratio in which the jobs will be produced. For example, if we have three different types of products J_1, J_2 and J_3 and we want to produce 100 items of J_1, 300 items of J_2 and 200 item of J_3 then a minimal part set would be given by $J_1 : 1$, $J_2 : 3$ and $J_3 : 2$. The non-cyclic approach would be to find a schedule in which all 600 job items are produced, whereas the cyclic approach would be to find a schedule in which one repetition of J_1, three repetitions of J_2 and two repetitions of J_3 are produced and repeat this schedule 100 times. Note that the minimal makespan for producing all 600 items in the cyclic case is at most as good (usually it is worse) as the one for the non-cyclic case. On the other hand, solving the cyclic problem usually takes less time, so one has to to deliberate about whether time or quality matters more. Summarising, cyclic scheduling tries to find a good schedule for the MPS and simply repeat this schedule over and over again. This is the basic idea of cyclic job-shop scheduling which will be introduced within this section.

2.2.1 The Basic Model

To distinguish between the different repetitions of each operation, we denote the r-th repetition of operation $i \in \Omega^\star$ by $\langle i; r\rangle$ where $r \in \mathbb{Z}$ is called *repetition number*. Applying the latter idea, a cyclic schedule for the data in Example 2.1.1 would be as

shown in Figure 2.10. A schedule is called *cyclic* with *cycle time* $\alpha \geq 0$ if

$$S_i(r) = S_i(0) + r\alpha, \tag{2.19}$$

for all $i \in \Omega$ and $r \in \mathbb{Z}$, where $S_i(r)$ denotes the starting time of operation i at the r-th repetition.

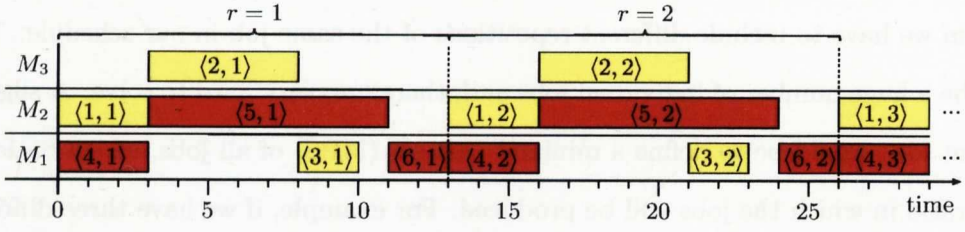


Figure 2.10: Possible cyclic schedule for Example 2.1.1 with cycle time $\alpha = 13$

We also enforce, that the $(r+1)$-th repetition of operation i cannot start before the r-th repetition of it has been finished. Therefore, we get the following constraint

$$S_i(r) + p_i \leq S_i(r+1), \tag{2.20}$$

for all $i \in \Omega$ and $r \in \mathbb{Z}$. A time interval of length α is called a *cycle*. Constraint (2.19) implies that in every cycle all operations have to start exactly once, because the distance between two consecutive repetitions of an operation (e.g. $\langle i, r\rangle$ and $\langle i, r+1\rangle$) is the cycle length α. Furthermore, constraint (2.20) together with (2.19) guarantees that $p_i \leq \alpha$ for all $i \in \Omega$ which means, that every operation also finishes exactly once in each cycle.

The remaining precedence and machine constraints are similar to the non-cyclic job-shop problem.

$$S_i(r) + p_i \leq S_{suc(i)}(r), \tag{2.21}$$

for all $i \in \Omega, r \in \mathbb{Z}$ and assuming that the machines have a sufficient large buffer to store unprocessed or finished jobs

$$\begin{aligned} &\text{either } S_i(r_i) + p_i \leq S_j(r_j) \\ &\text{or } S_j(r_j) + p_j \leq S_i(r_i), \end{aligned} \tag{2.22}$$

has to hold for all $i, j \in \Omega$ and $r_i, r_j \in \mathbb{Z}$ with $M(i) = M(j)$. In general, the objective of the cyclic scheduling problem is to find a schedule with minimum cycle time α. This is equivalent to maximising the throughput rate, which is the number of completed jobs in a given time window.

The key property of cyclic scheduling is that r_i and r_j in constraint (2.22) do not have to be the same in a specific cycle. Consider again Figure 2.10 which shows two complete repetitions of a cycle. The first repetition goes from 0 to 13 and the second from 13 to 26. In both cycles every operation has the same repetition number. Also noticeable is that every machine has some idle time in a cycle which usually is an indicator that the quality of the schedule might be improvable. A possible improved schedule for this problem with cycle time $\alpha = 11$ is shown in Figure 2.11. It is important to realise, that the precedence constraints do still hold for operations belonging to the same repetition of a job and are not violated. It just happens that the operations of a specific repetition of a job are spread out over more than one cycle. This time, from the start of the first operation of a job repetition until the last operation has finished, is called the *flow time* φ_j of job J_j. For instance, job J_1 has a flow time of $\varphi_1 = 10$ in Gantt-chart 2.10 and a flow time of $\varphi_1 = 16$ in the improved schedule in Figure 2.11. Obviously, a lower bound for a job's flow time is given by the sum of all processing times of this job. The cycle time and the flow time are usually negatively correlated. That means, a smaller cycle time tends to lead to a larger flow time and vice versa. Especially in practice, the flow time can be an important factor (e.g. delivery deadlines). Even if every job repetition

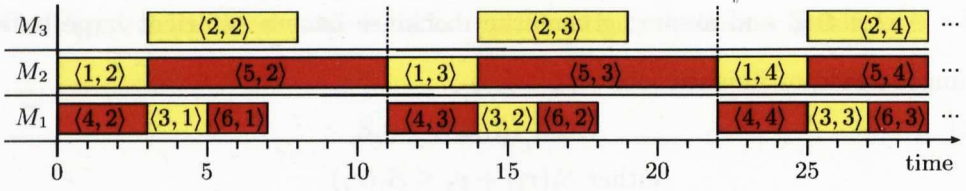


Figure 2.11: Improved cyclic schedule for Example 2.1.1 with cycle time $\alpha = 11$

is of the same type, they can still be individual. E.g. in car manufacturing every cycle a complete car leaves the assembly line. However, the cars have different colours, configuration, engine, and so on. Consequently, a larger flow time makes customers wait longer for their special orders. The variable, which builds the connection between the flow time and the cycle time, is the *height*. For a general constraint between two starting points $S_i(r_i)$ and $S_j(r_j)$ it holds

$$S_i(r_i) + d_{ij} \leq S_j(r_j), \tag{2.23}$$

where $d_{ij} \in \mathbb{R}$ is a minimum time lag between the two starting points. The height $h_{ij} \in \mathbb{Z}$ of such a constraint is defined as the difference between r_j and r_i:

$$h_{ij} = r_j - r_i. \tag{2.24}$$

In the following, we will introduce different models for cyclic job-shop problems as they appear in the literature. The difference will be in some additional constraints, that have an upper bound for the height given in advance. We will see how different heights will influence the resulting solutions.

2.2.2 Specific Models

In the literature, four different models of a cyclic job-shop problem are presented. We describe these models in terms of the underlying graph $G = (V, E)$. In general, a *cyclic*

job-shop scheduling problem without blocking (CJSP) can be formulated as minimise α such that constraints (2.19) - (2.23) are fulfilled. As in the non-cyclic problem, the difficulty lies in finding a feasible selection for the machine constraints (2.22).

The first model is very general and also provides the basic structure of the three other models. In this case, the set V is equal to Ω and

$$E = \{(i,i) \mid i \in \Omega\} \cup \{(i, suc(i)) \mid i \in \Omega;\ \text{successor } suc(i) \text{ exists}\}.$$

We assign to each arc $(i, suc(i))$ a delay $d_{i,suc(i)} = p_i$ and a height $h_{i,suc(i)} = 0$ (cf. constraints (2.21)). Furthermore, for every operation i we add a loop from i to i with delay $d_{ii} = p_i$ and height $h_{ii} = 1$. These arcs are representing the constraints (2.20).

Example 2.2.1. *Consider a cyclic job-shop problem with $N = 2$ jobs and $m = 4$ machines. Each job consists of 4 operations. The processing times and the machine allocations are given in the following table.*

Job	J_1				J_2			
Operation	1	2	3	4	5	6	7	8
Processing time	4	4	4	4	2	3	3	2
Machine	M_1	M_2	M_3	M_4	M_1	M_3	M_4	M_2

Figure 2.12 shows the basic graph for this example. The optimal cycle time is $\alpha = 7$. A corresponding schedule is shown in Figure 2.13.

Note that the optimal cycle time α_{opt} by using this model is always equivalent to the maximum *machine utilisation*, which is the sum of processing times on the same machine, i.e.

$$\alpha_{opt} = \max_{k=1}^{m} \sum_{\substack{i \in \Omega \\ M(i)=M_k}} p_i. \tag{2.25}$$

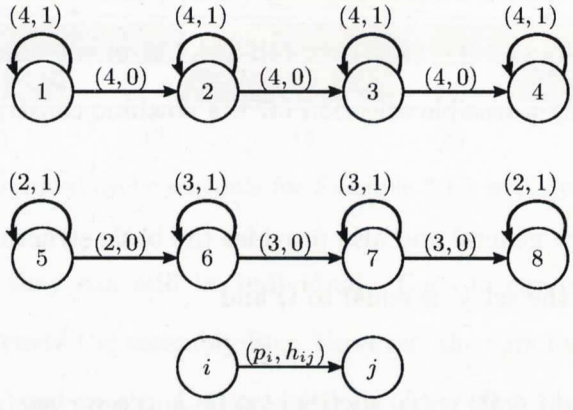


Figure 2.12: Graph for Example 2.2.1

It is easy to see that a schedule with a smaller cycle time than α_{opt} in (2.25) cannot be reached. Otherwise, some operations would overlap on a machine. The following method shows how to construct a feasible schedule with a maximum utilisation of at least one machine.

1. Start with a machine M_k that has maximum utilisation.

2. Put the jobs in ascending order according to their number of operations and schedule all operations one after another on M_k in this order. Assume that this order is $J_1, J_2, \ldots, J_m$ where $m \leq N$. Therefore, let $r \in \mathbb{Z}$ be the repetition number of all operations on M_k in this cycle. We start with the job that has the fewest operations and schedule its operation i with $M(i) = M_k$.

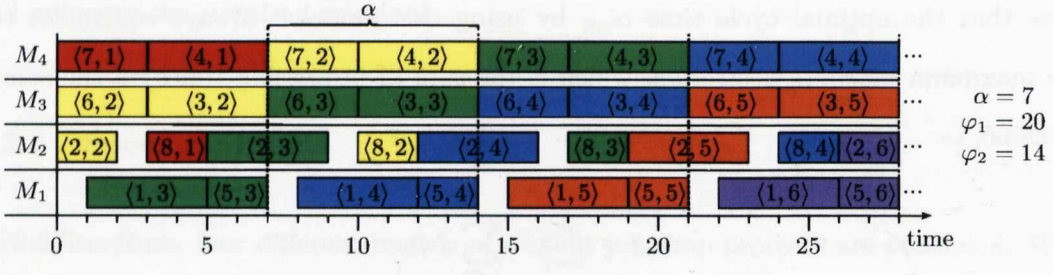


Figure 2.13: Optimal schedule for Example 2.2.1

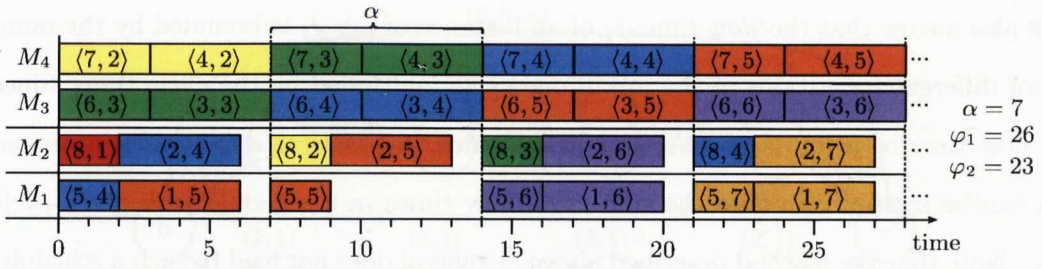


Figure 2.14: Constructed schedule for Example 2.2.1

3. Schedule all remaining operations of the job in ascending order according to the precedence constraints at the earliest point in time $\geq S_i(r)$ on their machines. After all operations j with $J(j) = J(i)$ have been scheduled we update their repetition numbers to make sure that the precedence constraints are not violated. This can be done in the following way:

   - Start with the last operation $i^\star$ of a job and assign an arbitrary repetition number $r^\star$ to it. If the proceeding operation $pre(i^\star)$ has finished before $i^\star$ starts, assign $r^\star$ as a repetition number to $pre(i^\star)$. Else. assign $r^\star + 1$.
   - Repeat the process for all remaining jobs $J_2, \ldots, J_m$ on M_k and afterwards with all other remaining jobs $J_{m+1}, \ldots, J_N$.

Figure 2.14 shows another feasible schedule for Example 2.2.1 constructed by the method described above. The machines with the highest utilisation of 7 are M_3 and M_4. Since both jobs have the same number of operations we start with job J_2, in particular with operation $\langle 7;2\rangle$ on M_4. After scheduling this operation at time 0, we schedule all other operations of J_2 in ascending order as early as possible but not before $S_7(2) = 0$. Afterwards, we adjust the repetition numbers. Thus, operation $\langle 5;4\rangle$ is scheduled on M_1 at time 0, $\langle 6;3\rangle$ on M_3 at time 0 and operation $\langle 8;1\rangle$ is scheduled on M_2 at time 0. After that, we schedule J_1 in the same way.

Applying this method to a problem without job repetition (which we always consider within this work), there are at most m different repetitions of each job in one cycle.

This also means that the *flow time* φ_j of an instance of job J_j is bounded by the number of different repetitions of the job in one cycle multiplied by the cycle time. Since the flow time of a job is another important value in practical application, a second aim can be seen to minimise the sum of all flow times in a schedule with given cycle time. Note that the method described above in general does not lead to such a schedule.

The second model is called the *cyclic job-shop problem* and is a generalisation of the model of the non-cyclic version. We expand the graph of the first model by introducing a dummy start node 0 and a dummy end node $\star$. For every job, there is an arc leading from 0 to its first operation with delay and height equal to 0. Also, we have an arc from the last operation i^* of every job to $\star$ with $d_{i^*\star} = p_{i^*}$ and $h_{i^*\star} = 0$.
Furthermore, we add an arc from the dummy end node $\star$ to the dummy start node 0 where $d_{\star,0} = 0$ and $h_{\star,0}$ is a parameter. In this model, for each operation i the r-th repetition of i has to be finished before the $(r + h_{\star,0})$-th repetition of i can start. The height $h_{\star,0}$ also limits the number of different repetitions of operations in the same cycle. In general, the following holds: the larger the height $h_{\star,0}$, the smaller the optimal cycle time (until the maximal utilisation of one machine is reached). On the other hand, for the flow time the following holds:

$$\varphi_j \leq h_{\star,0} \cdot \alpha$$

for all $j \in \{1, \ldots, N\}$. In case $h_{\star,0} = 1$, the optimal cycle time and completion time of all jobs is equivalent to the minimal makespan of the classical job-shop problem. The following example will illustrate how the choice of a maximal height $h_{\star,0}$ can influence the optimal solution of a cyclic job-shop problem.

Example 2.2.2. *Consider the same data as in Example* 2.2.1. *Figure* 2.15 *shows the directed graph for this model. This time, the optimal cycle time depends on the*

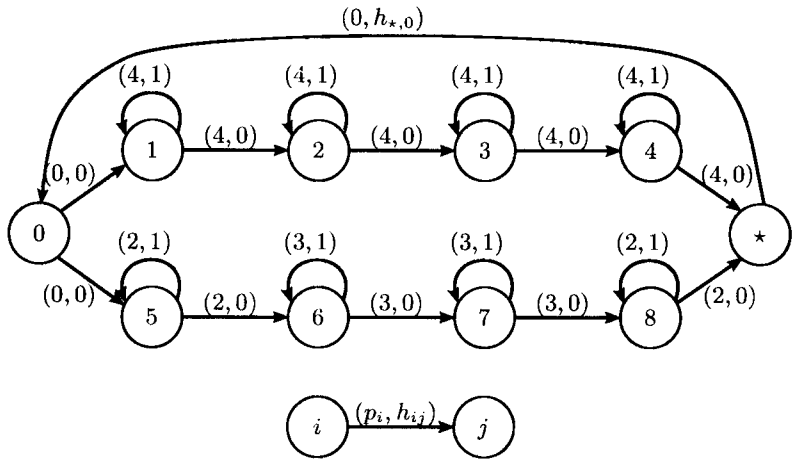


Figure 2.15: Graph for Example 2.2.2

value of $h_{\star,0}$. *For* $h_{\star,0} = 1$ *we get an optimal schedule with cycle time* $\alpha = 17$ *equal to the makespan of the non-cyclic case as shown in the first Gantt-chart of Figure* 2.16. *Increasing the height* $h_{\star,0}$ *to* 2, *gives the possibility that operations with different repetition numbers can be processed in the same cycle. Hence, the utilisation of the machines also increases and the schedule gets more compact as we can see in the second Gantt-chart of Figure* 2.16. *The optimal cycle time for* $h_{\star,0} = 2$ *is* $\alpha = 9$. *Note that the flow time* φ_1 *of job* J_1 *has increased from* 14 *to* 17.

The third model is called the *cyclic job-shop problem with job repetition.* Again, we expand the first model. For each job an arc from its last operation to its first operation is introduced. The delays of these arcs are equal to the processing times of the corresponding last operation. The parameter for the height of these arcs is denoted by h_{J_j} with $j \in \{1, \ldots, N\}$. In this model, the r-th repetition of all operations belonging to the same job J_j has to be finished, before the $(r + h_{J_j})$-th repetition of this job can start.

Example 2.2.3. *As before, we illustrate this model with the data of Example* 2.2.1. *To the basic graph in Figure* 2.12 *we add an arc from* 4 *to* 1 *with delay* $d_{41} = p_4 = 4$ *and another one from* 8 *to* 5 *with delay* $d_{85} = p_8 = 2$. *Both arcs have the same*

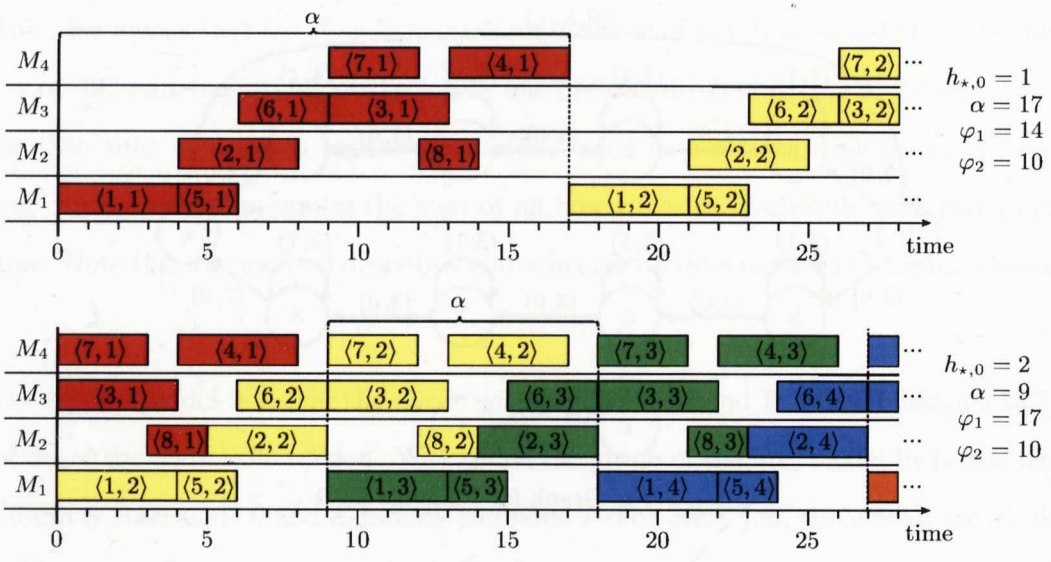


Figure 2.16: Schedules for Example 2.2.2

height parameter $h_{J_1} = h_{J_2}$. *Figure* 2.17 *shows the corresponding graph. For purposes of clarity, we omit the labels of the loops, because they are identical to the labels in Example* 2.2.1 *and* 2.2.2. *If we set* $h_{J_1} = h_{J_2} = 1$ *we get the schedule with cycle time* $\alpha = 16$ *(cf. first Gantt-chart in Figure* 2.18*) and by increasing* h_{J_1}, h_{J_2} *to* 2, *the cycle time reduces to* $\alpha = 9$ *(cf. second Gantt-chart in Figure* 2.18*).*

Compared to the second model, we can see that heights of $h_{J_1} = h_{J_2} = 1$ can already

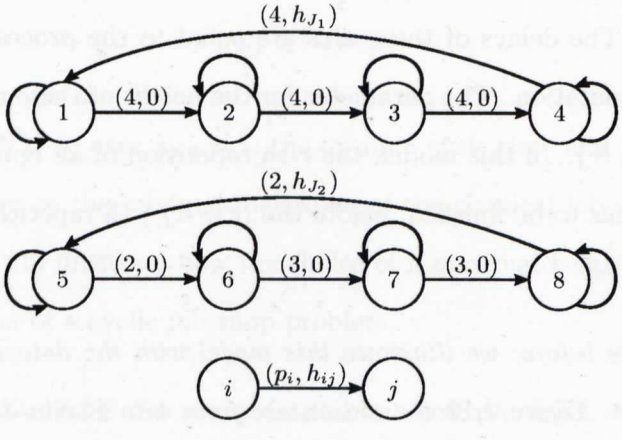


Figure 2.17: Graph for Example 2.2.3

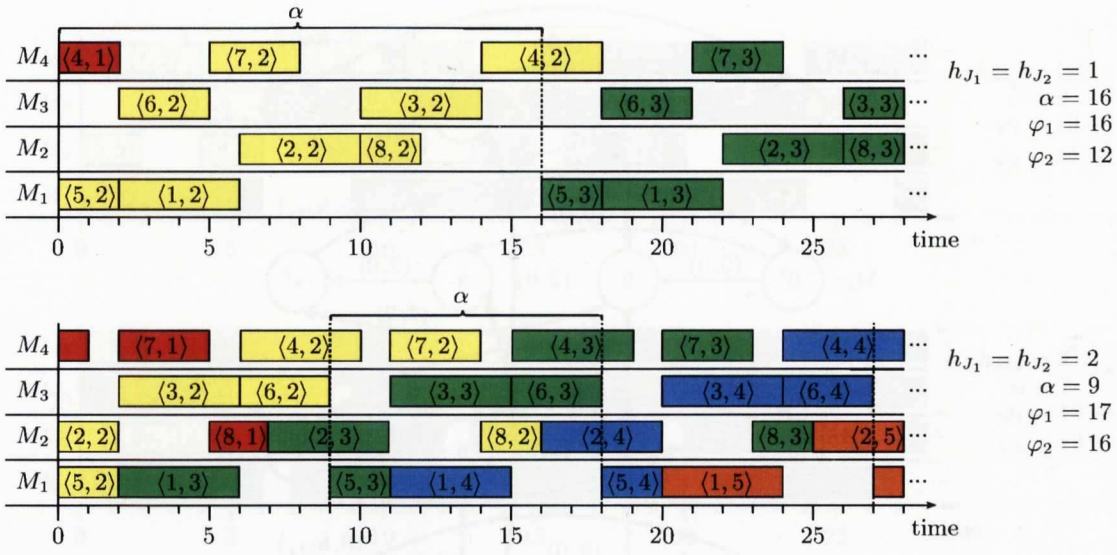


Figure 2.18: Schedules for Example 2.2.3

provide a schedule where operations with different repetition numbers can occur in the same cycle. The reason is that the start of processing a new instance of a job only depends on the finishing of a previous instance of the *same* job and not on the finishing of *all* jobs.

The fourth and last model we will describe is called the *cyclic job-shop problem with machine repetition*. Within this, the graph of the first model is extended as follows. We add a dummy start node 0^k and a dummy end node $\star^k$ for each machine $M_k \in M$ to V. Furthermore, we introduce an arc from 0^k to each operation processed on machine M_k of length 0 and height 0 and an arc from each operation $i \in \Omega$ processed on M_k to the dummy end node $\star^k$ of length p_i and height 0. Finally, for each machine M_k there exists an arc from $\star^k$ to 0^k of length 0 and variable height h_{M_k}. In this model, the height limits the number of different repetitions on the same machine during one cycle. In particular, on each machine the difference between the repetition numbers of the operations processed on the same machine during one cycle can be at most h_{M_k}. The following example will illustrate this extension.

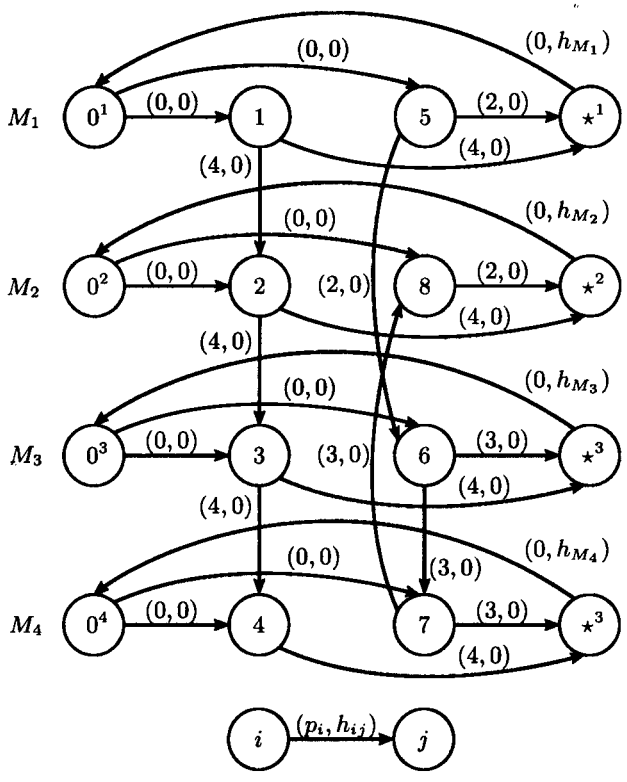


Figure 2.19: Graph for Example 2.2.4

Example 2.2.4. *Using the data from Example* 2.2.1 *the graph for the the cyclic job-shop problem with machine repetition is shown in Figure* 2.19. *We again omit the loops for the purposes of clarity and refer to the loops in Example* 2.2.1. *An optimal schedule with height* $h_{M_k} = 1$ *is shown in the first Gantt-chart of Figure* 2.20. *The optimal cycle time is* $\alpha = 9$. *By increasing* h_{M_1} *and* h_{M_2} *up to* 2 *we get an optimal schedule as shown by the second Gantt-chart. The minimal cycle time is* $\alpha = 7$.

2.2.3 Blocking Constraints

In this section, we briefly expand the CJSP to the blocking case, in which we assume that an operation i is blocking a machine $M(i)$ as long as the succeeding operation $suc(i)$ has not started its processing on $M(suc(i))$ (cf. Section 2.1.2). Note that the last operation of every job is never a blocking operation, since the succeeding machine

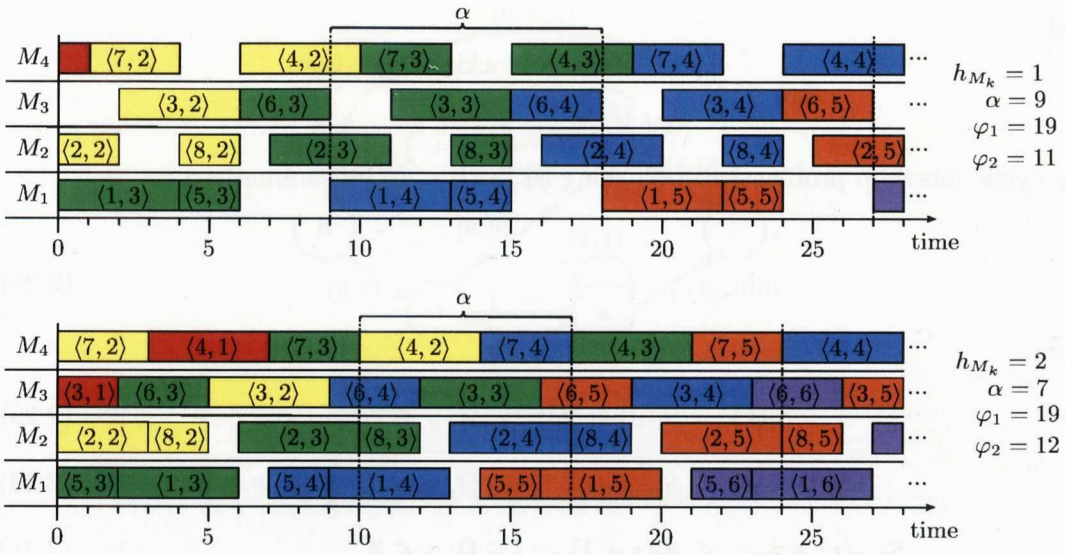


Figure 2.20: Schedules for Example 2.2.4

$M_\star$ is supposed to have a large enough buffer to store the finished jobs. Assume that $i, j \in \Omega$ with $M(i) = M(j)$ are blocking operations. Then, we have to replace constraint (2.22) by

$$\begin{aligned} S_{suc(i)}(r_{suc(i)}) &\leq S_j(r_j) \\ \text{or } S_{suc(j)}(r_{suc(j)}) &\leq S_i(r_i), \end{aligned} \tag{2.26}$$

for all blocking operations $i, j \in \Omega$, $r_i, r_j, r_{suc(i)}, r_{suc(j)} \in \mathbb{Z}$ with $M(i) = M(j)$. Furthermore, for a blocking operation i it has to be satisfied, that the $(r+1)$-th repetition of it can only start when the r-th repetition of its succeeding operation $suc(i)$ has started. Thus, we get

$$S_{suc(i)}(r) \leq S_i(r+1), \tag{2.27}$$

for all blocking operations $i \in \Omega$ and $r \in \mathbb{Z}$. Defining

$$b(i) := \begin{cases} suc(i), & \text{if } i \text{ is a blocking operation;} \\ i, & \text{else,} \end{cases}$$

and

$$p_{b(i)} := \begin{cases} 0, & \text{if } i \text{ is a blocking operation;} \\ p_i, & \text{else,} \end{cases}$$

the cyclic job-shop problem with blocking (CJSPB) can be summarised as

$$\min \ \alpha \tag{2.28}$$

s.t.

$$S_i(r) = S_i(0) + \alpha r \quad i \in \Omega;\ r \in \mathbb{Z} \tag{2.29}$$

$$S_i(r) + p_i \leq S_{suc(i)}(r) \quad i \in \Omega, suc(i) \text{ exists};\ r \in \mathbb{Z} \tag{2.30}$$

$$S_{b(i)}(r) + p_{b(i)} \leq S_i(r+1) \quad i \in \Omega;\ r \in \mathbb{Z} \tag{2.31}$$

$$S_{b(i)}(r_{b(i)}) + p_{b(i)} \leq S_j(r_j) \quad i, j \in \Omega;\ i \neq j;\ r_i, r_j, r_{b(i)}, r_{b(j)} \in \mathbb{Z}$$

$$\text{or } S_{b(j)}(r_{b(j)}) + p_{b(j)} \leq S_i(r_i) \quad \text{with } M(i) = M(j). \tag{2.32}$$

As in the non-blocking case, there are different ways of modeling the problem and building a graph. We will only briefly discuss the second case with one backward arc from the previous section. Again, we have a node for each operation $i \in \Omega$ as well as a source node 0 and a sink node $\star$. We introduce arcs leading from the source node to the first operation of every job with delay and height equal to 0. From the last operation $i^\star$ of every job, we add an arc to the sink node of delay equal to $p_{i^\star}$ and height equal 0. Furthermore, we add an arc from the sink node to the source node with $d_{\star,0} = 0$ and $h_{\star,0}$ as a parameter. Constraints (2.30) are modeled in the same way as before. For every constraint (2.31), we add an arc from $b(i)$ to j with $d_{b(i),j} = p_{b(i)}$ and height $h_{b(i),j} = 1$. Moreover, for every constraint (2.32) we add two alternative arcs: one from $b(i)$ to j with $d_{b(i),j} = p_{b(i)}$ and height $h_{b(i),j} = 0$ and another one from $b(j)$ to i with $d_{b(j),i} = p_{b(j)}$ and height $h_{b(j),i} = 0$. We illustrate the blocking situation with a simple example.

Example 2.2.5. *We consider two jobs J_1, J_2 and two machines M_1, M_2. Each job*

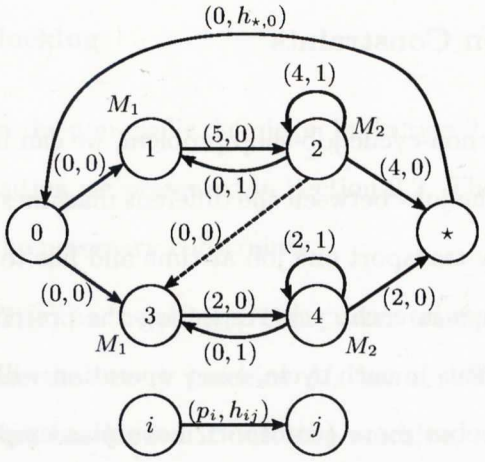


Figure 2.21: Graph for Example 2.2.5

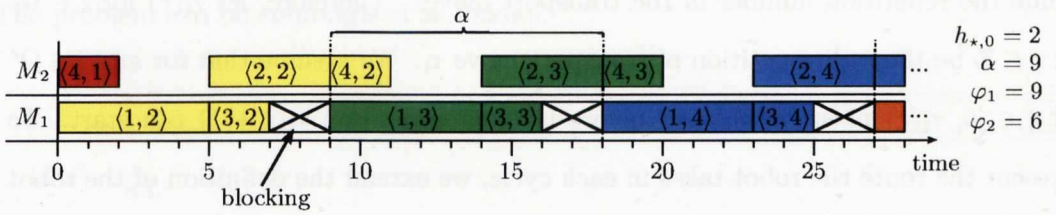


Figure 2.22: Schedules for Example 2.2.5

consists of two operations. The following table shows the processing times and machine allocations of the operations.

Job	J_1		J_2	
Operation	1	2	3	4
Processing time	5	4	2	2
Machine	M_1	M_2	M_1	M_2

The first operation of each job is a blocking operation. Figure 2.21 *shows the graph of the problem. Additionally, to the precedence constraints we have also included the selection of the machine constraint in an optimal solution. It is represented by a dashed arc from* 2 *to* 3. *An optimal schedule with height* $h_{\star,0} = 2$ *and cycle time* $\alpha = 9$ *can be found in Figure* 2.22.

2.2.4 Transportation Constraints

In the same way as in the non-cyclic job-shop problem, we can include a robot into the process, that transports the jobs between the different machines (cf. Section 2.1.4). As before, the robot can only transport one job at time and has to perform a sequence of transport and empty moves in each cycle respecting the precedence constraints of all jobs. Again, we assume that, in each cycle, every operation will be done exactly once, which also means every robot move (transport move plus empty move or waiting) is done exactly once. To distinguish between the different repetitions of those moves, we include the repetition number in the transport moves. Therefore, let $\tau_i(r)$ for $i \in \Omega^\star$ and $r \in \mathbb{Z}$ be the r-th repetition of transport move τ_i. We assume that for any $i \in \Omega^\star$ and $r \in \mathbb{Z}$, $\tau_i(r)$ must be finished before its next repetition $\tau_i(r+1)$ can start. To represent the route the robot takes in each cycle, we extend the definition of the robot route R from Section 2.1.4 to a *robot cycle* by adding the repetition number. Hence,

$$R = \tau_{\sigma(1)}(r_{\sigma(1)}), \tau_{\sigma(2)}(r_{\sigma(2)}), \ldots, \tau_{\sigma(N+n)}(r_{\sigma(N+n)}),$$

where $\sigma : \Omega^\star \to \Omega^\star$ is a permutation of the set $\Omega^\star$. As before, we assume that after an empty move of the robot a transport move always follows, and that for all empty moves the triangle inequality

$$e_{ik} + e_{kl} \geq e_{il} \text{ for all } i, k, l \in \Omega^\star \cup \{0\}$$

holds. We also assume that $e_{ij} = 0$ for $M(i) = M(j)$ and the empty moving time from $M(i)$ to $M(j)$ is the same as from $M(j)$ to $M(i)$, so $e_{ij} = e_{ji}$.

Again, we distinguish between the non-blocking and the blocking case. Furthermore, each of the four models described in Section 2.2 can be used to describe the cyclic behaviour. In this part we will only consider the "cyclic job-shop problem" model.

Problems without Blocking

The transformation from the non-cyclic definition in Section 2.1.4 to the cyclic problem is analogous to the transition we presented in Section 2.2. Therefore, we will only give a brief introduction of the necessary constraints.

In addition to $S_i(r)$, let $T_i(r)$ be the point in time, when transport move $\tau_i(r)$ for $i \in \Omega^\star$ starts. Again, we assume that every machine M_k has a sufficient large buffer B_k to store a job after the robot has loaded it onto the machine, or after its processing on the machine has been finished.

The problem can be summarised as follows.

$$\min \ \alpha \tag{2.33}$$

s.t.

$$S_i(r) = S_i(0) + \alpha r \quad i \in \Omega^\star; r \in \mathbb{Z} \tag{2.34}$$

$$T_i(r) = T_i(0) + \alpha r \quad i \in \Omega^\star; r \in \mathbb{Z} \tag{2.35}$$

$$S_i(r) + p_i \leq S_i(r+1) \quad i \in \Omega^\star; r \in \mathbb{Z} \tag{2.36}$$

$$S_i(r) + p_i \leq T_{suc(i)}(r) \quad i \in \Omega^\star; r \in \mathbb{Z} \tag{2.37}$$

$$T_i(r) + t_i + e_{i,pre(i)} \leq T_i(r+1) \quad i \in \Omega^\star; r \in \mathbb{Z} \tag{2.38}$$

$$T_i(r) + t_i \leq S_i(r) \quad i \in \Omega^\star; r \in \mathbb{Z} \tag{2.39}$$

$$\begin{aligned} & T_i(r_i) + t_i + e_{i,pre(j)} \leq T_j(r_j) \quad i,j \in \Omega^\star, i \neq j \\ \text{or} \quad & T_j(r_j) + t_j + e_{j,pre(i)} \leq T_i(r_i) \quad r_i, r_j \in \mathbb{Z} \end{aligned} \tag{2.40}$$

$$\begin{aligned} & S_i(r_i) + p_i \leq S_j(r_j) \quad i,j \in \Omega^\star; r_i, r_j \in \mathbb{Z} \\ \text{or} \quad & S_j(r_j) + p_j \leq S_i(r_i) \quad i \neq j; M(i) = M(j) \end{aligned} \tag{2.41}$$

A schedule $S = (T_i(r), S_i(r))$ with $i \in \Omega^\star$ is called *cyclic* with cycle time α if (2.34)-(2.41) are fulfilled.

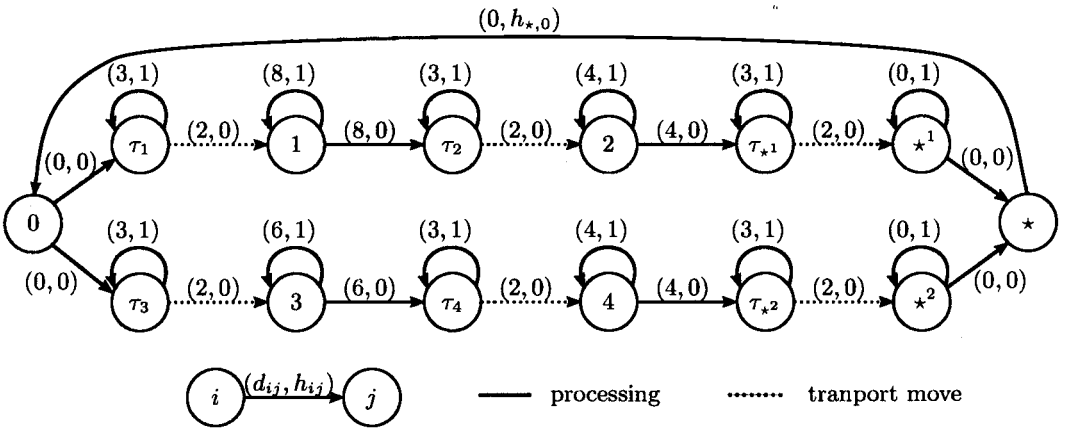


Figure 2.23: Graph for Example 2.2.6

Example 2.2.6. *Consider a cyclic job-shop problem with $N = 2$ jobs and $m = 2$ machines. Both jobs consist of 2 operations. The processing times and the machine allocations are given in the following table.*

Job	J_1		J_2	
Operation	1	2	3	4
Processing time	8	4	6	4
Machine	M_1	M_2	M_1	M_2

Again, every machine M_k has a large enough buffer B_k to store the jobs. Additionally, we have one transport robot. The transportation time for all operations $i \in \Omega^\star$ is set to $t_i = 2$, whereas the empty moving time between any two machines is given by $e_{ij} = 1$. Figure 2.23 shows the basic graph for this example.

By setting the height of the arc from $\star$ to 0 equal to 1, the minimal cycle time is $\alpha = 25$. Increasing the height up to 2, the cycle time decreases to $\alpha = 15$. Possible schedules for these solutions can be found in Figure 2.24.

Most of the work in the area of cyclic job-shop scheduling with transportation is focused on the blocking situation which we are going to describe in the following section.

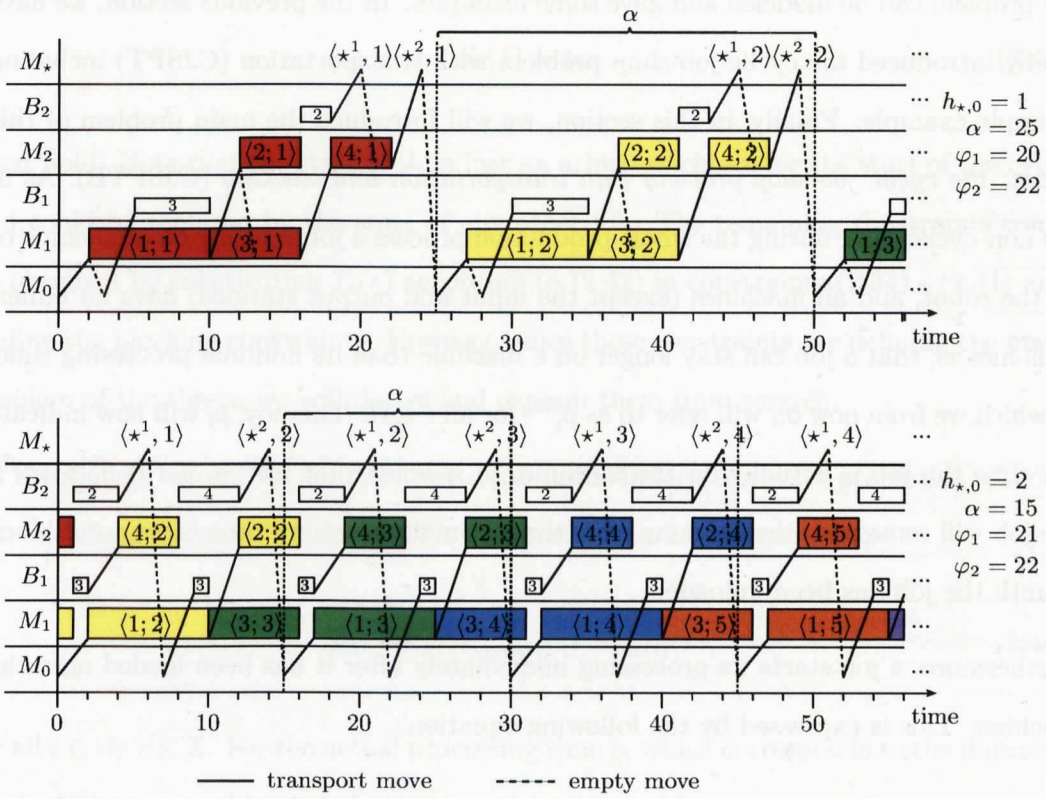


Figure 2.24: Schedules for Example 2.2.6

2.2.5 The Cyclic Job-Shop Problem with Transportation and Blocking

Recapitulating our discussion so far, we have introduced the classical non-cyclic job-shop problem with additional constraints in Section 2.1 and have also included the use of a transport robot. After that, we presented different ways of how a cyclic version of the problem can be modeled and gave some examples. In the previous section, we have briefly introduced the cyclic job-shop problem with transportation (CJSPT) including a simple example. Finally, in this section, we will introduce the main problem of this thesis: *the cyclic job-shop problem with transportation and blocking* (CJSPTB). As in the non-cyclic case, during the whole production process a job is either on a machine or on the robot, and all machines (except the input and output stations) have no buffer. This means, that a job can stay longer on a machine than its minimal processing time, to which we from now on will refer to as $p_i^{\min}$ for all $i \in \Omega$. Thereby, p_i will now indicate the time the job is actually on the machine, so it holds that $p_i^{\min} \leq p_i$. If $p_i > p_i^{\min}$, the job will remain on the machine after the minimal processing time is over and block it until the job has been unloaded.

Furthermore, a job starts its processing immediately after it has been loaded onto the machine. This is expressed by the following equation

$$\begin{aligned} & T_i(r) + t_i = S_i(r) \\ \Leftrightarrow \; & T_i(r) = S_i(r) - t_i, \end{aligned} \tag{2.42}$$

for all $i \in \Omega^\star$ and $r \in \mathbb{Z}$. That means, a solution for the CJSPTB can be presented as a vector including the starting times of the processing or the starting times of the transport moves for the operations. We will use the former version, which means a schedule S for the CJSPTB is represented by a vector $S = (S_i(r_i))$ for all $i \in \Omega^\star, r_i \in \mathbb{Z}$ including the starting times of each operation and their repetition numbers in an

arbitrary time interval (*cycle*) of length α. We recall, that in any cycle every operation starts and finishes exactly once. Note that there are indefinite many schedules, that have the same cycle time but different starting times for the operations. Hence, we can assume without loss of generality that at the beginning of a cycle, the robot is at time 0 at the input station M_0 and starts with transporting operation $\langle 1, 1\rangle$ to its machine. Thus,

$$S_1(1) = t_1 \tag{2.43}$$

must hold. Note that the start of J_1 is just an arbitrary choice for the start of the cycle and could be replaced by the start of any other job. The remaining constraints could be obtained by substituting $T_i(r)$ according to (2.42) in constraints (2.34) - (2.41) and adding the blocking constraints. However, since these constraints are defining the main problem of the thesis, we will derive and explain them from scratch.

As mentioned before, we assume that a job immediately starts its processing after it has been loaded onto a machine. This convention can be formulated as

$$S_i(r) + p_i + t_{suc(i)} = S_{suc(i)}(r), \tag{2.44}$$

for all $i \in \Omega$; $r \in \mathbb{Z}$. For the actual processing time p_i which corresponds to the duration a job stays at a machine it holds

$$p_i^{\min} \leq p_i, \tag{2.45}$$

for all $i \in \Omega$. Constraints (2.44) and (2.45) also ensure, that an operation has to be processed at least for its minimal processing time before its succeeding operation can start.

We also enforce, that w.l.o.g. the $(r+1)$-th repetition of operation $i \in \Omega$ cannot start

before the r-th repetition of it has been finished and transported to the next machine $M(suc(i))$. After that, the robot has to move to $M(pre(i))$ and repeat the transport move τ_i. Therefore, we get the following constraint:

$$S_{suc(i)}(r) + e_{suc(i),pre(i)} + t_i \leq S_i(r+1), \tag{2.46}$$

for all $i \in \Omega$, $r \in \mathbb{Z}$. For $i \in \{\star^1, \ldots, \star^N\}$ constraint (2.46) changes to

$$S_i(r) + e_{i,pre(i)} + t_i \leq S_i(r+1), \tag{2.47}$$

for all $r \in \mathbb{Z}$, since these operations do not have a successor.

Now, consider two operations i, j with $M(i) \neq M(j)$. As there exists no storage at the machines, operation j can only start its processing after $J(j)$ has been loaded onto the machine. In case i starts its processing immediately before j, the robot has to finish the loading of job $J(i)$, drive empty to the machine on which the predecessor $pre(j)$ of operation j is processed, unload the job, transport it to $M(j)$ and load it onto that machine. Depending on the order of processing the r_i-th repetition of i and the r_j-th repetition of j, we have

$$\begin{aligned} &S_i(r_i) + e_{i,pre(j)} + t_j \leq S_j(r_j) \\ \text{or } &S_j(r_j) + e_{j,pre(i)} + t_i \leq S_i(r_i), \end{aligned} \tag{2.48}$$

for all $i, j \in \Omega^\star$ and $r_i, r_j \in \mathbb{Z}$ with $M(i) \neq M(j)$. Note that these constraints (and the following ones as well) are only valid if the triangle inequality for the empty moving times holds.

For two operations $i, j \in \Omega$, that have to be processed on the same machine $M(i) = M(j)$, we have to decide, which job has to be processed first on the machine. Let us assume that i will be processed before j. Because of the blocking situation it follows,

that after the processing of $J(i)$ has been finished the robot first has to transport $J(i)$ to the next machine, before it can drive empty to $M(pre(j))$ to transport $J(j)$ to $M(j)$. Otherwise, $M(i)$ would be blocked by $J(i)$, and we would have a deadlock situation. Therefore, one of the following constraints must hold:

$$\begin{aligned} S_{suc(i)}(r_{suc(i)}) + e_{suc(i),pre(j)} + t_j &\leq S_j(r_j) \\ \text{or } S_{suc(j)}(r_{suc(j)}) + e_{suc(j),pre(i)} + t_i &\leq S_i(r_i), \end{aligned} \tag{2.49}$$

for all $i, j \in \Omega$ and $r_i, r_j, r_{suc(i)}, r_{suc(j)} \in \mathbb{Z}$ with $i \neq j;\ M(i) = M(j)$.

Finally, the constraint that each operation should start every α time units can be formalised as

$$S_i(r) = S_i(0) + \alpha r, \tag{2.50}$$

for all $i \in \Omega^\star,\ r \in \mathbb{Z}$.

We briefly want to recall, why constraints (2.43)-(2.50) are not just necessary, but also sufficient to define our problem. The question is, if a schedule, that fulfills these constraints is also a feasible one. First of all, the minimal processing and transportation times have to maintained. The processing times are trivially fulfilled by (2.44) and (2.45). The robot also has always sufficient time drive to a machine, pick up a job and transport it to its next machine, and a job does not start before it has been transported to its machine. This is given by constraints (2.48) and (2.49). The latter one also ensures, that a machine is always free before another job will be loaded onto it, so no two jobs can overlap. Finally, it is not possible to miss out a repetition of a job, since (2.46) and (2.47) are satisfied.

As with all the other cyclic problems there are different ways of modeling the CJSPTB. We will again use the second model from Section 2.2.2, the "cyclic job-shop problem"-model for an example. Thus, we have a general height $h^{\max} = h_{\star,0}$ which limits the

number of all different job repetitions in one cycle.

Example 2.2.7. *We consider the same data as in Example* 2.2.6. *The graph for this problem can be found in Figure* 2.25. *Note that the backwards arcs between an operation and its predecessor are caused by constraint* (2.46) *and replacing the loops from the node to itself are redundant. Setting the height* $h_{\star,0}$ *to* 1, *the minimal cycle time is* $\alpha = 30$. *Increasing the height to* 2, *the cycle time decreases to* $\alpha = 24$. *Possible schedules for these solutions can be found in Figure* 2.26.

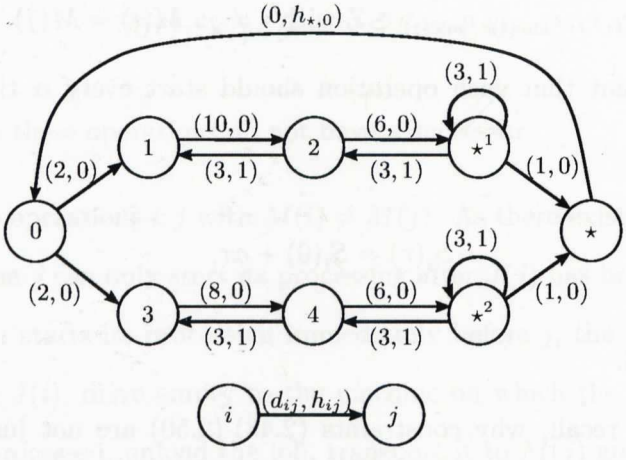


Figure 2.25: Graph for Example 2.2.7

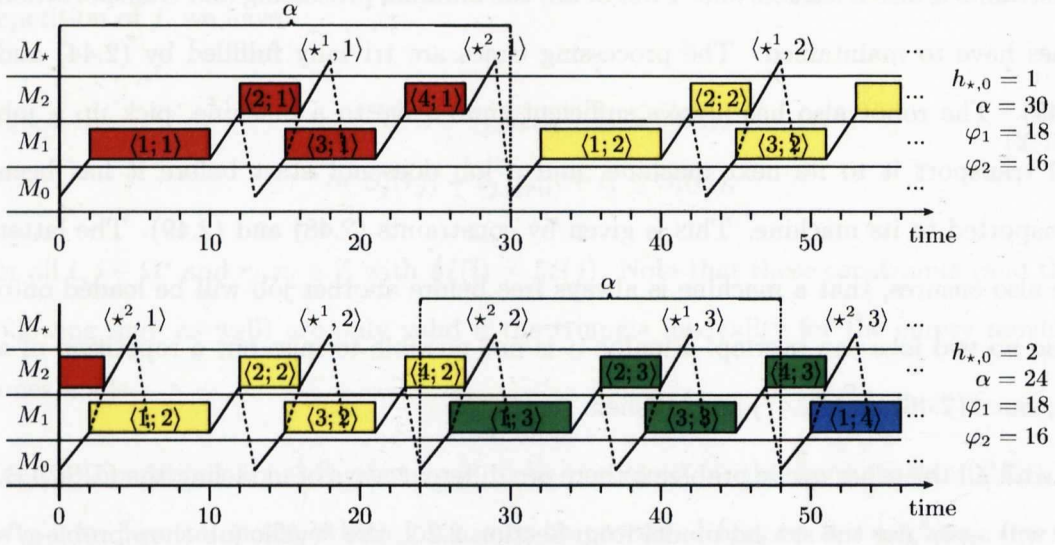


Figure 2.26: Schedules for Example 2.2.7

2.2.6 Literature Review

Cyclic scheduling problems have, in addition to mass production, other applications, such as compilation of loops for parallel computers, hoist routing in electroplating facilities, the design of embedded architectures or network scheduling. Different models have been proposed to handle those kinds of problem. Trouillet et al. (2007) and Chrétienne (1985) investigated the use of petri nets whereas Baccelli et al. (1992); Cohen et al. (1989) applied Max-plus algebras to the problem. Probably the most common approach, and also the one we will apply in this work, is modeling the problem using graphs (cf. Hanen and Munier (1995); Kats and Levner (1998a)). A good overview of cyclic scheduling can be found in Robert and Vivien (2009).

The range of areas in which cyclic scheduling problems occur is wide. Many researchers investigated the problem of software pipelining or scheduling on parallel processor. Further details for those applications can be found in Artigues et al. (2010); Calland et al. (1998); Eichenberger and Davidson (1997); Gasperoni and Schwiegelshohn (1994); Sucha et al. (2004).

The area we are studying in this work can be characterised as production scheduling where a set of products has to pass several machines in specific orders. And the corresponding production plan has to be repeated over and over again, as it is the case in large scale productions.

As already mentioned, it is easy to determine the minimal cycle time and a feasible corresponding schedule of a cyclic job-shop problem if the maximal flow time (or the height) is unbounded. Thus, this problem has not reached much attention in the literature. However, determining an optimal schedule that has minimal cycle time on the one hand and according to this cycle time also minimal flow-time on the other hand is of greater interest. Chauvet et al. (2003) define a criteria to determine if a schedule is optimal according to this definition. Furthermore, they present a construction

method based on an event graph model to generate good schedules with small flow time. Caggiano and Jackson (2008) developed schedule construction and improvement techniques to provide an effective solution approach for producing cyclic schedules with minimum weighted flow time.

The general cyclic job-shop model we have introduced is very similar to the model of the non-cyclic job-shop problem and is used by Hanen (1994) and Brucker and Kampmeyer (2005). Hanen (1994) also proved that the sequencing problem for general cyclic job-shops is strongly NP-hard. Other complexity results for cyclic scheduling problems can be found in McCormick and Rao (1994) and Levner et al. (2010).

Brucker and Kampmeyer (2005) introduced the cyclic job-shop problem with job repetition model. The probably most discussed model in the literature however, is the cyclic job-shop problem with machine repetition model. Additionally several authors assume that the number of jobs to be produced could be different for each job. For this model, the *minimal part set* (MPS), which indicates the ratio of the amount of produced jobs, has been introduced. For example, let us consider 3 jobs J_1, J_2 and J_3. We want to produce 100 parts of J_1, 300 parts of J_2 and 200 parts of J_3. Then, we get an MPS of $(1J_1, 3J_2, 2J_3)$ which has to be repeated 100 times. There are several ways of choosing or setting up an MPS and usually this is done in advance (cf. Lee (2000)). The aim is to find for a given MPS a periodic schedule with minimal cycle time. In the most discussed cases in the literature the height h_M is set equal to 1. Several complexity results for $h_M = 1$ can be found in Hall et al. (2002). They present a polynomial time algorithm for determining an optimal schedule for cyclic job-shop problems in which each job has at most two operations. They also solved an important open question discussed in Lenstra et al. (1977): For a job-shop problem with $n \geq 5$ operations, the decision version of the job-shop problem with makespan objective is strongly NP-complete. Furthermore, they show that the cycle time minimisation problem with $m \geq 3$ machines and $n \geq 3$ operations is also strongly NP-hard. For the special case

with two machines and two operations, Hall et al. (1998) provided a polynomial time algorithm. Most of the authors like McCormick and Rao (1994), Lee and Posner (1997) and Lee (2000) who considered this problem assume that each machine processes all of its operations for the i-th MPS before it can start the processing of any operation of the $(i+1)$-th MPS. Contrary to this, Caggiano and Jackson (2002) presented a model without this restriction.

For the general cyclic job shop problem with blocking Brucker and Kampmeyer (2008b) developed a tabu search algorithm and presented computational results. The same problem with an additional no-wait constraint is studied by Hall and Sriskandarajah (1997). Kamoun and Sriskandarajah (1993) review algorithmic and complexity issues for this problem. In Roundy (1992) an exact method for the special case with only one single job is presented. McCormick et al. (1989) considered the problem with blocking constraints and limited buffers where blocking only occurs when buffers are full. They proposed heuristic approaches to this problem based on an equivalent maximum flow problem and critical path techniques.

A cyclic production environment where a robot is in charge of transporting the jobs between the machines are often called *robotic cells* in the literature. Robots have increased the production capabilities in manufacturing by making the assembly process faster, more efficient and reliable than ever before. They also save workers from doing monotonous work on an assembly line. The literature distinguishes between three major robotic cell environments: the *cyclic robotic flow-shop*, the *cyclic robotic job-shop* and the *multiprocessor cyclic scheduling*.

The robotic flow-shop is probably the most well studied case. A flow-shop problem is a special version of the job-shop problem in which every job has the same order in which it has to be processed on the machines. The first research in this area dates

back to the early 1960s. Aizenshtat (1963); D.A. Suprunenko and Metelsky (1962) and Tanaev (1964) investigated cyclic processes in manufacturing lines served by transport devices. Within their work, they have introduced the cyclic robotic flow-shop problem and already suggested the *method of forbidden intervals*, which since then has been further developed by author such as Livshits et al. (1974), Kats et al. (1997, 1999), Che and Chu (2005a,b), Che et al. (2003) and Chu (2006). For more details in this area, we refer to Hall (1999), Crama et al. (2000), Dawande (2007); Dawande et al. (2005b).

On the other hand, the general cyclic job-shop problem with transportation or robotic job-shop is rarely discussed in the literature. Even with a single part MPS and a single robot the problem of scheduling the robot operations to minimise the cycle time is known to be NP-hard in the strong sense (cf. Livshits et al. (1974), Lei and Wang (1989)). In Kampmeyer (2006) and Brucker and Kampmeyer (2008a), a mixed linear integer programm as well as a tabu search for the cyclic job-shop problem are presented. This can be applied to the problem with transportation and also blocking.

Finally, multiprocessor cyclic scheduling differs from the robotic job-shop problem in the way, that the operations are only partially ordered, and that there are sufficient machines (including parallel ones) to process the jobs, such that the problem of finding the optimal sequence on a specific machine is left out. The problem definition was introduced by Romanovskil (1967). For an overview of these types of problems, we refer to Hall (1999) and Crama et al. (2000).

Furthermore, there are variations of cyclic scheduling problems. In this work, we only consider problems in which exactly one instance of every job enters and leaves the production process in a cycle. In a relaxed version of this definition, which is called k-cyclic scheduling, exactly k instances of a job are allowed to start and finish in a cycle. It is known that an optimal multi-unit cyclic solution can be better than an optimal 1-unit

cyclic solution. Dawande et al. (2005a) present some upper bounds on the difference in the per unit cycle times between an optimal multi-unit cycle and an optimal 1-unit cycle. Other results can be found in Kats and Levner (2002); Kats et al. (1999). The special case of 2-cyclic robotic scheduling with a single product has been investigated by Che et al. (2003), Chu (2006) and Kats and Levner (2011).

Another problem, which has reached a lot attention in the literature, is represented by cyclic robotic scheduling problems with blocking, in which the route the robot takes is given in advance. The problem is sometimes named the *cyclic hoist scheduling problem.* It has been shown to be polynomial solvable by various different approaches (cf. Matsuo (1990), Lei (1993), Ioachim and Soumis (1995), Ng and Leung (1997) and Lee (2000)). Another problem, which we will explain in more detail in the next chapter, has been developed throughout various publications. The main idea concerns a critical path algorithm, which is applied to a network with parametrical arc length. It has been developed in Kats and Levner (1998a). Several applications based on this algorithm are published in Kats and Levner (1998b) or Alcaide et al. (2007).

Chapter 3

The CJSPTB for a Fixed Robotic Cycle

Introduction

In cyclic scheduling problems, the height is one of the most important parameters. It distinguishes such problems from non-cyclic problems. However, many authors, just treat it as a standard parameter and do not discuss possible meanings and interpretations of it. As we have seen before, one can think of cyclic scheduling as finding a schedule for a minimal part set and then repeating this schedule all over again. However, at the end of one repetition and at the beginning of the next one, there often are only a few machines occupied by a job. Especially in the case of the CJSPTB, since we only have on robot available to unload the machines at the end of a cycle. Thus, we try to improve our schedule, by slightly shifting two

adjacent repetitions of a schedule into each other (cf. schedules in Figure 2.26). The parameter which on one hand allows those improvement to certain levels is the height. Within Section 3.1, we will give various different interpretations of the height in different cyclic scheduling models, that can be used to analyse and influence the solution for both theoretical and practical purposes.

A special sub problem of the CJSPTB arises, when the robotic cycle is given in advance. In Section 3.2, we will discuss the conditions under which a robotic cycle is feasible for a given problem and how to check this feasibility. Given a feasible robotic cycle, the problem simplifies in the sense that it becomes polynomially solvable. In practice, this situation is by far not exceptional. In circuit board printing, for instance, the robot is mounted on a fixed track and the route is predefined so that the robot is simply running along the track over and over again.

On a different note, it is very useful to have a fast procedure for solving the CJSPTB for a given robotic cycle in case you want to solve the general CJSPTB. Since branch and bound methods or heuristics need to evaluate many solutions during their optimisation process, it is of great interest to have a fast method performing this evaluation and therefore improve the solving time.

There are different algorithms in the literature, to calculate the cycle time when a feasible robotic cycle is provided. We shortly present two good algorithm from the literature in Sections 3.3 and 3.4. The first is a pseudo polynomial method which is supposed to perform well in practice and the other one is, to our knowledge, the fastest one in the literature based on computational complexity. In Section 3.5, we present a new algorithm, that

(except for some special cases) has a lower complexity than the ones in the literature. Since theoretical complexity can differ from the actual running times for solving real instances, we are also comparing our algorithm to the other two algorithms and we show that it also performs better on real computation. The computational results and a discussion about them are provided in Section 3.6.

3.1 Heights

Before we start discussing robotic cycles in more detail, we will give a short overview about the height parameter. So far, we have seen how different heights can influence the solution. However, the literature does not provide many interpretations of the height for the various models. Since the height is the main parameter in cyclic job shop problems, we will give some more insight to understand its meaning.

There is different information that one might like to know and be able to adjust in a production. For example: How many different job instances or repetitions are being processed at a time? Or, for how long will a specific job instance be in the production process? The height can, in a way, answer these questions.

In the previous chapter, we have looked at 3 different cyclic job-shop scheduling models, where everyone of those models has special height parameters. Let us recall the individual definitions of the heights for these different models.

- *Cyclic job-shop problem* with height $h_{\star,0}$: The $(r+h_{\star,0})$-th repetition of a job can only start if the r-th repetition of any job has finished.
- *Cyclic job-shop problem with job repetition* with heights h_{J_j}: The $(r+h_{J_j})$-th repetition of job J_j can only start if the r-th repetition of the same job has finished.
- *Cyclic job-shop problem with machine repetition* with heights h_{M_k}: The $(r+h_{M_k})$-th repetition of an operation on machine M_k can only start if the r-th repetition of all operations on the same machine have finished.

Example 3.1.1. *In Figure* 3.1, *the schedules of the Examples* 2.2.2 - 2.2.4 *for the different cyclic models are shown again. The heights are* $h_{\star,0} = h_{J_j} = h_{M_k} = 2$ *for* $j = 1,2$ *and* $k = 1,\ldots,4$. *In the first Gantt-chart, the first operation in the fourth*

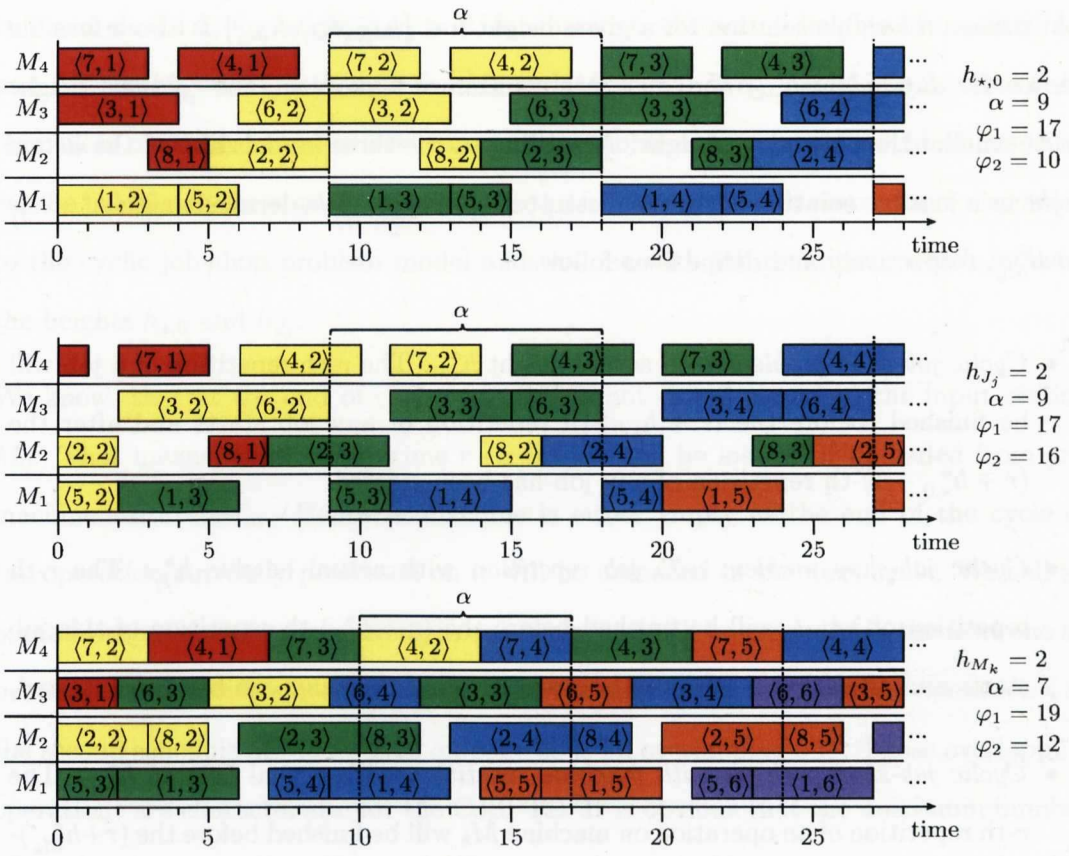


Figure 3.1: Schedules for Example 3.1.1

repetition is $\langle 1, 4\rangle$. *It starts after the last operation of the second repetition* ($\langle 4, 2\rangle$) *has finished. For* $h_{\star,0} = 2$, *this schedule is feasible.*

For the model with job repetition in the second schedule, the fourth repetition of job J_1 *starts at time* 11 *which is after the second repetition of* J_1 *has finished (at time* 10*). For job* J_2, *it also holds that the first operation* $\langle 5, 4\rangle$ *starts after the last operation* $\langle 8, 2\rangle$ *has finished. Thus, for* $h_{J_1} = h_{J_2} = 2$ *this is a feasible schedule.*

For the last model with machine repetition, the final schedule is also feasible. For instance, on machine M_4 *the first operation in its fourth repetition is* $\langle 7, 4\rangle$. *It starts after the last operation in the second repetition* ($\langle 4, 2\rangle$) *has finished.*

Note that according to these definitions, the height is considered to be an upper bound.

That means, a feasible solution for a given height $h \in \{h_{\star,0}, h_{J_j}, h_{M_k}\}$ is also a feasible solution for any height $h' \geq h$. Thus, in a solution of a problem, the "actual" height can be smaller than the given height. According to the three models above, the *actual height* in a feasible solution is the smallest possible height. We denote it with $h^*_{\star,0}, h^*_{J_j}$ and $h^*_{M_k}$, respectively and define it as follows.

- *Cyclic job-shop problem* with actual height $h^*_{\star,0}$: The r-th repetition of a job will be finished, before the $(r + h_{\star,0})$-th repetition of any job starts and after the $(r + h^*_{\star,0} - 1)$-th repetition of any job has been started.

- *Cyclic job-shop problem with job repetition* with actual heights $h^*_{J_j}$: The r-th repetition of job J_j will be finished before the $(r + h^*_{J_j})$-th repetition of this job starts and after the $(r + h^*_{J_j} - 1)$-th repetition of the same job has been started.

- *Cyclic job-shop problem with machine repetition* with actual heights $h^*_{M_k}$: The r-th repetition of an operation on machine M_k will be finished before the $(r+h^*_{M_k})$-th repetition of an operation on this machine starts and after the $(r+h^*_{M_k}-1)$-th repetition of any operation on this machine has been started.

An important fact about the height is, that it depends on the different repetition numbers of the operations in a cycle. For each job these repetition numbers can be increased or decreased and the solution would still be feasible (in any perspective except the height restriction, since the height might change). Thus, by convention the repetition numbers are chosen in a way, such that the actual heights are as small as possible. For the cyclic job-shop problem, one can simply give the first operation of each job the repetition number $h^*_{\star,0}$. For the one with job repetition the first operation of each job gets the repetition number equal to $h^*_{J_j}$ and for the one with machine repetition the first operation on machine M_k gets the repetition number equal to $h^*_{M_k}$. Procedure 3.2.1 in the next section shows, how the repetition numbers can be assigned.

In Example 3.1.1, for height $h_{M_k} = 2$, for instance, the actual heights of machine M_1 and M_2 are $h^*_{M_1} = h^*_{M_2} = 1$ and for M_3 and M_4 they are $h^*_{M_3} = h^*_{M_4} = 2$. Note that, regardless of which model is used to solve the problem, any actual height of the three can be calculated for a given solution. In the following, we will restrict our discussion to the cyclic job-shop problem model and the one with job repetition, which includes the heights $h_{\star,0}$ and h_{J_j}.

We know, that at the end of each cycle, the robot drives empty to the input station M_0. That means, that at any time $r \cdot \alpha$ no job will be loaded or unloaded from any machine $M_1, \ldots, M_m$. Hence, a machine is either empty at the end of the cycle or the operation currently processed on it will be unloaded in the next cycle. We call an operation that starts its processing in one cycle, and will be unloaded in the next one an *overlapping operation* since it 'overlaps' into the next cycle. For instance, operation 4 in the second schedule of Figure 2.26 on page 60 is an overlapping one. Those overlapping operations are characteristic for the CJSPTB. It is obvious that the maximum number of overlapping operations is $m - 1$. Every machine can have at most one overlapping operation, except the machine on which the first operation at the beginning of a cycle is processed. However, what is the connection between the height and the overlapping operations? This and other useful properties of the height are summarised in the following Theorems.

Theorem 3.1.1. *Consider job J_j in a feasible solution of a CJSPTB with cycle time α. If we consider an arbitrary cycle that starts with unloading the first operation of job J_j from the input station then the following statements are equivalent:*

1. *The actual height of the solution is $hs_{\star,0}$.*
2. *The number o_k of overlapping operations of each job J_k ($k = 1, \ldots, N$) is less than or equal to $h^*_{\star,0}$.*

Proof. The important assumption in this theorem is, that a cycle starts with unloading the first operation of J_j from the input station. This means that at the end of every cycle, the robot drives empty to M_0. The definition of the actual height $h^*_{\star,0}$ states that the r-th repetition of a job will be finished before the $(r+h^*_{\star,0})$-th repetition of any job starts and after the $(r+h^*_{\star,0}-1)$-th repetition of any job has been started. Assume that the cycle starts with unloading the r-th repetition of job J_j from the input station.

$1 \Rightarrow 2$: Consider a job J_k of the problem. Since in every repetition of a cycle one instance of each job starts and another one (can be the same) finishes, the number of overlapping operations must belong to different job repetitions. We distinguish between two different cases.

In the first one, let $J_j = J_k$. Then, the overlapping operation must have repetition numbers between $r-1$ and $r-o_k$. Hence, the $(r-o_k)$-th repetition of J_j will finish in this cycle. This is after the r-th repetition of J_j has been started and before the $(r+1)$-th repetition has been started. Therefore, the actual height will be at least $o_j + 1$.

For the second case, we consider $J_j \neq J_k$. Then we can always adjust the repetition numbers in a way, such that the first overlapping operation of J_k has the same repetition number as the first operation of J_j. Therefore, the instance of J_k finishing in that cycle, after the r-th repetition of J_j has been started and before the $(r+1)-th$ repetition will start has the repetition number $r-o_k+1$. Therefore the actual height will be at least o.

$2 \Rightarrow 1$: Assume that the number o of overlapping operations of J_k is greater than $h^*_{\star,0}$. As before, we set the repetition numbers of the first overlapping operation of J_k to $r-1$ if $J_k = J_j$ and to r otherwise. In the first case, the $(r-o)$-th repetition will finish in the current cycle and in the second case the $(r-o+1)$-th repetition. Thus the actual height would be $h^*_{\star,0} = o$ or $h^*_{\star,0} = o+1$. However, this is a contradiction to the assumption that $o > h^*_{\star,0}$.

□

Theorem 3.1.2. *Consider job J_j in a feasible solution of a CJSPTB with cycle time α. If we consider an arbitrary cycle that starts with unloading the first operation of job J_j from the input station, then the following statements are equivalent:*

1. *The actual height of job J_j is $h^*_{J_j}$.*
2. *The number of cycles job J_j is in the production process for is $h^*_{J_j}$.*
3. *The number of overlapping operations of job J_j is equal to $h^*_{J_j} - 1$.*
4. *The number of different instances of job J_j processed in one cycle is equal to $h^*_{J_j}$.*

Proof. We again have the assumption, that a cycle starts with unloading the first operation of J_j from the input station. Hence, at the end of every cycle, the robot drives empty to M_0.

$1 \Rightarrow 2$: According to the definition of the actual job height $h^*_{J_j}$, the r-th repetition of job J_j will be finished, before the $(r + h^*_{J_j})$-th repetition of job J_j starts and after the $(r + h^*_{J_j} - 1)$-th repetition has been started. In every cycle, a new instance of the job will start its processing. Let the r-th repetition of J_j start in cycle c. Then it finishes in the cycle in which the $(r + h^*_{J_j} - 1)$-th repetition starts. This is cycle $c + h^*_{J_j} - 1$. Thus, the job has been processed for $h^*_{J_j}$ cycles.

$2 \Rightarrow 3$: Let $i_1, \ldots, i_{n_j}$ be the operations of job J_j and T be the point in time when $\tau_{i_1}(r)$ will be executed. In the case that the r-th repetition of J_j is completed in the interval $[T, T + \alpha[$, then $h^*_{J_j} = 1$ holds per definition and there are obviously no overlapping operations. Note that the time interval $[T, T + \alpha[$ is open to the right hand side, since the robot will move empty to the input station at the end of the cycle and J_j will be finished before $T + \alpha$. It follows, that the number of overlapping operations is $h^*_{J_j} - 1 = 1 - 1 = 0$.

In the other case, the r-th repetition of the job will finish in an interval $[T+l\alpha, T+(l+1)\alpha[$ for $l = h^*_{J_j} - 1 \geq 1$. At the beginning of every time interval $[T+\mu\alpha, T+(\mu+1)\alpha[$ $(\mu = 1\ldots, l)$ the μ-th repetition of J_j will be transported from M_0 to $M(i_1)$. Therefore, the r-th repetition of J_j must be on a machine during this time and its corresponding operation is overlapping. This means that J_j has exactly $l = h^*_{J_j} - 1$ overlapping operations.

$3 \Rightarrow 4$: In the case that job J_j has no overlapping operations ($h^*_{J_j} = 1$), a specific repetition of the job will start and finish in the same cycle. Since each operation will start and finish exactly once in each cycle, there must be one instance of the job in every cycle. In the other case, J_j has $o_j \geq 1$ overlapping operations. Consider a cycle given by the time interval $[T, T+\alpha[$ in which the r-th repetition of J_j starts. If the job has an overlapping operation that finishes in the next cycle $[T+\alpha, T+2\alpha[$, then the same operation of the $(r-1)$-th repetition of J_j must finish in the current cycle. Thus, there is another instance of J_j in the current cycle. In general, if J_j has an overlapping operation that finishes in the cycle $[T+l\alpha, T+(l+1)\alpha[$ then the $(r-l)$-th repetition of this operation will finish in the time interval $[T, T+\alpha[$. Hence, for every overlapping operation there is an additional instance of the same job processed in a cycle. In conclusion, the number of different job instances in one cycle is equal to $h^*_{J_j}$.

$4 \Rightarrow 1$: Consider the cycle in which the r-th repetition of J_j starts its processing. Baring in mind, that there are $h^*_{J_j}$ different instances of the job in this cycle, the $(r - h^*_{J_j} + 1)$-th repetition will be finished within this cycle. For the last operation of the r-th repetition this means, it will finish $h^*_{J_j} - 1$ cycles later and within that cycle the $(h^*_{J_j} - 1)$-th repetition of J_j will start. Hence, the r-th repetition of J_j will be finished between the starts of the $(h^*_{J_j} - 1)$-th and $h^*_{J_j}$-th repetition of the same job, which is the definition of the actual height $h^*_{J_j}$. □

With this, we can also draw a conclusion about the height and the flow time of a job.

Corollary 3.1.1. *For the flow time φ_j and the actual height $h^*_{J_j}$ in a feasible solution for the CJSPTB it holds:*

$$(h^*_{J_j} - 1)\alpha \leq \varphi_j \leq h^*_{J_j}\alpha$$

for all $j \in \{1, \ldots, N\}$.

Let us consider another example.

Example 3.1.2. *We are given a CJSPTB with 2 jobs and 4 machines. Job J_1 only consists of 1 operation and J_2 has 3 operations. The processing times and machine allocations are given in the following table.*

Job	J_1	J_2		
Operation	1	2	3	4
Processing time	10	6	14	14
Machine	M_1	M_2	M_3	M_4

As before, the transportation time is $t_i = 2$ for all $i \in \Omega^\star$ and the empty moving time between any two different machines M_k, M_l is equal to 1. Figure 3.2 shows the same solution twice for the problem with height $h^*_{\star,0} = 3$ and cycle time $\alpha = 19$. However, in the first solution we start the cycle with unloading job J_1 from the input station and in the second with J_2. Note that these solutions are identical and just offsets of each other. The number of overlapping operations in the first schedule are $o_1 = 0$ and $o_2 = 3$ whereas in the second schedule they are $o_1 = 0$ and $o_2 = 2$. In both cases these numbers are less than or equal to $h^*_{\star,0} = 3$ (cf. Theorem 3.1.1). The actual heights of the two jobs in these solutions are $h^*_{J_1} = 1$ and $h^*_{J_2} = 3$. Job J_1 always has only one repetition in a cycle and no overlapping operations in both solutions. The interesting job in this example is J_2. As one can see, there are 4 different repetitions of this job in every tagged cycle of the first schedule and 3 in every cycle of the second one. However, to apply Theorem 3.1.2 to J_2 we need to consider the second schedule, since a cycle starts

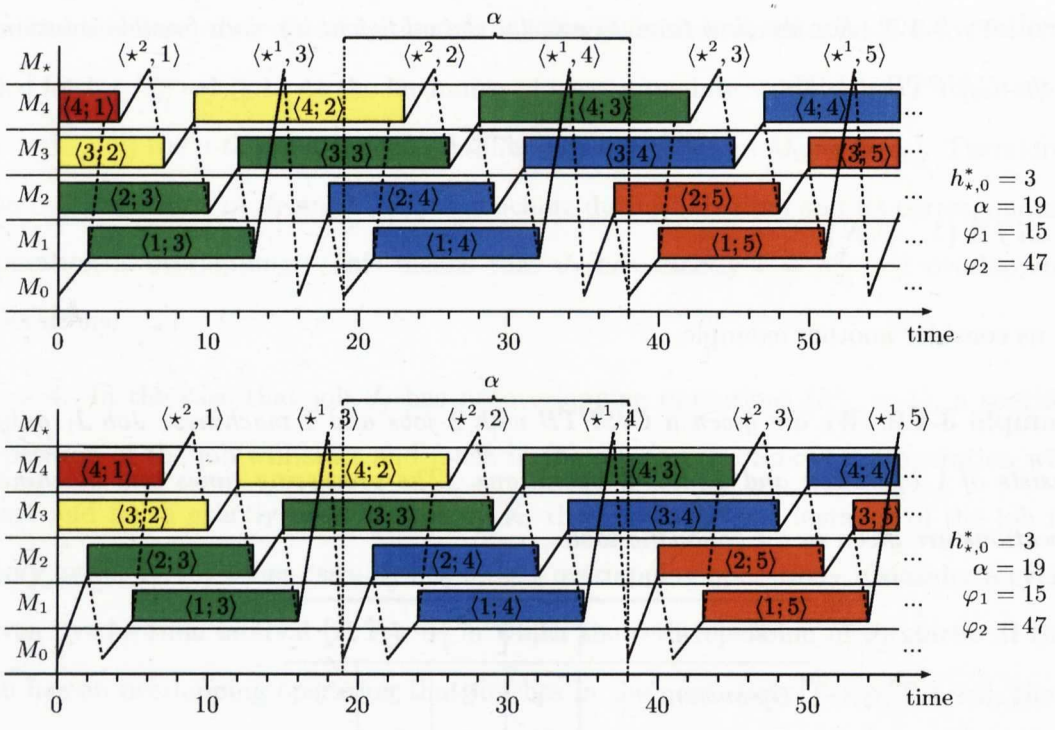


Figure 3.2: Schedule for Example 3.1.2

with unloading J_2 from M_0. Note that such a shift of the cycles can always be done to any job in any feasible solution. An instance of job J_2 is processed within 3 *(equal to $h^*_{J_j}$, cf. Theorem* 3.1.2, 2*) cycles, has* 2 *(equal to $h^*_{J_2} - 1$, cf. Theorem* 3.1.2, 3*) overlapping operations* 3 *and* 4 *and as already mentioned before, there are* 3 *(equal to $h^*_{J_j}$, cf. Theorem* 3.1.2, 4*) different repetitions of it in a cycle if the cycle starts with τ_2.*

For the connection between the flow time $\varphi_2 = 47$ and the cycle time $\alpha = 19$ it follows from Corollary 3.1.1 *that $(h^*_{J_2} - 1)\alpha = 38 \leq \varphi_2 \leq 57 = h^*_{J_2}\alpha$.*

3.2 Feasible Robotic Cycles

In this part, we will consider the CJSPTB with a given robotic cycle. Let us first of all have a brief discussion about how the problem changes if the robotic cycle is known in

advance. Assume that

$$R = \tau_{\sigma(1)}(r_{\sigma(1)}), \tau_{\sigma(2)}(r_{\sigma(2)}), \ldots, \tau_{\sigma(N+n)}(r_{\sigma(N+n)}),$$

with $\sigma : \Omega^\star \to \Omega^\star$ being a permutation of the set $\Omega^\star$, is the given robotic cycle. With a given order for the transport moves we can introduce further precedence constraints; firstly for the operations on each machine and secondly for the transport order based on the robotic cycle. Accordingly, let $pre^M(i)$ (respectively $suc^M(i)$) be the preceding (succeeding) operation of i processed on machine $M(i)$ and $pre^R(i)$ (respectively $suc^R(i)$) be the preceding (succeeding) operation of i in the robotic cycle. If $i = \sigma(N+n)$ is the last operation in R then we set $suc^R(i) = \sigma(1)$. To define an order between the operations in a robotic cycle, we write $\tau_i \prec \tau_j$ if τ_i precedes τ_j in R. Denoting the starting time of an operation i in a specific cycle by S_i^*, the CJSPTB with a given robotic cycle can be formulated as minimising the cycle time α subject to

$$S_i^* + p_i + t_{suc(i)} = \begin{cases} S^*_{suc(i)}, & \text{if } \tau_i \prec \tau_{suc(i)}; \\ S^*_{suc(i)} + \alpha & \text{else;} \end{cases} \qquad \forall\ i \in \Omega \tag{3.1}$$

$$S_i^* + e_{i,pre(suc^R(i))} + t_{suc^R(i)} \leq \begin{cases} S^*_{suc^R(i)}, & \text{if } \tau_i \text{ is not last in } R; \\ S^*_{suc^R(i)} + \alpha & \text{else;} \end{cases} \qquad \begin{array}{l} \forall\ i \in \Omega^\star \text{ with} \\ M(i) \neq M(suc^R(i)) \end{array} \tag{3.2}$$

$$S^*_{suc(i)} \leq \begin{cases} S^*_{suc^M(i)}, & \text{if } \tau_{suc(i)} \prec \tau_{suc^M(i)}; \\ S^*_{suc^M(i)} + \alpha & \text{else;} \end{cases} \qquad \forall\ i \in \Omega \tag{3.3}$$

$$p_i^{\min} \leq p_i \qquad \forall\ i \in \Omega \tag{3.4}$$

$$\alpha, p_i \geq 0 \qquad \forall\ i \in \Omega. \tag{3.5}$$

Since the order of the operations in the schedule depends on the given robotic cycle, the machine constraints also depend on R. Due to the no-wait constraint, equation (3.1) must hold. The cases, we have to distinguish between, are: S_i^* starts before its successor $S^*_{suc(i)}$ in the cycle or not. (This is equivalent to τ_i appearing before $\tau_{suc(i)}$ in R or not.) Note that it is not a contradiction to the precedence constraints if $\tau_{suc(i)} \prec \tau_i$, since

the operations would belong to different repetitions of the same job (e.g. $\langle i, r\rangle$ and $\langle suc(i), r-1\rangle$). The direct successor $\langle suc(i), r\rangle$ of $\langle i, r\rangle$ will start in the next repetition of the cycle, α time units later. Similarly, this can also be applied to the precedence constraints according to the robotic cycle or any machine. Constraint (3.2) ensures that the robot has enough time between delivering a job to a machine before picking up the next one in the cycle. To explain (3.3), let $suc(i) \neq suc^M(i)$. Thus, $suc(i)$ will not be processed on $M(i)$. Since $M(i)$ cannot be blocked at the time when $suc(i)$ starts and due to the fact that there is only one transport robot, constraint (3.3) must hold. Finally, we have to ensure that each operation will be processed at least for its minimal processing time (cf. (3.4)) and we do not allow any negative times (cf. (3.5)). Thus, constraints (3.1) - (3.5) are necessary to represent the CJSPTB with a fixed robotic cycle. What is often left out in a problem definition, is, that the described constraints are not just necessary but also sufficient to represent the problem. In our case, it is not difficult, since the problem is very compact, but we at least want to mention it. The 'critical' restrictions in this problem are, the minimal processing times, the blocking and the transportation constraints. If all of them are fulfilled, then the solution is feasible. And due to the discussion above all of these criteria are met. Note that constraints (3.1) - (3.5) would also be necessary for a problem, in which every machine needs some setup time between the processing of two jobs (e.g. cleaning the machine). However, it is not sufficient, since constraint (3.3) would have to be tightened in that case.

As we can see, with a given robotic cycle R there are no options available to decide which operation is processed first on a machine, or which job has to be transported next in a cycle. Looking back to the graph models, this means that by fixing the robotic cycle, every pair of disjunctive or alternative arcs has been fixed as well. Therefore, the problem is a lot easier to solve than the general CJSPTB. This also means, that a solution can be represented by a robotic cycle. However, a given robotic cycle for the CJSPTB is not necessarily a feasible one. Moreover, the number of feasible per-

mutations is way smaller than the number of infeasible ones. This results from two facts:

1. There are no buffers at the machines.
2. The overall height of the problem can be limited.

If there would be enough buffer space at every machine and a large enough value for the height would be allowed then any permutation of the transport moves would lead to a feasible robotic cycle. In order to decide whether a given robotic cycle would lead to a feasible solution, we define the following property.

Definition 3.2.1. *A robotic cycle R is called* blocking-feasible *if*

1. *before the robot executes operation τ_i, job $J(i)$ must be loaded and finished its processing on machine $M(pre(i))$ and*
2. *the robot is never required to transport a job to an already loaded machine.*

The following example shows a feasible and an infeasible robotic cycle for the same problem. Moreover, it implies a simple method to check whether a robotic cycle is blocking-feasible or not.

Example 3.2.1. *We will reuse the data of Example* 2.2.7. *Recapitulating, we had* 2 *jobs to be processed on* 2 *machines with the following data.*

Job	J_1		J_2	
Operation	1	2	3	4
Processing time	8	4	6	4
Machine	M_1	M_2	M_1	M_2

Consider the first cycle from time 0 *to* 24 *of the solution provided in Example* 2.2.7 *with height $h_{\star,0} = 2$ (cf. Figure* 2.26 *on page 60). This implies the following robotic*

cycle:

$$R_1 = \tau_1, \tau_{\star^2}, \tau_2, \tau_3, \tau_{\star^1}, \tau_4.$$

Using Table 3.1, *we can show that robotic cycle* R_1 *is feasible for the problem. The structure of the table is as follows. In column "move: from* $\rightarrow$ *to" the transport moves and the affected machines are shown, columns "*M_1*" and "*M_2*" show the operations that are currently loaded on the machines and column "r" shows how often this transport move has been executed. If a transport move is not possible, because the previous transport move has not been done yet, we write "n/a". Filling the table according to* R_1*, the robot starts with transporting job* J_1 *from the input station to its first machine* M_1*. Thus,* M_1 *is blocked by operation* 1. *The second move* $\tau_{\star^2}$ *cannot be executed because* $\tau_{pre(\star^2)} = \tau_4$ *has not been done yet. If we continue, we observe that before executing move* τ_i *operation* $pre(i)$ *is always available at* $M(pre(i))$ *and that the robot is never required to load an already loaded machine. That means, that* R_1 *is a blocking-feasible robotic cycle.*

On the other hand, Table 3.2 *shows an infeasible robotic cycle*

$$R_2 = \tau_1, \tau_{\star^1}, \tau_2, \tau_3, \tau_4, \tau_{\star^2}.$$

Before executing τ_4 *the first time, both machines* M_1 *and* M_2 *are blocked by operation* 3 *and* 2. *Transport move* τ_4 *wants to move job* J_2 *from machine* M_1 *to machine* M_2 *which is currently blocked by an instance of* J_1*. Thus, the robotic cycle* R_2 *is infeasible.*

The second property, that a feasible robotic cycle must fulfill is the height, depends on the different job repetitions in one cycle. Thus, for a given permutation of all transport moves τ_i with $i \in \Omega^\star$ we need to calculate the corresponding repetition number r_i in order to decide, whether the limit of the height is maintained or not. Procedure 3.2.1 calculates the repetition numbers and the actual job heights $h^*_{J_j}$ for a given robotic

move:	*from* $\rightarrow$ *to*	M_1	M_2	r
τ_1 :	$M_0 \rightarrow M_1$	1	–	1
$\tau_{\star^2}$:	$M_2 \rightarrow M_\star$	1	n/a	n/a
τ_2 :	$M_1 \rightarrow M_2$	–	2	1
τ_3 :	$M_0 \rightarrow M_1$	3	2	1
$\tau_{\star^1}$:	$M_2 \rightarrow M_\star$	3	–	1
τ_4 :	$M_1 \rightarrow M_2$	–	4	1
τ_1 :	$M_0 \rightarrow M_1$	1	4	2
$\tau_{\star^2}$:	$M_2 \rightarrow M_\star$	1	-	1
τ_2 :	$M_1 \rightarrow M_2$	–	2	2
τ_3 :	$M_0 \rightarrow M_1$	3	2	2
$\tau_{\star^1}$:	$M_2 \rightarrow M_\star$	3	–	2
τ_4 :	$M_1 \rightarrow M_2$	–	4	2

Table 3.1: Blocked machines in R_1

move:	*from* $\rightarrow$ *to*	M_1	M_2	r
τ_1 :	$M_0 \rightarrow M_1$	1	–	1
$\tau_{\star^1}$:	$M_2 \rightarrow M_\star$	1	n/a	n/a
τ_2 :	$M_1 \rightarrow M_2$	–	2	1
τ_3 :	$M_0 \rightarrow M_1$	3	2	1
τ_4 :	$M_1 \rightarrow M_2$	**-failed-**		
$\tau_{\star^2}$:	$M_2 \rightarrow M_\star$	M_2 is blocked		
τ_1 :	$M_0 \rightarrow M_1$			
$\tau_{\star^1}$:	$M_2 \rightarrow M_\star$			
τ_2 :	$M_1 \rightarrow M_2$			
τ_3 :	$M_0 \rightarrow M_1$			
τ_4 :	$M_1 \rightarrow M_2$			
$\tau_{\star^2}$:	$M_2 \rightarrow M_\star$			

Table 3.2: Blocked machines in R_2

cycle. (It can easily be adjusted to calculate any other height depending on the model.) The method works as follows. For every job J_j, the default height is set to $h^*_{J_j} = 1$ and the last operation $\star^j$ gets the repetition number $r_{\star^j} = 1$ (of course this can be any integer number). It then loops through the operations of the job in an descending order (line 4). If the successor $suc(i)$ is scheduled after operation i, then both must belong to the same job instance and therefore have the same repetition number. If not, the instance of operation i belonging to the same job as $suc(i)$ must have been scheduled in the previous cycle. Hence, the actual occurrence of i must belong to the next repetition of J_j, and the repetition number r_i as well as the height $h^*_{J_j}$ have to be increased by 1 (cf. line 5 - 8). We repeat this procedure for every job and finally return the repetition numbers and the heights (cf. 11).

Example 3.2.2. *We apply the described procedure to the robotic cycle R_1 from Example* 3.2.1. *Starting with the last operation $\tau_{\star^1}$ of job J_1 and assign repetition number $r_{\star^1} = 1$ to it. The job height is also set to $h^*_{J_1} = 1$. Entering the for-loop, the last operation of J_1 is $i = 2$ and its successor is $suc(i) = \star^1$. Since $\tau_{\star^1}$ is executed after τ_2, neither the height nor the repetition number is increasing. The same holds for the next operation $i = 1$. Thus, all repetition numbers of J_1 are equal to 1 and the actual job height $h^*_{J_1} = 1$ as well.*

Procedure 3.2.1 Determine repetition numbers and job heights

```
1: procedure determineRepetitionNumbers(R)
2:     for j = 1 to N do
3:         h*_{J_j} = 1 and r_{*j} = 1
4:         for i = last operation of J_j down to first operation of J_j do
5:             if τ_{suc(i)} precedes τ_i in R then
6:                 h*_{J_j} = h*_{J_j} + 1
7:             end if
8:             r_i = h*_{J_j};
9:         end for
10:    end for
11:    return r_1, ..., r_n and h*_{J_1}, ..., h*_{J_N}
12: end procedure
```

Considering job J_2 and starting again with $r_{\star 2} = h^*_{J_2} = 1$ we see, that $\tau_{\star 2}$ precedes its direct processor τ_4, and therefore we increase $h^*_{J_2}$ up to 2. The repetition number r_4 is also set to 2. Continuing with the algorithm the robotic cycle including repetition numbers is given by:

$$R = \tau_1(1), \tau_{\star 2}(1), \tau_2(1), \tau_3(2), \tau_{\star 1}(1), \tau_4(2).$$

The actual heights $h^*_{\star,0}$ and $h^*_{M_k}$ of the problem can easily be determined by looking at the repetition numbers of the corresponding operations. For instance, the second repetition of J_2 starts before the first repetition of J_1 has finished, so that $h^*_{\star,0} = 2$.

In order to find an optimal solution for the general CJSPTB (which is as we know NP-hard), most heuristics or branch and bound methods need to evaluate many sub optimal solutions during their execution. Since this number tends to be very large, it is of great interest to have a method that evaluates a determined solution as quickly as possible. That is, what this chapter is mainly about.

One of the methods we will analyse within this chapter is the fastest one tested in Dasdan et al. (1998) which is known as "Howard's Algorithm" and which was first

introduced in Howard (1960) (cf. Section 3.3). Furthermore, we will analyse a so called "Parametric Critical Path Algorithm" published in Alcaide et al. (2007) and updated in Kats et al. (2008) (cf. Section 3.4). Finally, in Section 3.5 we will then present a new algorithm and compare all three methods in Section 3.6.

3.3 Howard's Algorithm

The general idea of this algorithm is very simple. We start with a small enough value for the cycle time α and then continuously increase α until an optimality condition is fulfilled. We assume that a CJSPTB is represented by a graph $G = (V, E \cup A)$ as described in Section 2.2.4. Since the constraints (2.48) and (2.49) are, due to the feasible selection A, now fixed, the problem can be reformulated after building the corresponding graph. Every arc $(i,j) \in E \cup A$ is representing a type of constraint and consists of a delay d_{ij} and a height h_{ij}. Therefore, the problem can be summarised as

$$\min \ \alpha \tag{3.6}$$

s.t.

$$S_i(r) = S_i(0) + r\alpha \quad i \in V, r \in \mathbb{Z} \tag{3.7}$$

$$S_i(r) + d_{ij} \leq S_j(r + h_{ij}) \quad (i,j) \in E \cup A, r \in \mathbb{Z}. \tag{3.8}$$

By setting $S_i(0) = S_i$ for all $i \in V$ and a substitution of (3.7) into (3.8) we get the following formulation

$$\min \ \alpha \tag{3.9}$$

s.t.

$$S_i + d_{ij} - \alpha h_{ij} \leq S_j \quad (i,j) \in E \cup A. \tag{3.10}$$

The problem defined by (3.9)-(3.10) is a special case of the maximum cost-to-time ratio problem and an easy way to solve this problem is for instance applying the simplex method (cf. Dantzig et al. (1967)). However, there are other specialised algorithms that solve the problem much faster and one of them is Howard's algorithm. Before we start to discuss Howard's Algorithm, we want to recall some necessary and sufficient

conditions for the existence of a cyclic schedule from the literature. Furthermore, we conclude that the optimal cycle time can be computed by analysing the cycles in the graph G. The following results are based on the work of Kampmeyer (2006) which can be consulted for proofs and further details.

Let $\mu \subseteq E \cup A$ be a circuit in G. Then we define

$$d(\mu) := \sum_{(i,j)\in\mu} d_{ij}, \qquad h(\mu) := \sum_{(i,j)\in\mu} h_{ij} \qquad \text{and} \qquad v(\mu) := \frac{d(\mu)}{h(\mu)}$$

the *length*, the *height* and the *value* of μ respectively. A circuit with maximum value and positive height is called a *critical circuit.*

Theorem 3.3.1. *The problem described by* (3.9)-(3.10) *has a feasible cyclic solution with cycle time $\alpha > 0$ if and only if each circuit μ fulfills one of the following conditions:*

1. *The circuit μ has a positive height and arbitrary length.*
2. *The circuit μ has a negative height and a negative length.*
3. *The circuit μ has height zero and a non-positive length.*

And additionally the following inequalities

$$\min\left\{\frac{d(\mu)}{h(\mu)} \mid \mu \text{ is a circuit with } h(\mu) < 0\right\} \geq \alpha \geq \max\left\{\frac{d(\mu)}{h(\mu)} \mid \mu \text{ is a circuit with } h(\mu) > 0\right\}$$

have to hold.

Theorem 3.3.2. *Assume that the problem described by* (3.9)-(3.10) *has a feasible cyclic solution. Then the optimal cycle time is equal to the value of a critical circuit.*

The pseudocode of the algorithm is given in 3.3.1. Instead of using the complete graph $G = (V, E \cup A)$, the algorithm operates on a special subgraph $G_\sigma = (V, (E \cup A)_\sigma)$ of

G where $(E \cup A)_\sigma = \{(i, \sigma(i)) \mid i, \sigma(i) \in V\}$. The set $(E \cup A)_\sigma \subseteq E \cup A$ is updated in every iteration of the algorithm and thus the so called *policy graph* G_σ changes as well. The node set of this graph is the same as in G. However, every node i has exactly one successor $\sigma(i)$.

The algorithm can be divided into three parts. In the first one (cf. lines 1-5), the policy graph G_σ is initialised. Every node starts to be its own successor $\sigma(i) = i$. The distance label of every node $i \in V$ is denoted by $d(i)$ and initially set to $d(i) = p_i^{\min} + t_i$. The boolean variable 'improved' which is initialised in line 5 is used to indicate whether the current solution has changed and, if this is not the case, to stop the algorithm.

After initialising G_σ the graph consists of several disjunctive circuits. The next part of the algorithm is a while loop (lines 6 - 32) which can be divided into two parts. In the first part, we find all circuits μ in the graph and check with Theorem 3.3.1 if the problem has no feasible solution. If this is not the case, we set the cycle time α equal to the maximum value of a critical circuit in G_σ (cf. line 13). In the second part (lines 14 - 20), the graph is updated in a way, such that the critical circuit remains the only circuit in the graph. Furthermore, we recalculate the distance labels $d(i)$ for all nodes $i \in V$ (line 22). Within this line, the length of a longest path between two nodes, is the sum of the arc lengths contained in this path. Thereby, the length of an arc (i, j) does not only consists of its time lag d_{ij} but also of its height h_{ij}. In particular the following holds:

$$length((i, j)) = d(j) + d_{ij} - \alpha h_{ij}.$$

These labels are an estimation of the value of a longest path from i to the selected node s.

In the last part of the algorithm (lines 24-31), we check, whether the labels can be improved by changing the arcs in the graph G_σ. If they cannot be improved, the algorithm terminates and α is the minimal cycle time (line 33). Otherwise, the policy

graph has changed and we restart at the beginning of the first while loop (line 6) with the updated graph.

We want to show by an example how the algorithm works.

Example 3.3.1. *We will reuse the data of Example* 2.2.7 *on page 60. There are* 2 *jobs to be processed on* 2 *machines with the following data.*

Job	J_1		J_2	
Operation	1	2	3	4
Processing time	8	4	6	4
Machine	M_1	M_2	M_1	M_2

Transportation times were set to $t_i = 2$ *for all* $i \in \Omega^\star$ *and* $t_0 = t_\star = 0$. *The empty moving times were given by* $e_{ij} = 1$ *for all* $i, j \in \Omega^\star \cup \{0, \star\}$ *with* $i \neq j$ *and* $e_{ii} = 0$. *Again, consider the first cycle from time* 0 *to* 24 *of the solution provided in Example* 2.2.7 *with height* $h_{\star,0} = 2$ *(cf. Figure* 2.26 *on page 60)and the implied robotic cycle:*

$$R = \tau_1(2), \tau_{\star^2}(1), \tau_2(2), \tau_3(2), \tau_{\star^1}(2), \tau_4(2).$$

The fixed alternative arcs for this final solution are shown in Figure 3.3. *As one can see, the* h_{ij}*-values on the arcs are compensating the differences between the repetition numbers in the cycle. The arc set* $E \cup A$ *of the graph* G*, to which we will apply Howard's Algorithm, consists of both arc sets belonging to the graphs that are shown in Figures* 2.25 *and* 3.3.

Starting with the initialisation phase of the algorithm, the first policy graph G_σ^1 *is shown in Figure* 3.4(a). *The corresponding labels and successors are given in the table below.*

Node i	0	1	2	$\star^1$	3	4	$\star^2$	$\star$
$d(i)$	0	10	6	2	8	6	2	0
$\sigma(i)$	0	1	2	$\star^1$	3	4	$\star^2$	$\star$

Procedure 3.3.1 Howard's Algorithm

for all nodes $i \in V$ **do**
  set $d(i) := p_i^{\min} + t_i$;
  set $\sigma(i) := i$;
end for
improved := TRUE;
while improved is TRUE **do**
  $(E \cup A)_\sigma := \{(i, \sigma(i)) \mid i \in V\}$;
  Find all circuits in $G_\sigma := (V, (E \cup A)_\sigma)$;
  if G_σ has circuit μ with $h(\mu) < 0$ and $d(\mu) \geq 0$ or $h(\mu) = 0$ and $d(\mu) > 0$ **then**
    return infeasible;
  end if
  Let μ be the circuit with maximum value in G_σ and $h(\mu) > 0$;
  set $\alpha := v(\mu) = d(\mu)/h(\mu)$;
  Select node $s \in \mu$ with smallest index;
  $V_\mu := \{i \in V \mid$ there exists a path from i to s in $G_\sigma\}$;
  while $V_\mu \neq V$ **do**
    Find node $i \in V \setminus V_\mu$ s.t. there is a $j \in V_\mu$ and $(i, j) \in E \cup A$;
    set $\sigma(i) := j$;
    set $V_\mu := V_\mu \cup \{i\}$;
  end while
  for all $i \in V$ **do**
    update label $d(i)$ to length of the longest path from i to s in G_σ;
  end for
  set improved := FALSE;
  for all arcs $(i, j) \in E \cup A$ **do**
    if $d(i) < d(j) + d_{ij} - \alpha h_{ij}$ **then**
      set improved := TRUE;
      set $d(i) := d(j) + d_{ij} - \alpha h_{ij}$;
      set $\sigma(i) := j$;
    end if
  end for
end while
return α;

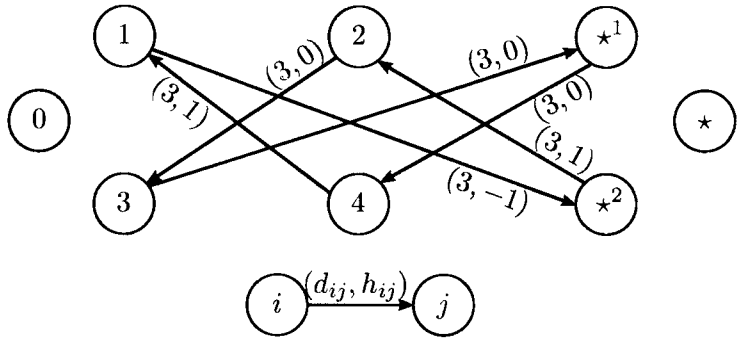


Figure 3.3: Arcs representing the robotic cycle for Example 3.3.1

The circuit with maximum value in G_σ^1 *is* $\mu = (1,1)$ *and emphasised by a dotted arc. Its value is* $v(\mu) = 10/1 = 10$, *which gives an initial minimal cycle time of* $\alpha = 10$ *and the only node in* μ *is* $s = 1$. *Then, the policy graph is updated in lines* 16 - 20 *and the new graph* G_σ^2 *is shown in Figure* 3.4(b). *Updating the labels and successors gives the following values:*

Node i	0	1	2	$\star^1$	3	4	$\star^2$	$\star$
$d(i)$	2	0	−7	−14	1	−7	−14	−20
$\sigma(i)$	1	1	1	2	4	1	2	0

For instance, the path from 3 *to* $s = 1$ *is* $3 \to 4 \to 1$. *Its length is* $8 + 3 = 11$ *and its height is* $0 + 1 = 1$. *Thus the label is set to* $d(3) = 11 - \alpha \cdot 1 = 11 - 10 = 1$. *Based on these labels and successors we update the graph again (lines* 24 - 31*) and the result is shown in Figure* 3.4(c). *The labels and successors have changed as follows:*

Node i	0	1	2	$\star^1$	3	4	$\star^2$	$\star$
$d(i)$	2	14	4	−3	1	−7	−3	−20
$\sigma(i)$	1	2	3	2	4	1	2	0

Returning to the beginning of the while-loop the critical circuit has changed to $\mu = (1,2,3,4,1)$ *which is again indicated by dotted arcs in Figure* 3.4(c). *The value of this circuit (and at the same time the new cycle time) is* $\alpha = 24/1 = 24$. *Applying the rest*

of the algorithm we see that the if-statement in line 26 *is never true and thus* $\alpha = 24$ *is returned as the optimal cycle time.*

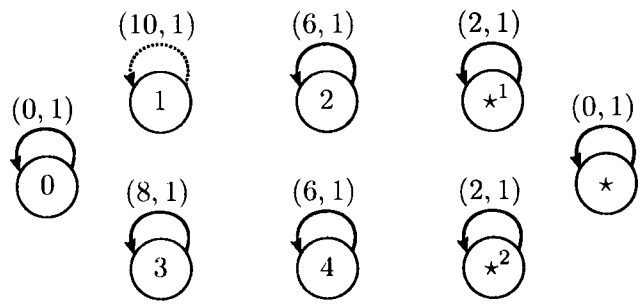
The running time for this version of Howard's Algorithm is pseudo-polynomial, which means it is polynomial in the *numeric* value of the input. (Note that computational complexity is measured in relation to the *size* of the input rather than the numeric value which is exponential in the size of the input). However, the running time can be summarised as

$$O\left(n^2 m \alpha^{opt} \frac{h(\mu^+)}{\varepsilon}\right),$$

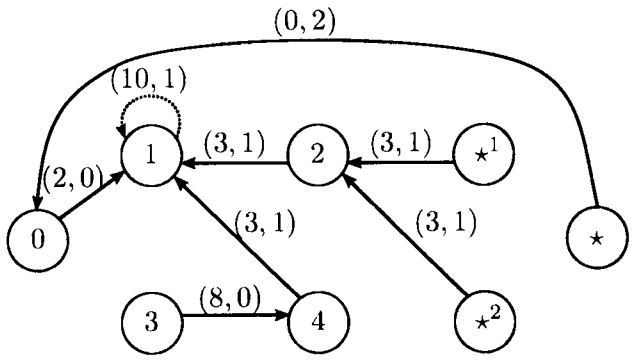
where α^{opt} is the minimal cycle time,

$$h(\mu^+) := \max\{h(\mu) \mid \mu \text{ is a circuit in } G \text{ without node repetition}\}$$

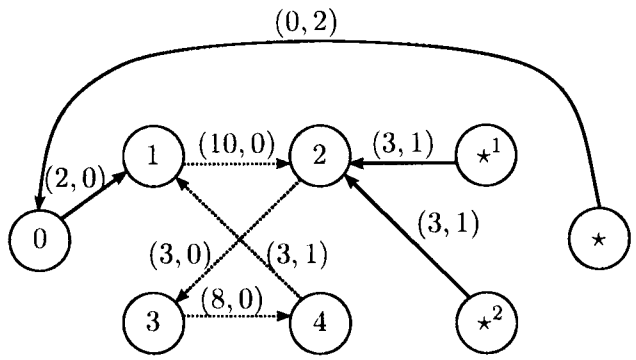
and $\varepsilon = d(i) - (d(j) + d_{ij} - \alpha h_{ij})$ is the maximal improvement during the label correction in lines 24 - 31. For further details and proofs we refer to Kampmeyer (2006). Even if the running time is pseudo-polynomial its practicality has already been shown in Dasdan et al. (1998) and for that reason, we included it in our study as well.



(a) Policy graph G_σ^1



(b) Policy graph G_σ^2



(c) Policy graph G_σ^3

Figure 3.4: Policy graphs of Example 3.3.1

3.4 A Parametric Critical Path Algorithm

The basic concept of this algorithm is a modification of the Bellman–Ford algorithm (cf. Bellman (1958)). It was developed by Kats and Levner and from then on it had some modification and was applied to different problem types (cf. Kats and Levner (1998a), Alcaide et al. (2007) and Kats et al. (2008)). However, the general idea of the algorithm always stayed the same.

Since this algorithm is based on a network algorithm, it also operates on a graph $G = (V, E \cup A)$. The node set $V = \{0\} \cup \Omega^\star$ consists of the dummy start node 0 and all operations in $\Omega^\star$ representing the starting times of these operations. Note that the dummy end operation $\star$ is not included in the node set. The arcs of the graph are representing minimum time lags between the starting points. Those time lags are based on the formulation in Section 3.2 given by (3.1) - (3.5). Substituting p_i in constraint (3.1) by (3.4), we get:

$$S_i + p_i^{\min} + t_{suc(i)} \leq \begin{cases} S_{suc(i)}, & \text{if } \tau_i \prec \tau_{suc(i)}; \\ S_{suc(i)} + \alpha, & \text{else.} \end{cases} \tag{3.11}$$

for all $i \in \Omega$. Thus, the length of the arcs representing the precedence constraints depends on the order of transport moves in the robotic cycle. If τ_i precedes $\tau_{suc(i)}$ in R then we add an arc from i to $suc(i)$ of length $p_i^{\min} + t_{suc(i)}$. Otherwise, the arc length from i to $suc(i)$ is going to be $p_i^{\min} + t_{suc(i)} - \alpha$. Note that those lengths can be affine functions rather than constant values, since they depend on the parameter α. This is also the reason, why the algorithm is called "parametric".

For the transport constraints the following holds:

$$S_i + e_{i,pre(suc^R(i))} + t_{suc^R(i)} \leq \begin{cases} S_{suc^R(i)}, & \text{if } \tau_i \text{ is not last in } R; \\ S_{suc^R(i)} + \alpha, & \text{else.} \end{cases} \tag{3.12}$$

for all $i \in \Omega^\star$ with $M(i) \neq M(suc^M(i))$. Those constraints can be represented by an arc from i to $suc^R(i)$ of length $e_{i,pre(suc^R(i))} + t_{suc^R(i)}$ in case τ_i is not last in R and one from the start of the last operation in R to 1 of length $e_{i,0} + t_1 - \alpha$.

Finally, for the machine constraints the following holds:

$$S_{suc(i)} \leq \begin{cases} S_{suc^M(i)}, & \text{if } \tau_i \prec \tau_{suc(i)}; \\ S_{suc^M(i)} + \alpha, & \text{else.} \end{cases} \tag{3.13}$$

for all $i \in \Omega$. Therefore, we add an arc from $suc(i)$ to $suc^M(i)$ of either length 0 or of length $-\alpha$ depending on the order of the transport moves. Note that these arcs are only relevant in case of a non-blocking-feasible robotic cycle. Otherwise, they are implied by the route the robot takes.

Finally, there is an arc from the dummy start node 0 to the first node in the cycle $(i = 1)$ of length t_1.

Example 3.4.1. *The graph for the data used in Example* 3.3.1 *is shown in Figure* 3.5. *The solid arcs are presenting precedence constraints and (except for one arc) are of length* $p_i^{\min} + t_{suc(i)}$. *The arc from* 4 *to* $\star^2$ *is of length* $6 - \alpha$ *since* $\star^2$ *precedes* 4 *in the robotic cycle. The dashed arcs are representing the transport constraints. They are all of length* $e_{i,pre(suc^R(i))} + t_{suc^R(i)} = 3$ *except the arc from* 0 *to* 1 *(which is of length* $t_1 = 2$*) and the one from the last operation* 4 *to the first operation* 1 *in the cycle (which is of length* $3 - \alpha$*).*

Bellman's and Ford's algorithm finds the single-source shortest paths in a weighted directed graph. For graphs with only non-positive arc lengths, the faster Dijkstra's algorithm also solves the problem. Thus, the Bellman-Ford Algorithm is primarily used for graphs with positive arc lengths. In the graph presented above, some arcs lengths are affine functions of the form $a - b\alpha$, and therefore, might become negative for a large enough value for b or α. Kats and Levner modified the Bellman-Ford Algorithm, so that

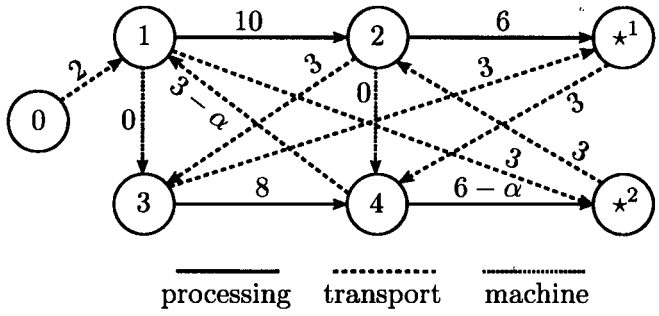


Figure 3.5: Graph for Example 3.4.1

it finds the longest paths in such a graph. The pseudo code of their Parametric Critical Path Algorithm (PCP) is shown in Procedure 3.4.1. Since it is a labeling algorithm, every node $i \in V$ has a distance label $d^u(i)$. The different superscripts u are indicating the different updates during the algorithm.

It starts with initialising the labels by setting the label of the source node 0 to $d^0(0) = 0$ and the ones for the remaining nodes i to $d^0(i) = -\infty$ (cf, lines 1-2). Then the label correction is carried out in lines 3 - 11. Within this, the algorithm alternates between two cases. It either considers the arcs from a node that precedes the current one in the robotic cycle or not. However, the ordering in the two cases could be chosen in any other way. Since this algorithm is searching for the longest critical path from the source node 0 to all other nodes, in every iteration, it updates the labels of every node i to the maximum of the current label $d(i)^u$ and all labels $d(j)^u + d_{ji}$, for which exist an arc (j, i) in G. Note that the labels are not just integer values, but sets of affine function depending on α. The label correction has been updated since its first version presented in Alcaide et al. (2007). Originally, the outer for-loop would have been executed $|V| - 1$ times and therefore one wouldn't distinguish between u being even or odd. Apparently, in Kats et al. (2008) the authors changed that to the current version, since it should save on average 50% of operations while computing the max operator. The origin of this idea is based on Yen (1970) who considered the problem with constant arc lengths.

After the label correction, one needs to find the feasible values for α that satisfy constraint (3.14). If there does not exist one, the problem is infeasible and otherwise the minimum feasible value for α is returned (cf. line 17).

Procedure 3.4.1 Parametric Critical Path Algorithm

1: $d^0(0) = 0$;
2: $d^0(i) = -\infty$ for all $i \in V \setminus \{0\}$
3: **for** $u = 0$ to $U - 1$, where $U = 2(|V| - 1)$ **do**
4: **for** $i = 1$ to $|V|$ **do**
5: **if** u is even **then**
6: **set** $d^{u+1}(i) := \max\{d(i)^u, \max_{\tau_j \prec \tau_i}\{d(j)^u + d_{ji}\}\}$;
7: **else**
8: **set** $d^{u+1}(i) := \max\{d(i)^u, \max_{\tau_j > \tau_i}\{d(j)^u + d_{ji}\}\}$;
9: **end if**
10: **end for**
11: **end for**
12: **for all** arcs $(i, j) \in E \cup A$ **do**
13: **solve** the system of functional relations

$$d^u(i) + d_{ij} \leq d^u(j) \tag{3.14}$$

with respect to α;
14: **end for**
15: Let T be the set of feasible values satisfying (3.14).
16: **if** $T \neq \emptyset$ **then**
17: **return** $\alpha^{opt} = \min\{T\}$;
18: **else**
19: **return** infeasible;
20: **end if**

The following example shows how the algorithm works.

Example 3.4.2. *We are going to apply the algorithm on the graph of Example* 3.4.1. *The labels after the initialisation are as follows*

Node i	0	1	2	$\star^1$	3	4	$\star^2$
$d^0(i)$	0	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

For $u = 0$ the labels do not change. In the second repetition of the label correction, the label of node 1 changes to $d^2(1) = 2$ since the label of 0 is $d^1(0) = 0$ and there exists an arc from 0 to 1 of length 2. The maximum of $\{-\infty, 2\}$ is obviously 2. After the repetition for $u = 6$ the labels have changed to:

Node i	0	1	2	$\star^1$	3	4	$\star^2$
$d^7(i)$	0	2	16	18	15	21	12

For the next iteration ($u = 7$) let us consider node 1 with a current label of $d^7(1) = 2$. Since u is odd, we only consider nodes having an arc to 1 that appear later in the robotic cycle. Operation 4 is the only one and the length of the arc from 4 to 1 is $d_{41} = 3 - \alpha$. Thus, the label of 1 is updated to

$$\begin{aligned} d^8(1) &= \max\{d(1)^7, \max\{d(4)^7 + d_{41}\}\} \\ &= \max\{2, \max\{21 + 3 - \alpha\}\} \\ &= \max\{2, 24 - \alpha\} \end{aligned}$$

Here we can see, how the maximum operator starts to increase the number of terms of the label. For every different coefficient of the variable α there will be another function added to the label. At the end of the label correction the final labels are as follows:

$$\begin{aligned} &d^{12}(0) = 0 && d^{12}(3) = \max\{15, 39 - \alpha\} \\ &d^{12}(1) = \max\{2, 26 - \alpha, 48 - 2\alpha\} && d^{12}(4) = \max\{23, 45 - \alpha\} \\ &d^{12}(2) = \max\{12, 36 - \alpha\} && d^{12}(\star^2) = \max\{5, 29 - \alpha, 51 - 2\alpha\} \\ &d^{12}(\star^1) = \max\{18, 42 - \alpha\} \end{aligned}$$

Solving constraints (3.14), the minimal value for the cycle time is $\alpha = 24$. The critical

nodes dominating this result are 3 *and* 4. *According to* (3.14) *we have*

$$\begin{aligned} & d^{12}(3) + d_{34} \leq d^{12}(4) \\ \Leftrightarrow\ & \max\{15, 39 - \alpha\} + 8 \leq \max\{23, 45 - \alpha\} \\ \Leftrightarrow\ & \max\{23, 47 - \alpha\} \leq \max\{23, 45 - \alpha\}. \end{aligned} \tag{3.15}$$

We also have

$$\max\{23, 47 - \alpha\} = \begin{cases} 23, & \text{for } \alpha \geq 24; \\ 47 - \alpha, & \text{else,} \end{cases}$$

and

$$\max\{23, 45 - \alpha\} = \begin{cases} 23, & \text{for } \alpha \geq 22; \\ 45 - \alpha, & \text{else.} \end{cases}$$

Hence, constraint (3.15) *holds for all* $\alpha \geq 24$. *Continuing to solve the remaining constraints of* (3.14), *the minimum cycle time is given by* $\alpha = 24$.

The running time of the PCP is $O(|V|^4)$ which in case of the CJSPTB is equivalent to $O(n^4)$ and therefore strongly polynomial (proofs can be found in Kats et al. (2008)). The algorithm also has the advantages of being able to handle time window constraints which would lead to negative time lags between some nodes.

3.5 A New Algorithm

In this section we will present a new algorithm that solves the CJSPTB with a given robotic cycle. The idea of the algorithm can be described as follows. Assume we are given a feasible robotic route for the non-cyclic job-shop problem with transportation and blocking (cf. Section 2.1.4). An earliest start schedule can be obtained as follows. We start with fixing the starting time of the first operation in the robotic route. Then, we successively consider every other operation associated with a transport move in the robot route and calculate its earliest possible starting time. Obviously, this depends on the precedence and robot constraints. If we use the same graph representation as before, then the earliest point in time a task i in the project can be started is equivalent to the length of a longest (critical) path from the starting task of the project to node i (cf. Shtub et al. (1994)). (The Parametric Critical Path Algorithm from the previous section also relies on the critical path method.) Therefore, in the non-cyclic problem, an earliest start schedule can be calculated in linear time. The same idea will be used in our approach. However, there is one problem in the cyclic problem, which makes it slightly more difficult to solve: the overlapping operations. Since such an operation consists of two processing parts (one at the beginning an one at the end of the cycle), the sum of them needs to be equal to the minimum processing time of the operation. Therefore, we cannot use the same critical path method as for the non-cyclic problem. Anyway, the basic idea is very effective and we will try make use out of it.

Since graphs or networks, as we have seen before, are common ways of representing production processes or any kind of projects we have based our new approach on a directed graph $G = (V, E \cup A)$ as well. The nodes are representing the starting times of all operations $i \in \Omega^\star$ and the arcs again minimal time distances between the start of the operations. Additionally to the operation nodes, there is a dummy start node 0

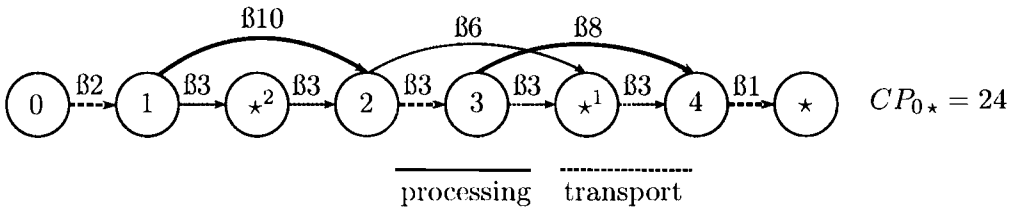


Figure 3.6: Graph representing robotic cycle for Example 3.5.1

and a dummy end node $\star$. Thus, $V = \Omega^\star \cup \{0, \star\}$. The arcs in graph G are representing those parts of constraints (3.1) to (3.3) which do not depend on α. We distinguish between the following arcs:

- for every precedence constraint $S_i + p_i + t_{suc(i)} = S_{suc(i)}$ together with $p_i^{\min} \leq p_i$ an arc from i to $suc(i)$ of length $d_{i,suc(i)} = p_i^{\min} + t_{suc(i)}$,
- for every robot constraint $S_i + e_{i,pre(suc^R(i))} + t_{suc^R(i)} \leq S_{suc^R(i)}$ an arc from i to $suc^R(i)$ of length $d_{i,suc^R(i)} = e_{i,pre(suc^R(i))} + t_{suc^R(i)}$,
- for every machine constraint $S_{suc(i)} \leq S_{suc^M(i)}$ an arc from $suc(i)$ to $suc^M(i)$ of length $d_{suc(i),suc^M(i)} = 0$.

Furthermore, we add an arc of length $t_{\sigma(1)}$ from dummy node 0 to the first operation in the cycle $\sigma(1)$ and one from the last operation in the cycle $\sigma(N+n)$ to $\star$ of length $d_{\sigma(N+n),\star} = e_{\sigma(N+n),0}$. If there exists a path from node i to node j then the length of a longest path from i to j is denoted by $CP_{i,j}$. We call a longest path from 0 to $\star$ *critical path* and an arc belonging to such a critical path is called *critical arc*. The following example will present the corresponding network for our already known problem.

Example 3.5.1. *Again, we use the same data as before (cf. Examples* 2.2.7, 3.3.1, 3.4.2*). Since we know that the robotic cycle* $R = \tau_1(2), \tau_{\star^2}(1), \tau_2(2), \tau_3(2), \tau_{\star^1}(2), \tau_4(2)$ *is blocking feasible we omit the arcs for the machine constraints given by* (3.3)*. The graph for this problem is shown in Figure* 3.6*. The nodes are ordered in a way that is*

analogous to the robotic cycle. Between every two nodes i and $suc^R(i)$ we have an arc of length 3 since $e_{i,pre(suc^R(i))} + t_{suc^R(i)} = 3$ for all $i \in \Omega^\star \cup \{0\}$. Furthermore, we have arcs from i to $suc(i)$ of length $p_i^{\min} + 2$ for all i where $\tau_i \prec \tau_{suc(i)}$. Additionally, there are two dummy arcs. One from 0 to 1 of length 2 and one from $\star^3$ to $\star$ of length 0. The critical path from 0 to $\star$ is $(0 \to 1 \to 2 \to 3 \to 4 \to \star)$ and emphasised through bold arcs in the graph. Its length is $CP_{0,\star} = 24$. Note that this length is a lower bound for the cycle time.

As we can see in the example, there are still some constraints missing in the graph. For instance, there is no arc representing the precedence constraints between operation 4 and $\star^2$. And those are the ones which depend on α or on the *overlapping* operations (cf. page 73). To get a better understanding of those operations, imagine a cyclic schedule of length α as a Gantt-chart of length α which is wrapped around a cylinder with perimeter equal to α. One starts with the transport move $\tau_1(1)$ of operation 1 to its machine and then moves around the cylinder. To get the consecutive schedule, one simply increases the repetition number every time the starting point of an operation is reached again. A single repetition of the cyclic schedule can be obtained by slicing through the cylinder at the beginning of τ_1 and unfolding it. Obviously, it can happen that an operation i is cut in two parts. Those operations are the overlapping ones. In our case, an overlapping relation can only be induced by a precedence constraint (constraint (3.1)). If we would assume that the cycle does not necessarily end with the robot arriving at the input station M_0 one could also slice through a transport move and we would also need to take these ones into account. Note that for a machine M_k there can be at most one overlapping relation. This is the last operation i processed on machine M_k for which the 'else'-case in (constraint (3.1)) holds. Looking at the graph in Figure 3.6 the overlapping relations are those that are not represented by an arc so far. E.g., there would be an arc representing a precedence constraint from node 4 in this cycle to node $\star^2$ in the next cycle of length 6, i.e. an arc from node 4 in this cycle

to node $\star^2$ in the same cycle of length $6 - \alpha$.

We denote by P the set of *pairs* $|j,i|$ with $i,j \in V$ for which there exits an overlapping relation from i to j. (The order of j and i in this definition is chosen because an overlapping relation can be seen as one in which the order in the robotic cycle is the wrong way round.) For each pair $|j,i| \in P$, we introduce two arcs $(0,j)_{|j,i|}$ and $(i,\star)_{|j,i|}$ of variable lengths $x_{|j,i|}$ and $y_{|j,i|}$, respectively, which depend on the precedence constraint (3.1). For every overlapping operation i with $S_i + p_i + t_{suc(i)} = S_{suc(i)} + \alpha$ together with $p_i^{\min} \leq p_i$, we add one arc from 0 to $suc(i)$ of variable length $x_{|suc(i),i|}$ and another arc from i to $\star$ of variable length $y_{|suc(i),i|}$. The property of those lengths is, that

$$x_{|suc(i),i|} + y_{|suc(i),i|} = p_i^{\min} + t_{suc(i)}.$$

To include those relations in the graph, we define the following set of (possible parallel) arcs associated with P by

$$(E \cup A)' = \{(0,j)_{|j,i|}, (i,\star)_{|j,i|} \mid |j,i| \in P\}.$$

Associated with every arc pair $|j,i| \in P$ there exists a value $d_{|j,i|} \geq 0$, which denotes the minimal length of a pair, and corresponds to the minimal distance between node i and node j. By adding the arcs of $(E \cup A)'$ to graph G we get an extended graph G' for the problem.

Example 3.5.2. *Consider again the data of Example* 3.5.1. *For the precedence constraints, there is one overlapping operation which is* $i = 4$. *This operation leads to one arc from* 0 *to* $\star^2$ *of variable length* $x_{|\star^2,4|}$ *and one from* 4 *to* $\star$ *of variable length* $y_{|\star^2,4|}$. *For the sum of those lengths we can see that* $x_{|\star^2,4|} + y_{|\star^2,4|} = p_4^{\min} + t_{\star^2} = 6$. *Figure* 3.7 *shows the corresponding graph* G'.

The aim now is to determine the lengths of the arcs in $(E \cup A)'$ such that

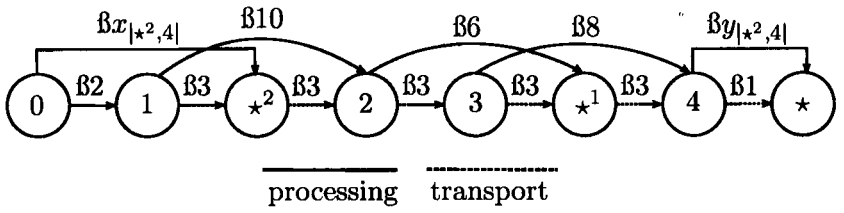


Figure 3.7: Graph G' for Example 3.5.2

1. for all pairs $|j,i| \in P$ it holds that $x_{|j,i|} + y_{|j,i|} = d_{|j,i|}$ and

2. the length of a critical path from 0 to $\star$ in the corresponding graph G' is minimal.

However, this problem can be reduced to an equivalent problem in which the underlying graph $\hat{G}$ has only $O(m)$ vertices, where m is the number of machines. To achieve this, we use the longest path lengths $CP_{i,j}$ in graph $G = (V, E \cup A)$. The graph $\hat{G} = (\hat{V}, \widehat{E \cup A})$ is defined as follows. Let $V_1 = \{j \ : \ |j,i| \in P\}$ and $V_2 = \{i \ : \ |j,i| \in P\}$. Then, $\hat{V} = \{0, \star\} \cup V_1 \cup V_2$. Furthermore, $\widehat{E \cup A}$ is defined by the following arcs:

(a) arcs $(0,i)$ for all $i \in V_2$ with length $CP_{0,i}$,

(b) arcs $(j,\star)$ for all $j \in V_1$ with length $CP_{j,\star}$,

(c) arc $(0,\star)$ with length $CP_{0,\star}$,

(d) arcs (j,i) with lengths $CP_{j,i}$ for all $j \in V_1$ and $i \in V_2$ with the property that there exists a path from j to i,

(e) arcs $(0,j)_{|j,i|}$ with variable length $x_{|j,i|}$ for all $|j,i| \in P$ and

(f) arcs $(i,\star)_{|j,i|}$ with variable length $y_{|j,i|}$ for all $|j,i| \in P$.

The reason for defining $\widehat{E \cup A}$ in this way, is that there are four different possibilities to create a critical path in Graph $\hat{G}$. Such a path can start and finish in arcs of constant length (cf. (c)), it can start in an arc of variable length and finish in an arc

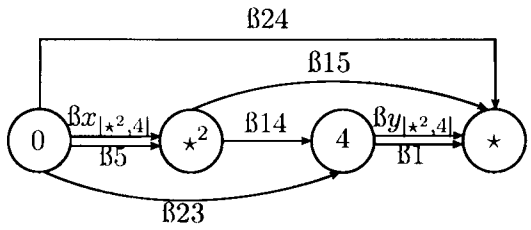


Figure 3.8: Graph $\hat{G}$ for Example 3.5.2

of constant length (cf. (e),(b)), it can start in an arc of constant length and finish in an arc of variable length (cf. (a),(f)), and it can start and finish in arcs of variable length (cf. (e),(d),(f)). Figure 3.8 shows the reduced graph $\hat{G}$ for Example 3.5.2. It is easy to see that the critical path length in G' is equal to the critical path length in $\hat{G}$. Hence, the problem can be reformulated to: Determine the lengths of all arcs $(0,j)_{|j,i|}, (i,\star)_{|j,i|} \in (E \cup A)'$ such that $x_{|j,i|} + y_{|j,i|} = d_{|j,i|}$ and the length of a critical path from 0 to $\star$ is minimal in $\hat{G}$.

3.5.1 A Linear Programming Formulation

The problem derived in the previous section can be formulated as the following linear program $LP1$.

$$\text{minimise } \alpha \tag{3.16}$$

s.t.

$$x_{|j,i|} + y_{|j,i|} = d_{|j,i|} \quad \forall\ |j,i| \in P \tag{3.17}$$

$$x_{|j,i|} + CP_{j,k} + y_{|l,k|} \leq \alpha \quad \forall\ |j,i|, |l,k| \in P \tag{3.18}$$

$$x_{|j,i|} + CP_{j,\star} \leq \alpha \quad \forall\ |j,i| \in P \tag{3.19}$$

$$CP_{0,i} + y_{|j,i|} \leq \alpha \quad \forall\ |j,i| \in P \tag{3.20}$$

$$CP_{0,\star} \leq \alpha \tag{3.21}$$

$$x_{|j,i|}, y_{|j,i|} \geq 0 \quad \forall\ |j,i| \in P. \tag{3.22}$$

To derive optimality conditions for $LP1$, we consider a non-directed bipartite graph B with vertex sets V_1 and V_2 and arc set $A_{CP} \cup A_P$ where

$$\begin{aligned} A_{CP} = \{&(j,i),(i,j) \ : \ j \in V_1, i \in V_2 \text{ and in } \hat{G} \text{ there} \\ &\text{exists a critical path containing both } j \text{ and } i\}, \\ A_P = \{&(j,i),(i,j) \ : \ |j,i| \in P\}. \end{aligned}$$

Note that A_{CP} depends on the given solution implied by the vectors $\left(x_{|j,i|}\right)_{|j,i| \in P}$, $\left(y_{|j,i|}\right)_{|j,i| \in P}$. A path in B is called an *alternating path* if its arcs are alternating between arcs from A_{CP} and A_P. Furthermore, an *alternating cycle* is an alternating path that starts and ends in the same node.

Theorem 3.5.1. *A feasible solution of the linear program $LP1$ is optimal if one of the following conditions holds:*

1. *The corresponding non-directed bipartite graph B contains an alternating cycle.*
2. *There exists an alternating path in the corresponding non-directed bipartite graph B that*
   - *starts with an arc $(c_1, i_1) \in A_{CP}$, where $(i_1, \star)_{|j_1,i_1|}$ is on a critical path with an arc $(0, c_1)$ of constant length, and*
   - *ends with an arc $(j_k, c_2) \in A_{CP}$, where $(0, j_k)_{|j_k,i_k|}$ is on a critical path with an arc $(c_2, \star)$ of constant length.*

Proof. Case 1: Assume that there exists an alternating cycle

$$\underbrace{(i_1,j_1)}_{\in A_P}(j_1,i_2)\underbrace{(i_2,j_2)}_{\in A_P}(j_2,i_3)\cdots(j_{k-1},i_k)\underbrace{(i_k,j_k)}_{\in A_P}(j_k,i_1).$$

To improve the solution, which means decreasing the critical path length, we have to

decrease at least one $x_{|j_\nu,i_\nu|}$-value or $y_{|j_\nu,i_\nu|}$-value. W.l.o.g. we consider the length of $y_{|j_1,i_1|}$. This value is replaced by $y_{|j_1,i_1|} - \varepsilon_1$. Due to condition (3.17), $x_{|j_1,i_1|}$ must be replaced by $x_{|j_1,i_1|} + \varepsilon_1$. Since $(0,j_1)_{|j_1,i_1|}$ is on a critical path with $(i_2,\star)_{|j_2,i_2|}$ the value $y_{|j_2,i_2|}$ must be replaced by $y_{|j_2,i_2|} - \varepsilon_1 - \varepsilon_2$, because otherwise there is no decrease of the solution value. Again this implies that $x_{|j_2,i_2|}$ must be replaced by $x_{|j_2,i_2|} + \varepsilon_1 + \varepsilon_2$, etc. (cf. Figure 3.9(a)). Finally, $x_{|j_k,i_k|}$ must be replaced by $x_{|j_k,i_k|} + \varepsilon_1 + \varepsilon_2 + \ldots + \varepsilon_k$ which implies that the path $(0,j_k)_{|j_k,i_k|}(j_k,i_1)(i_1,\star)_{|j_1,i_1|}$ is of length

$$x_{|j_k,i_k|} + \varepsilon_1 + \varepsilon_2 + \ldots + \varepsilon_k + CP_{j_k i_1} + y_{|j_1,i_1|} - \varepsilon_1 > x_{|j_k,i_k|} + CP_{j_k i_1} + y_{|j_1,i_1|}.$$

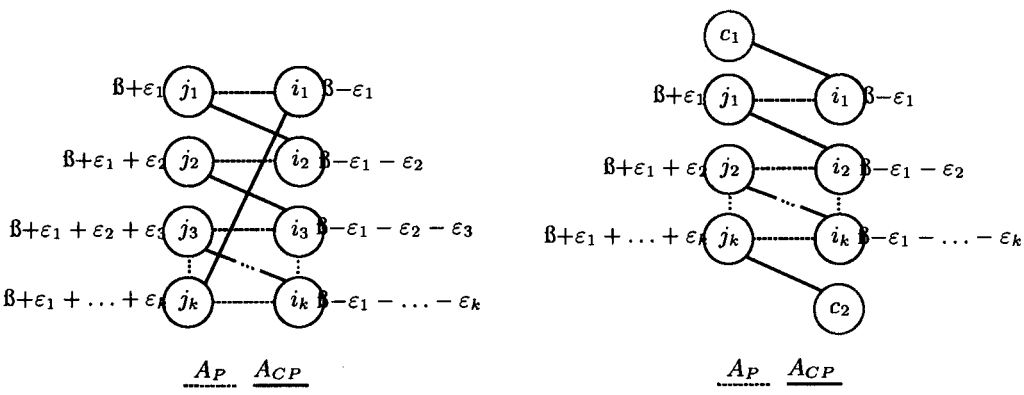
In other words, the critical path length cannot be decreased by decreasing $y_{|j_1,i_1|}$. Symmetrically, the critical path length cannot be decreased by decreasing $x_{|j_1,i_1|}$.

Case 2: In this case, assume that there exists an alternating path which starts in a node c_1. This node implies a critical arc $(0,c_1)$ of constant length. Furthermore, there exists a critical path going through $(0,c_1)$ and $(i_1,\star)_{|j_1,i_1|}$. Symmetrically, there is a node c_2 at the end of the alternating path and there exists a critical path going through $(0,j_k)_{|j_1,i_1|}$ and $(c_2,\star)$ (cf. Figure 3.9(b)). The alternating path then is

$$(c_1,i_1)\underbrace{(i_1,j_1)}_{\in A_P}(j_1,i_2)\underbrace{(i_2,j_2)}_{\in A_P}(j_2,i_3)\ldots(j_{k-1},i_k)\underbrace{(i_k,j_k)}_{\in A_P}(j_k,c_2).$$

In a better solution, the $y_{|j_1,i_1|}$-value must be replaced by $y_{|j_1,i_1|} - \varepsilon_1$. Following the same principle as in case 1 the critical path through $(0,j_k)$ and $(c_2,\star)$ has increased by $\varepsilon_1 + \ldots + \varepsilon_k$ and, therefore, the solution was already optimal.

□



(a) Alternating cycle for item 1

(b) Alternating path for item 2

Figure 3.9: Bipartite graph B for proof of Theorem 3.5.1

3.5.2 An Algorithm to Solve the Special Linear Program $LP1$

In this section we present an algorithm to solve $LP1$. We choose an arbitrary numbering of the elements in $P = \{|j_1, i_1|, |j_2, i_2|, \ldots, |j_{|P|}, i_{|P|}|\}$ and define

$$P_\nu = \{|j_1, i_1|, |j_2, i_2|, \ldots, |j_\nu, i_\nu|\}$$

with $\nu = 1, \ldots, |P|$. The idea of the algorithm is to iteratively solve the relaxation of $LP1$ in which P is replaced by P_ν. That means, we start with graph G, add $P_1 = \{|j_1, i_1|\}$ to it and determine the optimal values for $x_{|j_1,i_1|}$ and $y_{|j_1,i_1|}$, such that the new critical path length is minimal. To this new graph, we add the next pair $|j_2, i_2|$ and solve the problem again. We continue until all arc pairs have been added to the graph. The underlying graph of each relaxation is denoted by $\hat{G}_\nu$. To solve the problem associated with $\hat{G}_{\nu+1}$ $(\nu = 0, \ldots, |P| - 1)$, we use the optimal solution for the problem associated with $\hat{G}_\nu$, where $\hat{G}_0$ is the graph consisting of the single arc $(0, \star)$ of length $CP_{0,\star}$. Note that if $\nu = 0$ then $\alpha = CP_{0,\star}$ is the optimal solution because $P = \emptyset$.

Determining the Arc Lengths of a Pair

We start with describing, how a relaxation associated with $\hat{G}_{\nu+1}$ can be solved. The pseudo-code of such an iteration can be found in Procedure 3.5.1. In this procedure, we use the following notations:

- $\widehat{CP}_{j,i}$ denotes the length of the longest paths from j to i in $\hat{G}_\nu$ for the current value of ν,
- $(\widehat{E \cup A})_\nu$ are the arcs corresponding to P_ν, and
- the lengths of the current solution values are denoted by $x_{|j,i|}, y_{|j,i|}$.

For every pair $|j,i|$, the algorithm works as follows:

Step 1. line 2-3: We start with initialising the lengths of the arc pair to a minimal value. If the length $\widehat{CP}_{0,j}$ of the longest path from 0 to j plus the length $\widehat{CP}_{i,\star}$ of the longest path from i to $\star$ is greater than or equal to $d_{|j,i|}$, then we set

$$x_{|j,i|} = \min\{d_{|j,i|}, \widehat{CP}_{0,j}\} \text{ and } y_{|j,i|} = \min\{d_{|j,i|} - x_{|j,i|}, \widehat{CP}_{i,\star}\}.$$

This choice is neither changing the length of the longest path from 0 to any other node in the network, nor from any node in the network to $\star$, since it is at most equal to the length of a path that is already in the graph.

Step 2. line 4-10: Else we need to increase the length of the arc pair. Therefore, let

$$\bar{d}_{|j,i|} = d_{|j,i|} - (x_{|j,i|} + y_{|j,i|})$$

be the value the arcs are still to short after the first step. We calculate the maximal length an arc from i to $\star$ can be set without increasing the critical

path length. That is $\widehat{CP}_{0,\star} - \widehat{CP}_{0,i}$. Thus $y_{|j,i|}$ can be increased by $\widehat{CP}_{0,\star} - \widehat{CP}_{0,i} - y_{|j,i|}$ without increasing $\widehat{CP}_{0,\star}$. We set $y_{|j,i|}$ to the minimum of this value and $\bar{d}_{|j,i|}$. If $x_{|j,i|} + y_{|j,i|} = d_{|j,i|}$ holds then we are finished.

Step 3. line 11-16: Else, we still need to increase the length of $x_{|j,i|} + y_{|j,i|}$. Again, let $\bar{d}_{|j,i|} = d_{|j,i|} - (x_{|j,i|} + y_{|j,i|})$ be the value the arcs are still to short after the first two steps. We calculate the maximal length an arc from 0 to j can be set without increasing the critical path length. That is $\widehat{CP}_{0,\star} - \widehat{CP}_{j,\star}$. Thus $x_{|j,i|}$ can be increased by $\widehat{CP}_{0,\star} - x_{|j,i|} - \widehat{CP}_{j,\star}$ without increasing $\widehat{CP}_{0,\star}$. We set $x_{|j,i|}$ to the minimum of this value and $\bar{d}_{|j,i|}$. If $x_{|j,i|} + y_{|j,i|} = d_{|j,i|}$ holds then we are finished.

Step 4. line 17-22: Else, we must increase the critical path lengths. Therefore we split the remaining difference $\bar{d}_{|j,i|}$ evenly on both arcs. That means, we increase $x_{|j,i|}$ and $y_{|j,i|}$ by $\frac{\bar{d}_{|j,i|}}{2}$. Note that this could lead to a non-integer value for $x_{|j,i|}, y_{|j,i|}$ and $\widehat{CP}_{0,\star}$. Since the critical path length has been increased we have to make sure, that the solution is still optimal. The following section describes how this can be done.

Correcting the Solution and Proof of Optimality

Assume that we start with a graph representing an optimal solution for the arc pairs inserted so far. That means, all lengths $x_{|j,i|}$, $y_{|j,i|}$ of the arc pairs in the graph are feasible ($x_{|j,i|} + y_{|j,i|} = d_{|j,i|}$) and the critical path length $\widehat{CP}_{0,\star}$ is minimal. If, after inserting an arc pair with Procedure 3.5.1, the critical path length has not changed then the solution is still optimal, since adding an arc pair cannot decrease the critical path length. On the other hand, if the length of the critical path has changed, we need to check, whether the solution is still optimal, and correct it if it is necessary. Finishing in Step 4 of the algorithm, it holds that every critical path contains either arc $(0,j)_{|j,i|}$

Procedure 3.5.1 Determine lengths of arc pair $(0,j),(i,\star) \in (E \cup A)'$

1: **procedure** `determineLengths`(pair $|j,i|$)
2: $x_{|j,i|} := \min\{d_{|j,i|}, \widehat{CP}_{0,j}\}$; $y_{|j,i|} := \min\{d_{|j,i|} - x_{|j,i|}, \widehat{CP}_{i,\star}\}$;
3: **if** $x_{|j,i|} + y_{|j,i|} \geq d_{|j,i|}$ **then return**
4: **else**
5: $\bar{d}_{|j,i|} = d_{|j,i|} - (x_{|j,i|} + y_{|j,i|})$
6: $\Delta_i := \widehat{CP}_{0,\star} - \widehat{CP}_{0,i} - y_{|j,i|}$
7: $y_{|j,i|} := y_{|j,i|} + \min\{\bar{d}_{|j,i|}, \Delta_i\}$
8: $\bar{d}_{|j,i|} := \bar{d}_{|j,i|} - \min\{\bar{d}_{|j,i|}, \Delta_i\}$
9: **end if**
10: **if** $\bar{d}_{|j,i|} = 0$ **then return**
11: **else** $\{a$ *critical path contains* $i\}$
12: $\Delta_j := \widehat{CP}_{0,\star} - x_{|j,i|} - \widehat{CP}_{j,\star}$
13: $x_{|j,i|} := x_{|j,i|} + \min\{\bar{d}_{|j,i|}, \Delta_j\}$
14: $\bar{d}_{|j,i|} := \bar{d}_{|j,i|} - \min\{\bar{d}_{|j,i|}, \Delta_j\}$
15: **end if**
16: **if** $\bar{d}_{|j,i|} = 0$ **then return**
17: **else** $\{a$ *critical path contains* $j\}$
18: $x_{|j,i|} := x_{|j,i|} + \frac{\bar{d}_{|j,i|}}{2}$
19: $y_{|j,i|} := y_{|j,i|} + \frac{\bar{d}_{|j,i|}}{2}$
20: **end if**
21: *optimality check and possible correction (cf. next section)*
22: **return**
23: **end procedure**

or $(i,\star)_{|j,i|}$ or both. In the latter case, due to Theorem 3.5.1, the solution is already optimal, because the cycle $(i,j)(j,i)$ is alternating. Otherwise, we distinguish between two different cases for the bipartite graph:

Case A: There is an alternating path starting in a node u and finishing in a node i or j. The arc corresponding to u has the form $(u,\star)$ or $(0,u)$, i.e. it has constant length, whereas the finishing arc has the form $(0,j)_{|j,i|}$ or $(i,\star)_{|j,i|}$, i.e. it has variable length.

Case B: There is an alternating path including node j and i that starts in a node u and ends in a node t where $(0,j)_{|j,i|}$, $(i,\star)_{|j,i|}$, $(u,\star)_{|v,u|}$ and $(0,t)_{|t,s|}$ are arcs

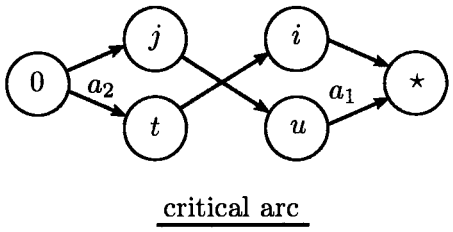


Figure 3.10: Graph for case A or B

in $\hat{G}$ of variable length.

The fact that one case of the two always holds can be seen as follows. Since $(0,j)_{|j,i|}$ and $(i,\star)_{|j,i|}$ are on a critical paths there exists an alternating path of the form

$$\underbrace{(u,j)}_{\in A_{CP}}\underbrace{(j,i)}_{\in A_P}\underbrace{(i,t)}_{\in A_{CP}},$$

where an arc a_1 from u to $\star$ is on a critical path with $(0,j)_{|j,i|}$ and an arc a_2 from 0 to t is on a critical path with $(i,\star)_{|j,i|}$ (cf. Figure 3.10). If a_1 or a_2 are of constant length then case A holds, otherwise case B holds.

The aim is now to decrease the length of the critical paths. Since there can be more than just one critical path in the graph, we obviously have to decrease the lengths of all of them. The rough idea is to follow the alternating path(s) and iteratively decrease every critical path by a (so far unknown) value of $\varepsilon > 0$, until we reach the end of every alternating path. Then we calculate the minimal value for ε, such that at least one additional arc becomes critical and update the graph. This will also add another arc to an alternating path in the corresponding bipartite graph B. We continue with this ε-correction until we either get an alternating cycle or an alternating path that starts and ends in a node belonging to a critical arc of constant length. This means the solution is optimal.

The difference between the two cases is, that in case A we start at node u and only follow the alternating path in one direction, whereas in case B we start at node j and i

and follow the alternating path in both directions. In case A, this method will always lead to an optimal solution since the correction could either lead to an alternating cycle or the alternating path also ends in a constant node. However, in case B we can also finish with an alternating cycle, or one side of an alternating path finishes in a constant node. In the latter case, we change to the correction method to the one presented in case A.

In the following we explain the two correction methods in more detail.

Case A: In this case, there must be a critical path including a constant arc and an arc belonging to a pair. As indicated before, we can assume that this critical path if of the special form $0 \to j \to u \to \star$, where $(u, \star)$ is an arc of constant length and $(0, j)_{|j,i|}$ belongs to a pair. Furthermore, let $0 \to t \to i \to \star$ be another critical path where $(0, t)_{|t,s|}$ belongs to an arc pair. Then the alternating path would be

$$\underbrace{(u,j)}_{\in A_{CP}}\underbrace{(j,i)}_{\in A_P}\underbrace{(i,t)}_{\in A_{CP}}\underbrace{(t,s)}_{\in A_P}.$$

This scenario is shown in Figure 3.11(a). (Note that this figure is not showing the bipartite graph but $\hat{G}$.) To decrease the lengths of the critical paths we start with decreasing the length of $x_{|j,i|}$ by ε_1. Due to constraint (3.17), we have to increase $y_{|j,i|}$ by ε_1. Since $(i, \star)_{|j,i|}$ is on a critical path as well, the extension by ε_1 would increase the critical path length. Therefore, we have to decrease the lengths of all arcs that are critical with $(i, \star)_{|j,i|}$ by $2\varepsilon_1$ to decrease the overall length of these paths by ε_1. This situation is shown in Figure 3.11(b). One can see that both critical paths $0 \to j \to u \to \star$ and $0 \to t \to i \to \star$ have been shortened by ε_1. Again, we have to increase the length of all partner arcs (here $(s, \star)_{|t,s|}$) by $2\varepsilon_1$. After reaching a non-critical arc (here $(s, \star)_{|t,s|}$) on this alternating path, we can calculate a value for ε_1, such that another critical path will be added to the graph, and a

previously non-critical arc becomes critical. (We will explain later, how ε_1 can be calculated.) A possibility for an updated graph is shown in Figure 3.11(c). Here, $(0,l)_{|l,k|}$ and $(s,\star)_{|t,s|}$ became critical after the first ε-correction and the alternating path has been extended to

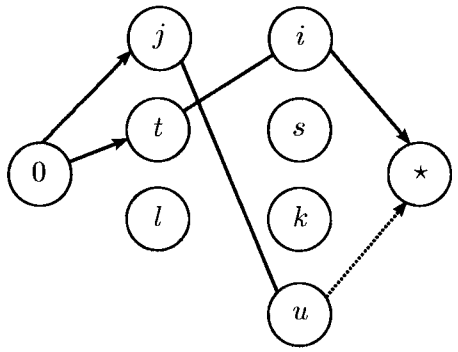
$$\underbrace{(u,j)}_{\in A_{CP}}\underbrace{(j,i)}_{\in A_P}\underbrace{(i,t)}_{\in A_{CP}}\underbrace{(t,s)}_{\in A_P}\underbrace{(s,l)}_{\in A_{CP}}\underbrace{(l,k)}_{\in A_P}.$$

If s would have become critical with j, then there would already be an alternating cycle

$$\underbrace{(j,i)}_{\in A_P}\underbrace{(i,t)}_{\in A_{CP}}\underbrace{(t,s)}_{\in A_P}\underbrace{(s,j)}_{\in A_{CP}}$$

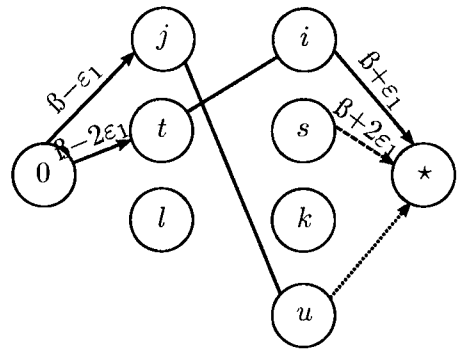
and the solution would be optimal. Otherwise we get the situation shown in Figure 3.11(c). Since the solution is still not optimal (there is no alternating cycle in the corresponding bipartite graph), we start again at the same arc $(u,\star)$ and try another attempt by correcting all critical paths by ε_2 (cf. Figure 3.11(d)). If, after this ε_2-correction, k is critical with j or t an alternating cycle has been created. Otherwise, we have to continue. An optimal solution after a correction with ε_2 is shown in Figure 3.11(e). Note that, after every ε-correction, an additional arc in the graph becomes critical, while the previously critical arcs stay critical. Since the number of non-critical arcs is limited, the approach must finally stop at some point. As soon as a non-critical arc becomes critical with a constant arc or an arc already included in the alternating path, the solution is optimal. In the worst case, every arc in the graph becomes critical.

Case B: This time, we have a critical path including $(0,j)_{|j,i|}$ and another one including $(i,\star)_{|j,i|}$. As indicated before, we can assume that this critical path is of the



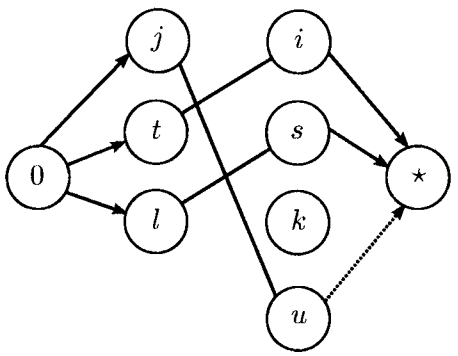
ßcritical arc ßconstant and ßcritical arc

(a) Initial situation

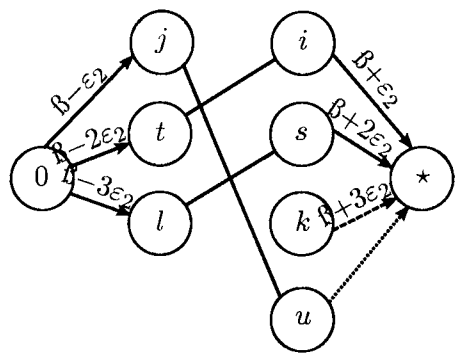


ßmight become ßcritical arc

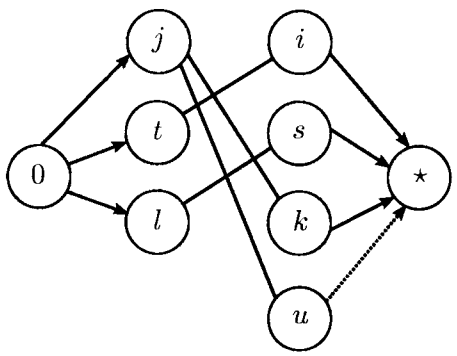
(b) Correction by ε_1



(c) $(0,l)_{|l,k|}$ and $(s,\star)_{|t,s|}$ became critical



(d) Correction by ε_2



(e) $(k,\star)_{|l,k|}$ became critical with $(0,j)_{|j,i|}$. Solution is optimal

Figure 3.11: Possible ε–corrections in case A after adding arc pair $(0,j)_{|j,i|}(i,\star)_{|j,i|}$

special form

$$\underbrace{(s,t)}_{\in A_P}\ \underbrace{(t,i)}_{\in A_{CP}}\ \underbrace{(i,j)}_{\in A_P}\ \underbrace{(j,u)}_{\in A_{CP}}\ \underbrace{(u,v)}_{\in A_P}$$

(cf. Figure 3.12(a)). We start in the middle of the path at $(0,j)_{|j,i|}$ and $(i,\star)_{|j,i|}$ and move "right" from node j and "left" from node i along the path. We start decreasing the length of the arcs that are critical with $(0,j)_{|j,i|}$ and $(i,\star)_{|j,i|}$. Figure 3.12(b) gives an example for such a situation. Both values $x_{|t,s|}$ and $y_{|v,u|}$ are decreased by ε_1 and their partner arcs must be increased by the same value. We choose ε_1 to be large enough so that at least another arc becomes critical (here $(s,\star)_{|t,s|}$ together with $(0,l)_{|l,k|}$) and the alternating path has been extended to

$$\underbrace{(k,l)}_{\in A_P}\ \underbrace{(l,s)}_{\in A_{CP}}\ \underbrace{(s,t)}_{\in A_P}\ \underbrace{(t,i)}_{\in A_{CP}}\ \underbrace{(i,j)}_{\in A_P}\ \underbrace{(j,u)}_{\in A_{CP}}\ \underbrace{(u,v)}_{\in A_P}.$$

Since the solution is still not optimal we carry on with the correction by ε_2 (cf. Figure 3.12(c)). Continuing with the procedure, there are two possible situations that can be reached.

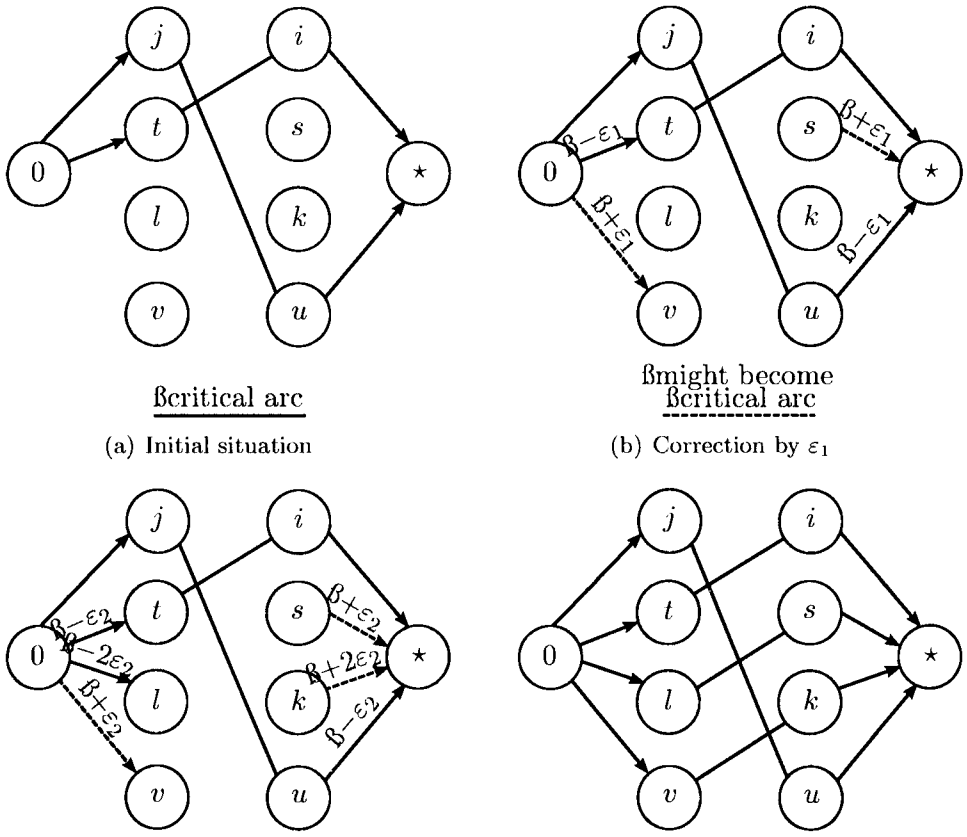
(a) *An arc of constant length becomes critical.* In this case, we cannot extend the alternating path in this direction. This means, we have a situation like in case A. Hence, we do not continue with this correction method but change to the method in case A starting with the constant arc the alternating path now finishes in.

(b) *An arc becomes critical with an arc that is already part of the alternating path.* In this case, we have an alternating cycle and the solution is optimal.

One of these situations must be reached after a finite number of extensions because new arcs become critical as long as none of those two cases occur. Continuing with the example and applying the correction by ε_2 at least one ad-

ditional arc (here $(0,v)_{|v,u|}$ and $(k,\star)_{|l,k|}$) becomes critical (cf. Figure 3.12(d)). This solution is optimal since there is an alternating cycle

$$\underbrace{(v,k)}_{\in A_{CP}}\underbrace{(k,l)}_{\in A_P}\underbrace{(l,s)}_{\in A_{CP}}\underbrace{(s,t)}_{\in A_P}\underbrace{(t,i)}_{\in A_{CP}}\underbrace{(i,j)}_{\in A_P}\underbrace{(j,u)}_{\in A_{CP}}\underbrace{(u,v)}_{\in A_P}.$$



(a) Initial situation

(b) Correction by ε_1

(c) $(0,l)_{|l,k|}$ and $(s,\star)_{|t,s|}$ became critical

(d) $(0,v)_{|v,u|}$ became critical with $(k,\star)_{|l,k|}$. Solution is optimal

Figure 3.12: Possible $\varepsilon-$corrections in case B after adding arc pair $(0,j)_{|j,i|}(i,\star)_{|j,i|}$

The pseudo-codes including a formal description of the corrections are given in Procedure A.1 and A.2 in Appendix A.

As mentioned before, there are different possibilities of generating new critical arcs or alternating paths after an ε-correction. During this correction some variable arc lengths have been increased or decreased by a multiple of ε. However, we do not know this value yet. It can be calculated as follows.

After the correction, the critical path is going to be decreased by ε, i.e.

$$\widehat{CP}_{0,\star}^{new} = \widehat{CP}_{0,\star}^{old} - \varepsilon. \tag{3.23}$$

Changing the arc lengths on the left and right hand side of the graph, changes the length of the paths from 0 to $\star$. Those lengths can be described by

$$x_{|t,s|}(\varepsilon) + \widehat{CP}_{t,\star}(\varepsilon) \text{ and } \widehat{CP}_{0,s}(\varepsilon) + y_{|t,s|}(\varepsilon),$$

where $(0,t)_{|t,s|}$ (respectively $(s,\star)_{|t,s|}$) is either critical or has been increased by a multiple of ε. Those lengths need to be at most as large as the new critical path length, which means

$$x_{|t,s|}(\varepsilon) + \widehat{CP}_{t,\star}(\varepsilon) \leq \widehat{CP}_{0,\star}^{old} - \varepsilon, \tag{3.24}$$

$$\widehat{CP}_{0,s}(\varepsilon) + y_{|t,s|}(\varepsilon) \leq \widehat{CP}_{0,\star}^{old} - \varepsilon, \tag{3.25}$$

must hold. Note that the values on the left hand side of these constraints do not necessarily need to be depending on ε. E.g. it is also possible that a new longest path only contains arcs of constant length. We now have to determine the largest ε, such that no constraint is violated and for at least one of the constraints (3.24) or (3.25) the equality holds for every arc that has been increased by a multiple of ε. We are interested in the minimum value of ε fulfilling the constraints which gives the optimal value for the ε-correction. If we consider the situation in Figure 3.12(b) then e.g. the

arcs $(0,v)_{|v,u|}$ and $(0,j)_{|j,i|}$ would lead to the following constraints for constraint (3.24):

$$x_{|v,u|} + \varepsilon_1 + CP_{v,i} + y_{|j,i|} \leq \widehat{CP}^{old} - \varepsilon_1, \qquad x_{|j,i|} + CP_{j,i} + y_{|j,i|} \leq \widehat{CP}^{old} - \varepsilon_1,$$

$$x_{|v,u|} + \varepsilon_1 + CP_{v,s} + y_{|t,s|} + \varepsilon_1 \leq \widehat{CP}^{old} - \varepsilon_1, \qquad x_{|j,i|} + CP_{j,s} + y_{|t,s|} + \varepsilon_1 \leq \widehat{CP}^{old} - \varepsilon_1,$$

$$x_{|v,u|} + \varepsilon_1 + CP_{v,u} + y_{|v,u|} \leq \widehat{CP}^{old} - \varepsilon_1, \qquad x_{|j,i|} + CP_{j,u} + y_{|v,u|} \leq \widehat{CP}^{old} - \varepsilon_1,$$

$$x_{|v,u|} + \varepsilon_1 + CP_{v,\star} \leq \widehat{CP}^{old} - \varepsilon_1, \qquad x_{|j,i|} + CP_{j,\star} \leq \widehat{CP}^{old} - \varepsilon_1.$$

Additionally, we have to add the constraints corresponding to (3.25). All constraints will lead to an upper bound for ε_1. The minimal upper bound also impacts which arcs will become critical. The calculated value is used to update the arc lengths in the graph that depend on ε_1 and the next iteration can start.

From the discussion above, the following theorem can be obtained.

Theorem 3.5.2. *Adding all arc pairs from $(E \cup A)'$ to G with Procedure* 3.5.1 *provides an optimal solution for the linear program $LP1$.*

Complexity Analysis and Numerical Example

Lemma 3.5.1. *The complexity of adding $|P|$ arc pairs $|j,i| \in P$ to the graph such that the graph has minimal critical path length is $O(|E \cup A| \cdot |P| + |P|^4)$.*

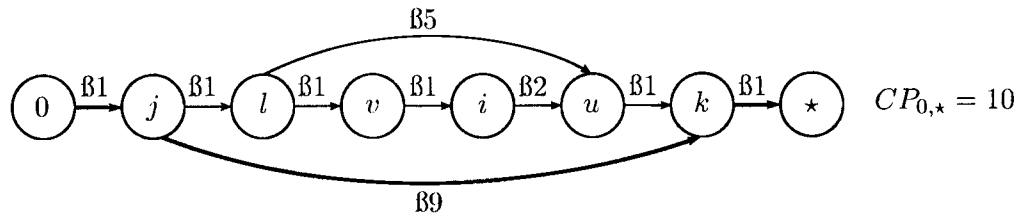
Proof. First, we calculate all longest paths $CP_{j,i}$ with $j,i \in V$ for the given graph G. In a directed acyclic graph with $|E \cup A|$ arcs and a unique topological order every critical path from a specific node to any other node can be calculated in time $O(|E \cup A|)$. Since there are $O(|P|)$ nodes in the reduced graph $\widehat{G}$ and a longest path can be dependent on all $O(|E \cup A|)$ arcs from the original graph G, all critical paths $\widehat{CP}_{j,i}$ with $j,i \in V$ can be calculated in time $O(|E \cup A| \cdot |P|)$. Note that these values will never change throughout the whole procedure, since these paths only contain arcs of constant lengths.

The Steps 1 - 4 of Procedure 3.5.1 without the correction part in line 21 can be done in constant time. If we have to correct the solution, then every critical path either goes

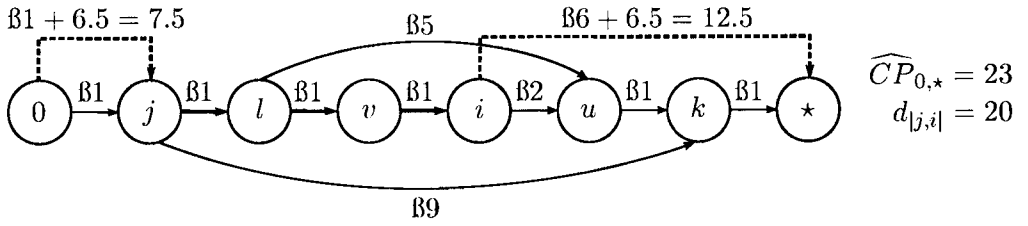
through $(0,j)_{|j,i|}$ or $(i,\star)_{|j,i|}$. In the bipartite graph, we follow every alternating path until we either prove that the solution is optimal or we find some arc pairs that can be corrected by ε and start again. In the worst case, we have to visit all nodes which leads to a complexity of $O(|P|)$. To check optimality, at each step we examine whether the visited arc is on a critical path with an arc already contained in the alternating path. In this case, we have an alternating cycle which proves optimality. This can also be done in time $O(|P|)$. Since at least one arc pair will be corrected and in the worst case we have to correct all arcs, the overall correction has a complexity of $O(|P|^3)$. Afterwards, the value for ε can be calculated in $O(|P|^2)$.
Overall there are $|P|$ arc pairs to be inserted, which means the total complexity including the calculation of all longest paths $CP_{j,i}$ with $j,i \in V$ at the beginning is $O(|E \cup A| \cdot |P| + |P|^4)$. □

Applying this algorithm to the problem used throughout the chapter so far, would not really demonstrate how the algorithm (especially the ε-correction) works. Indeed, the optimal solution of $CP_{0,\star} = 24$ is already given by the length of the critical path in G (cf. Figure 3.6). For the arc pair $|\star^2,4|$ of length $d_{|\star^2,4|} = 6$ the algorithm would already terminate after Step 1 with $x_{|\star^2,4|} = 5$ and $y_{|\star^2,4|} = 1$. Hence, we have chosen a different example, where also the ε-correction has to be applied. For reasons of clarity, we only present the graph G' instead of $\hat{G}$.

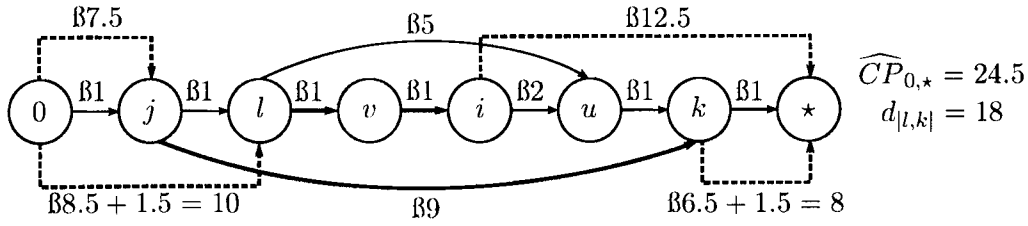
Example 3.5.3. *Consider the initial graph G in Figure* 3.13(a) *with $CP_{0,\star} = 10$. We will add three arc pairs to the graph. The first pair is $(0,j)_{|j,i|}$ and $(i,\star)_{|j,i|}$ with length $d_{|j,i|} = 20$. Starting with the first step of Procedure* 3.5.1, *we set $x_{|j,i|} = CP_{0,j} = 1$ and $y_{|j,i|} = CP_{i,\star} = 4$. The sum of these lengths is* 5, *so that the arcs are $\bar{d}_{ji} = 15$ units to short. Applying the next step, we increase the lengths $y_{|j,i|}$ to* 6 *so that arc $(i,\star)_{|j,i|}$ becomes critical. Since the length is still too short and arc $(0,j)_{|j,i|}$ is already on a critical path, we have to extend the critical path lengths in Step* 4. *The remainder*



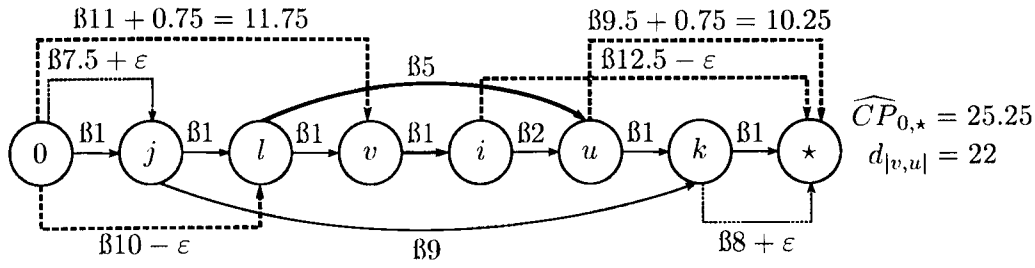
(a) Initial Graph G



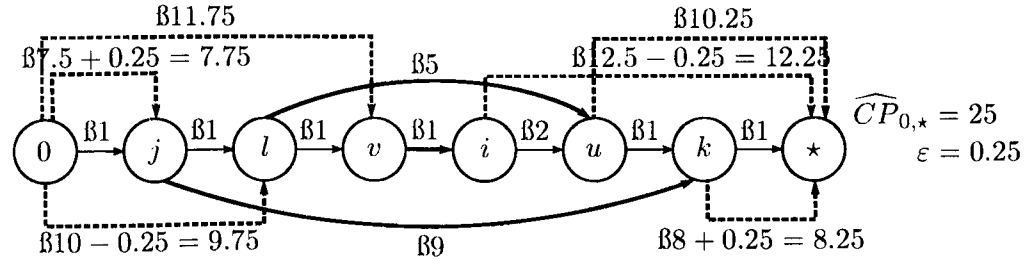
(b) Graph G'_1 after adding the first arc pair $|j, i|$



(c) Graph G'_2 after adding the second arc pair $|l, k|$

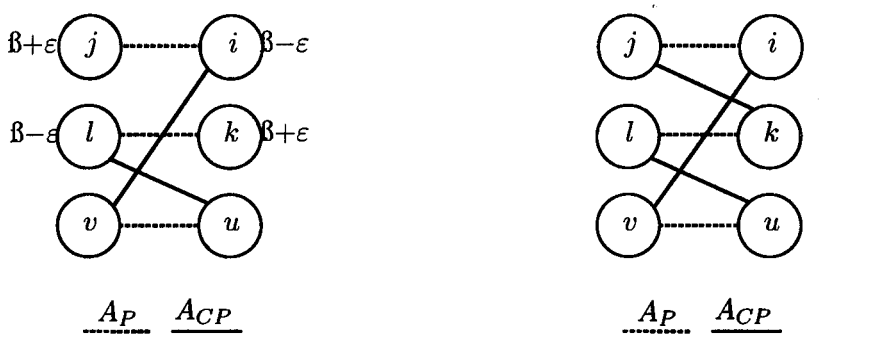


(d) Graph G'_3 after adding the third arc pair $|v, u|$



(e) Graph G'_3 after $\varepsilon-$correction

Figure 3.13: Iterative steps of Example 3.5.3



(a) Bipartite graph before ε-correction

(b) Bipartite graph after ε-correction

Figure 3.14: Bipartite graph of Example 3.5.3

$\bar{d}_{|j,i|} = 13$ *will be evenly split on both arcs, which means* $x_{|j,i|} = 7.5$ *and* $y_{|j,i|} = 12.5$. *The solution is optimal since both arcs are included in the same critical path with lengths* $\widehat{CP}_{0,\star} = 23$ *(cf. Figure* 3.13(b)*). The second arc pair is* $(0,l)_{|l,k|}$ *and* $(k,\star)_{|l,k|}$ *with length* $d_{|l,k|} = 18$. *It is also introduced in Step* 4 *and increases the critical path to* $\widehat{CP}_{0,\star} = 24.5$ *(cf. Figure* 3.13(c)*). This solution is optimal due to Theorem* 3.5.1 *since there are two critical paths* $0 \to j \to k \to \star$ *and* $0 \to l \to i \to \star$, *where each arc of the added pairs is included in one of them, and the bipartite graph contains the alternating cycle*

$$\underbrace{(j,i)}_{\in A_P}\ \underbrace{(i,l)}_{\in A_{CP}}\ \underbrace{(l,k)}_{\in A_P}\ \underbrace{(k,j)}_{\in A_{CP}}.$$

The last pair to add is $(0,v)_{|v,u|}$ *and* $(u,\star)_{|v,u|}$ *with length* $d_{|v,u|} = 22$ *and the algorithm again finishes in step* 4 *now with* $\widehat{CP}_{0,\star} = 25.25$. *However, the solution can be improved, since there is the following alternating path*

$$\underbrace{(j,i)}_{\in A_P}\ \underbrace{(i,v)}_{\in A_{CP}}\ \underbrace{(v,u)}_{\in A_P}\ \underbrace{(u,l)}_{\in A_{CP}}\ \underbrace{(l,k)}_{\in A_P}$$

in the corresponding bipartite graph (cf. Figure 3.14(a)*). Due to case* B *of the correction we extend the path to the left and to the right starting at the pair* $|v,u|$ *and finishing at the arcs* $(k,\star)_{|l,k|}$ *and* $(0,j)_{|j,i|}$ *of variable lengths. Thus, we can increase*

both arcs by ε and decrease their partner arcs by ε. Since both nodes are at the end of the alternating path, we can calculate ε. In particular, we need to find the largest ε such that

$$\begin{aligned}
7.5 + \varepsilon + CP_{j,\star} &\leq \widehat{CP}_{0,\star} - \varepsilon = 25.25 - \varepsilon, \\
10 - \varepsilon + CP_{l,\star} &\leq 25.25 - \varepsilon, \\
11.75 + CP_{v,\star} &\leq 25.25 - \varepsilon, \\
CP_{0,i} + 12.5 - \varepsilon &\leq 25.25 - \varepsilon, \\
CP_{0,k} + 8 + \varepsilon &\leq 25.25 - \varepsilon, \\
CP_{0,u} + 10.25 &\leq 25.25 - \varepsilon.
\end{aligned}$$

Therefore, we first calculate the length of each longest path given in these constraints which obviously can depend on ε. For example $CP_{j,\star} = \max\{15.5 - \varepsilon, 17 + \varepsilon, 16.25\} = 17 + \varepsilon$ since $\varepsilon \geq 0$ and the first constraint would lead to

$$\begin{aligned}
& 24.5 + 2\varepsilon && \leq 25.25 - \varepsilon \\
\Leftrightarrow \quad & 3\varepsilon && \leq 0.75 \\
\Leftrightarrow \quad & \varepsilon && \leq 0.25
\end{aligned}$$

Continuing with the other constraint we finally get $\varepsilon = 0.25$. Correcting all arcs the critical paths length decreases to $\widehat{CP}_{0,\star} = 25$ and the optimal graph can be found in Figure 3.13(e). *It is optimal, because there is a critical path including both arcs $(0,u)_{|v,u|}, (v,\star)_{|v,u|}$ of the same arc pair and the corresponding bipartite graph contains an alternating cycle (cf. Figure* 3.14(b)*).*

So far in this section, we have presented a method to change a cyclic graph representing a feasible solution of the CJSPTB into a directed cyclic graph. Each "backwards" arc

has been removed, which leaves a non-cyclic graph. Afterwards, the removed arcs are successively added to the graph as non-cyclic arc pairs and their lengths are determined in a way such that the overall length of a critical path from the source node 0 to the sink node $\star$ is minimal. We have shown that both graph representations can be used to determine an optimal solution for the CJSPTB with a given robotic cycle.

Lemma 3.5.2. *The complexity to calculate the minimal cycle time of the CJSPTB with a given robotic cycle is* $O(nm + m^4)$.

Proof. Every node $i \in \Omega^\star$ can have at most 3 outgoing arcs: one for the precedence constraint, one for the robotic constraints and one for the machine constraints. The source node 0 can have at most m outgoing arcs. One for every machine except $M(1)$ representing an overlapping operation, and another one for the first transport move τ_1. Hence, there can only be m pairs of arcs in $(E \cup A)'$. Using Lemma 3.5.1, we get an overall complexity of $O(nm + m^4)$. □

It is worth mentioning that the number of machines is usually significantly smaller than the number of operations. Therefore, the complexity is mostly depending on the number of operation. Moreover, for a fixed number of machines the complexity turns out to be linear in n. Furthermore, after determining the final graph, it is easy to calculate the starting times of all operations in the problem and finally building the schedule. The starting point of each operation i is equivalent to the longest path from 0 to i in the non-reduced graph G' including the final lengths of the arcs in $(E \cup A)'$. Since the critical path can be calculated in linear time, the complexity of the whole procedure will not increase when we include the calculation of the complete schedule.

For practical purposes, it is worth sorting the arc pairs before adding them to the graph. Inserting the "long" arc pairs $|j, i| \in P$, that might increase the length $CP_{0,j}, CP_{i,\star}$ or even $CP_{0,\star}$ the most, at the beginning, can already increase some critical path lengths,

such that "shorter" arc pairs can be added to the graph in an earlier stage of the algorithm. Also, possible corrections steps could be avoided. For this reason, we sort the arc pairs in descending order according to the ratio

$$\frac{d_{|j,i|}}{CP_{0,j} + CP_{i,\star}}, \tag{3.26}$$

for all $|j, i| \in P$. This means, the "shorter" arc pairs, that have a smaller possibility to change the critical path, will be inserted at the end of the algorithm. Note that this is not changing the complexity nor guaranteing to speed up the algorithm. However, to get an idea, we compared, the results of the algorithm by sorting the arc pairs once in descending order to the ratio given in (3.26) and the other time in the reverse order and applying it on the data set presented in the next section. The descending sorting slightly outperformed the ascending one on all instances. However, the improvements were only fractions of milliseconds, but used as a method to evaluate the current solutions in a heuristic, it still is an improvement and will therefore save some time.

3.6 Comparison of the Algorithms

In this section, we want to discuss the computational complexity of the three algorithms and present a comparison of their actual running times on various data sets. We start with the latter one. In Dasdan et al. (1998), a similar comparison of several algorithms (Howard's algorithm was one of them) has already been done for graphs in which the height of each arc was at most one

All algorithms have been re-implemented in C++ (single threaded) and built with the Microsoft Visual Studio 2010 compiler. The experiments were executed on a computer equipped with an Intel i5 quad core CPU, 4×2.8MHZ, 8GB of memory, running Microsoft Windows 7 professional, 64bit. To get a fair comparison, we used the same graph data structure for all three algorithms. For the statistical analysis we have used *R*, a software environment for statistical computing and graphics (cf. R Development Core Team).

The data for the comparison has been generated as follows. We have used a data set generator (cf. Section 4.4) to randomly generate 28 problem instances for the CJSPTB of different sizes. For each of these instances we have computed up to 500 feasible robotic cycles (without any restriction to the height) and randomly chose 5 out of them. So, in total we had 140 solutions which can be found in Tables 3.3-3.4. The columns from left to right are indicating the name of the problem instance (name), the number of jobs (N), the number of machines (m), the number of operations (n), a unique identification number (ID), the actual problem height ($h^*_{\star,0}$) and the number of overlapping operations (o). The instances are sorted according to their number of operations, their height and the number of the overlapping operations.

It usually is difficult, to compare the running times of different algorithms for specific problem instances, since they depend a lot on the underlying data structures and other

Name	N	m	n	ID	$h^*_{\star 0}$	o
jspt-2x5-1	2	5	9	1	2	1
				2	2	1
				3	2	1
				4	2	1
				5	2	1
jspt-3x5-1	3	5	13	6	2	1
				7	2	1
				8	2	1
				9	2	1
				10	2	1
jspt-4x5-1	4	5	17	11	2	1
				12	2	1
				13	2	1
				14	2	1
				15	2	1
jspt-2x10-1	2	10	19	16	3	4
				17	3	4
				18	4	5
				19	4	5
				20	4	5
jspt-5x5-1	5	5	21	21	2	1
				22	2	1
				23	2	1
				24	2	1
				25	2	1
jspt-6x5-1	6	5	25	26	2	1
				27	2	1
				28	2	1
				29	2	1
				30	2	1
jspt-3x10-1	3	10	28	31	2	2
				32	3	5
				33	3	5
				34	3	5
				35	3	5
jspt-7x5-1	7	5	29	36	1	0
				37	2	1
				38	2	1
				39	2	2
				40	2	2
jspt-6x6-1	6	6	31	41	2	1
				42	2	2
				43	2	2
				44	2	2
				45	2	2
jspt-8x5-1	8	5	33	46	2	1
				47	2	2
				48	2	2
				49	2	2
				50	2	2
jspt-4x10-1	4	10	37	51	3	3
				52	2	4
				53	3	4
				54	3	4
				55	3	5
jspt-9x5-1	9	5	37	56	2	2
				57	2	2
				58	2	2
				59	2	2
				60	2	2

Name	N	m	n	ID	$h^*_{\star 0}$	o
jspt-10x5-1	10	5	41	61	2	2
				62	2	2
				63	2	2
				64	2	2
				65	2	2
jspt-7x7-1	7	7	43	66	2	2
				67	2	2
				68	2	2
				69	2	2
				70	2	2
jspt-5x10-1	5	10	46	71	3	3
				72	2	3
				73	3	5
				74	3	5
				75	3	5
jspt-6x10-1	6	10	55	76	2	3
				77	3	3
				78	3	3
				79	3	4
				80	3	4
jspt-8x8-1	8	8	57	81	2	2
				82	2	2
				83	2	2
				84	2	2
				85	2	3
jspt-7x10-1	7	10	64	86	2	3
				87	2	3
				88	2	3
				89	2	4
				90	2	4
jspt-5x15-1	5	15	71	91	3	5
				92	3	6
				93	3	6
				94	3	6
				95	3	6
jspt-8x10-1	8	10	73	96	2	1
				97	2	2
				98	2	2
				99	2	2
				100	2	2
jspt-9x9-1	9	9	73	101	2	4
				102	2	4
				103	2	4
				104	2	4
				105	2	4
jspt-9x10-1	9	10	82	106	2	1
				107	2	1
				108	2	1
				109	2	3
				110	2	3
jspt-6x15-1	6	15	85	111	3	5
				112	3	6
				113	3	6
				114	3	6
				115	3	6
jspt-10x10-1	10	10	91	116	1	0
				117	2	2
				118	2	2
				119	2	2
				120	2	2

Table 3.3: Problem instances part 1

Name	N	m	n	ID	$h^*_{\star 0}$	o
jspt-7x15-1	7	15	99	121	2	3
				122	2	3
				123	2	3
				124	2	3
				125	2	3
jspt-8x15-1	8	15	113	126	1	0
				127	1	0
				128	1	0
				129	1	0
				130	2	1
jspt-9x15-1	9	15	127	131	2	1
				132	2	1
				133	2	2
				134	2	2
				135	2	2
jspt-10x15-1	10	15	141	136	2	1
				137	2	1
				138	2	1
				139	2	1
				140	2	1

Table 3.4: Problem instances part 2

implementation details. However, a variation in the input size should make a direct comparison more appropriate. We have solved each of the 140 problem instances 1000 times with every algorithm and took the mean of the running time. The time measured for each run included building the graph, solving the problem and returning the final cycle time. The graph for the running times can be seen in Figure 3.15. For a better overview, we also plotted the same results on a logarithmic scale in Figure 3.16. The first impression is that the PCP and Howard's Algorithm perform similarly for the first third of the instances and, for the last two thirds, Howard's Algorithm is faster. The running time of our new algorithm however, is faster on all 140 instances. The scale of the PCP and Howard's algorithm also has a polynomial shape, whereas our new algorithm tends to be linear (cf. Figure 3.17). We have tried to analyse the data with respect to the number of operations and the average running times using linear regression. The R^2-values[1] for estimated polynomial functions of different degrees and the algorithm's outputs are shown in the following table:

[1] In linear regression, the R^2-value $\in [0,1]$ is the so called coefficient of determination. It measures the discrepancy between the data and an estimation model. The closer this value is to 1.0, the more similar is the data series to the compared function. For more details, we refer to de Sá (2007), Chapter 7.

	degree 1	degree 2	degree 3	degree 4
PCP Algortihm	0.8548	0.9365	0.9367	0.9370
Howard 's Algorithm	0.9279	0.9838	0.9865	0.9920
New Algorithm	0.9947	0.9953	0.9953	0.9955

According to these values, it seems that the PCP and Howard's algorithm solve the problem instances in quadratic time (in relation to the number of operations) whereas our new algorithm needs linear time. Note, that this is only an indicator for this specific data set and cannot be considered as a general result. Also, the data of 26 different problem sizes is rather small.

To show the statistical significance of the computational results in a direct comparison, we have first of all checked whether the results are normally distributed, since parametric tests assume normally distributed data. The Shapiro-Wilk-Test (cf. Shapiro and Wilk (1965)) shows that the data is not normally distributed (p-value < 0.0001) for all three algorithms), which means we need to use a test that does not assume a normal distribution of the underlying data. Note that those non-parametric tests are generally weaker than the parametric ones. In our case, we used the *Wilcoxon signed-rank test* (cf. Wilcoxon (1945)). It is a non-parametric statistical hypothesis test, that checks for two related samples (or in our case measurements) whether their population means differ or not. We have compared pairwise all algorithms against each other. The result from the experimental data sets was that our new algorithm is better than Howard's and this one in turn is better than the PCP algorithm. All test results turned out to be highly significant (p-value $< 10^{-10}$), which means that our algorithm works better on real instances for the CJSPTB. However, we want to point out, that we have tested the algorithms on problem specific data sets and not on arbitrary graphs. On graphs, where our algorithm, for instance, has to perform a correction step after each inserted arc pair, another algorithm might be faster.

Also, the computational complexity of our algorithm is not necessarily better than the

complexity of the other ones (e.g. for $n = m$ the PCP has the same complexity of $O(m^4)$). However, its advantage is based on the separation of solving a relaxation very quickly ($O(nm)$) and correcting the solution if necessary ($O(m^4)$). In most cases, the correction, which is quite expensive in terms of computational complexity, does not need to be executed, which makes the running times appearing to be linear. The other algorithms do not make such a distinction. For example, the two nested for-loops in the PCP (cf. lines 3 - 11) always have to be executed $O(n)$ times.

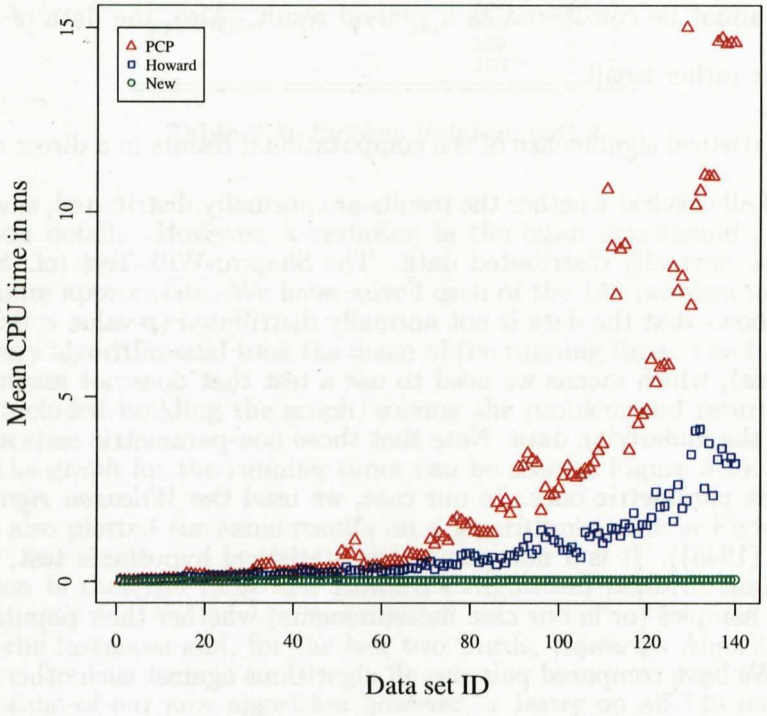


Figure 3.15: Average running times for the three algorithms

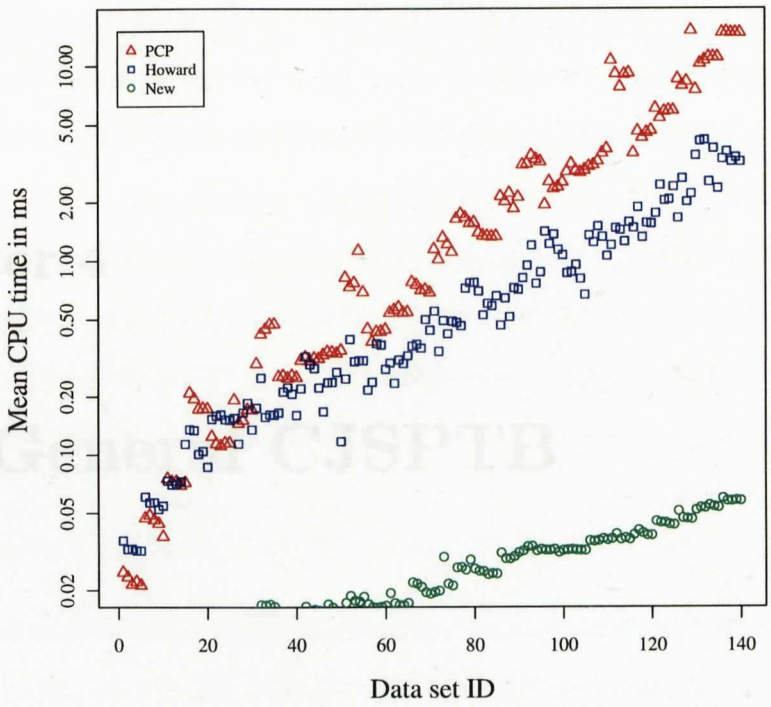


Figure 3.16: Average running times for the three algorithms on a logarithmic scale

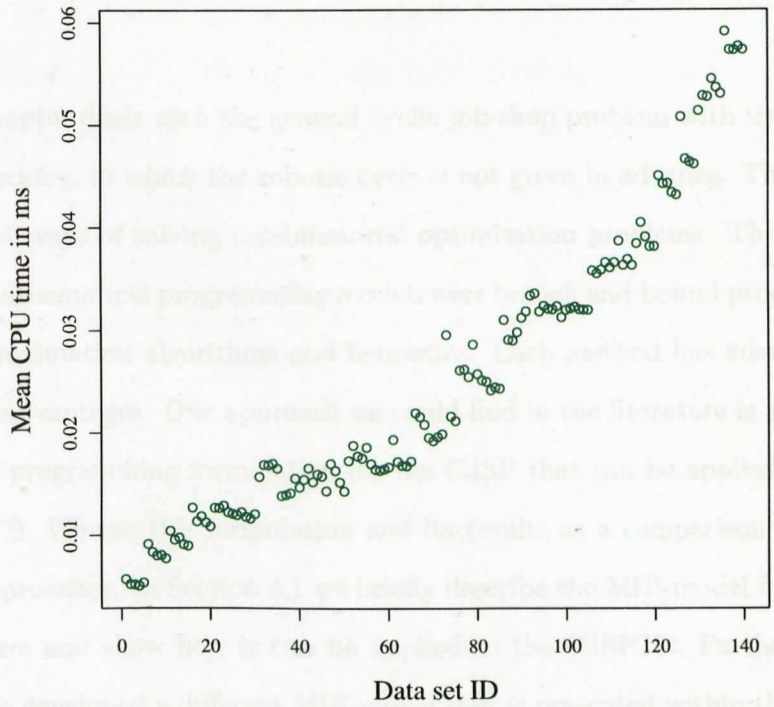


Figure 3.17: Average running times for our new algorithm

Chapter 4

The General CJSPTB

Introduction

This chapter deals with the general cyclic job-shop problem with transport and blocking, in which the robotic cycle is not given in advance. There are different ways of solving combinatorial optimisation problems. They start from mathematical programming models over branch and bound procedures to approximation algorithms and heuristics. Each method has advantages and disadvantages. One approach we could find in the literature is a mixed integer programming formulation for the CJSP that can be applied to the CJSPTB. We use this formulation and its results as a comparison for our new approaches. In Section 4.1 we briefly describe the MIP-model from the literature and show how it can be applied to the CJSPTB. Furthermore, we have developed a different MIP-model that is presented within the same section. The foundation of this model is the analysis of the structure of

cyclic schedules. In particular, we have used the derived connections between the height parameter from Section 3.1 and the number of overlapping operations to define new constraints for linear programming formulation.

In Section 4.2 we adopt the idea of feasible robotic cycles from Section 3.2 to create a tree search algorithm that generates feasible robotic cycles. This search has been incorporated into a branch and bound procedure to solve the overall problem. Since CPLEX often struggles to find solutions for large enough problem instances (which is also the case for our problem), we designed our procedure to be more solution orientated than focused on optimality.

While the first two approaches are exact algorithms, we also developed a tabu search heuristic, which is presented in Section 4.3. We mainly adjusted moves known from the non-cyclic version of the problem such that they can be applied to the CJSPTB.

Since this general problem is not well studied in the literature, there are also no standard benchmarks available. Therefore, we have developed a general instance generator which is briefly described in Section 4.4.

Finally, we generated various data sets and tried to solve them with the four discussed methods. The computational results are summarised in Section 4.5.

4.1 Mathematical Programming Models

Since integer programming software (such as CPLEX (ILOG (2010)) or Gurobi (Gurobi Optimization (2011)) is becoming more and more powerful to solve reasonably large problem instances it is useful to have integer programming formulations for a problem. Within this section, we will present two mixed integer programming formulations for the CJSPTB. One adjusted from the literature and a new one based on the idea of overlapping operations in a cycle.

4.1.1 A Mixed Integer Programming Model from the Literature

The following formulation is based on the work of Hanen (1994) and Brucker and Kampmeyer (2008a). We used their ideas to reformulate our problem definition from Section 2.2.5 with it.

Theorem 4.1.1. *By setting $S_i := S_i(0)$ problem (2.43) - (2.49) (cf. page 57 - 59) can be reformulated to the following mixed integer linear program.*

$$\min \ \alpha \tag{4.1}$$

s.t.

$$S_1 = t_1 \tag{4.2}$$

$$S_i + p_i + t_{suc(i)} = S_{suc(i)} \qquad i \in \Omega \tag{4.3}$$

$$p_i^{\min} \leq p_i \qquad i \in \Omega \tag{4.4}$$

$$S_i + e_{i,pre(j)} + t_j \leq S_j + \alpha HX_{ij} \qquad i,j \in \Omega^\star;\ i \neq j;\ M(i) \neq M(j) \tag{4.5}$$

$$HX_{ij} + HX_{ji} = 1 \qquad i,j \in \Omega^\star;\ i \neq j;\ M(i) \neq M(j) \tag{4.6}$$

$$S_i + p_i + t_{suc(i)} + e_{suc(i),pre(j)} + t_j \leq S_j + \alpha HX_{suc(i)j} \qquad i,j \in \Omega;\ i \neq j;\ M(i) = M(j) \tag{4.7}$$

$$HX_{suc(i)j} + HX_{suc(j)i} = 1 \qquad i,j \in \Omega;\ i \neq j;\ M(i) = M(j) \tag{4.8}$$

$$HX_{ij} \in \mathbb{Z} \qquad i,j \in \Omega^\star;\ i \neq j \tag{4.9}$$

$$e_{i,pre(i)} + t_i \leq \alpha \qquad i \in \Omega^\star \setminus \Omega \tag{4.10}$$

$$p_i + t_{suc(i)} + e_{suc(i),pre(i)} + t_i \leq \alpha \qquad i \in \Omega \tag{4.11}$$

Proof. First of all, we substitute $S_i(r)$ for all $i \in \Omega^\star$ according to constraint (2.50) in (2.44)-(2.49). Therefore, (4.3) is equivalent to (2.44).

Constraints (4.5),(4.6) and (4.7),(4.8) are all of the same structure and we are only going to prove that (4.5),(4.6) and (4.9) are equivalent to constraints (2.48) and (2.50). Applying the substitution described above to constraint (2.48) we get

$$S_i + \alpha r_i + e_{i,pre(j)} + t_j \leq S_j + \alpha r_j$$
$$\text{or } S_j + \alpha r_j + e_{j,pre(i)} + t_i \leq S_i + \alpha r_i,$$

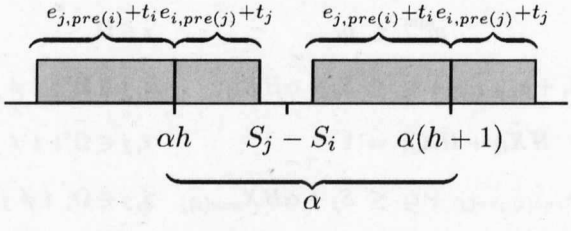
which is equivalent to

$$S_j - S_i \geq \alpha(r_i - r_j) + e_{i,pre(j)} + t_j$$
$$\text{or } S_j - S_i \leq \alpha(r_i - r_j) - e_{j,pre(i)} - t_i,$$

for $i, j \in \Omega^\star$, $i \neq j$, $M(i) \neq M(j)$ and $r_i, r_j \in \mathbb{Z}$. By setting $h := r_i - r_j$ it follows that $S_j - S_i$ cannot be included in any of the intervals

$$]\alpha h - e_{j,pre(i)} - t_i, \alpha h + e_{i,pre(j)} + t_j[$$

for any $h \in \mathbb{Z}$. The following graphic shows this relation.



Hence, there exists a $h' \in \mathbb{Z}$ with

$$S_j - S_i \in [-\alpha h' + e_{i,pre(j)} + t_j, -\alpha(h'-1) - e_{j,pre(i)} - t_i],$$

which means that

$$S_i + e_{i,pre(j)} + t_j \leq S_j + \alpha h'$$
$$\text{and } S_j + e_{j,pre(i)} + t_i \leq S_i + \alpha(1 - h').$$

By setting $HX_{ij} = h'$ and $HX_{ji} = 1 - h'$ this is equivalent to (4.5),(4.6) and (4.9). Analog, one can show that constraints (2.44) and (2.49) are equivalent to (4.3), (4.7),(4.8) and (4.9).

Finally, constraints (2.47) are, after substitution of $S_i(r)$ and $S_i(r+1)$, equivalent to (4.10), and (4.11), (4.3) are equivalent to (2.44) - (2.46). □

In this linear programm, the integer variables HX_{ij} restrict the job sequence on each machine and on the robot. In particular, the following holds.

Lemma 4.1.1. *For a problem given by* (4.1)-(4.11) *the following holds. Let* $i, j \in \Omega^\star$ *be two arbitrary operations with* $p_i^{\min}, p_j^{\min} > 0$. *Then* $S_i(r) \leq S_j(r)$ *for all* $r \in \mathbb{Z}$ *if and only if* $HX_{ij} < HX_{ji}$.

Proof. Because of constraint (2.43) it is sufficient to consider the case $r = 0$. Assume, S_i starts before S_j. Then, $0 \geq S_i - S_j \geq e_{j,pre(i)} + t_i - \alpha HX_{ji}$ holds according to (4.5). This is only true if $HX_{ji} \geq 1$ (since $\alpha \geq e_{j,pre(i)} + t_i$) which implies that $HX_{ij} \leq 0$ and furthermore $HX_{ij} < HX_{ji}$.

To show the other direction let $HX_{ij} < HX_{ji}$. Then it follows that $HX_{ij} \leq 0$ because of (4.6) and even more, $\alpha HX_{ij} \leq 0$. It follows from (4.5) that

$$S_i \leq S_i + e_{i,pre(j)} + t_j \leq S_j + \alpha HX_{ij} \leq S_j$$

and therefore $S_i \leq S_j$. □

The presented mixed integer programming model can be used to solve the CJSPTB for different cyclic models by simply adding another constraint expressing the height restriction. The one we will use within here is the *cyclic job-shop problem* (cf. page 44). To represent the height restriction we have to add the following constraints to the MIP-model:

$$S_{\star j} + e_{\star j,0} \leq S_{\star}, \tag{4.12}$$

for all $j \in \{1, \ldots, N\}$, where $S_\star$ is the start of a dummy end operation. And finally the actual height constraint

$$S_\star \leq S_0 + \alpha h_{\star,0}. \tag{4.13}$$

The result after solving the linear program is the minimal cycle time α and feasible starting times $S_i(0)$ for every operation $\langle i, 0\rangle$ with $i \in \Omega^\star$. Note, that these values do not necessarily have to be included in the interval $[0, \alpha]$ since not all operations of a specific job instance have to start in the same cycle. However, constraints (4.12) and (4.13) ensure that all starting times $S_i(0)$ are included in the interval $[0, \alpha h_{\star,0}]$, with $J(i) = J_j$. To get the starting times of each operation in the first considered cycle $[0, \alpha]$ we calculate the remainder of the division S_i/α for all $i \in \Omega^\star$ and thus, shift every operation in the first cycle. Note that the repetition number of an operation will change due to such a shift.

An obvious question one can ask is whether an operation might clash with another operation on the same machine by shifting it in an earlier cycle and the same question could be asked for the robot moves. Or, in other words: Is the cycle length α large enough to process all operations once? Note that neither (4.10) nor (4.11) are sufficient to bound the minimal cycle length. Recalling the equivalence showed in Theorem

4.1.1 we know that a solution of the MIP-model is also fulfilling constraints (2.43) - (2.49). The constraints guaranteeing that such a clash cannot happen are (2.43), (2.48) and (2.49). (The corresponding constraints in the MIP-model are (4.5)-(4.9)). Each operation starts every α time units and therefore in every cycle at the same position. Constraints (2.49) define the order of any two operations processed on the same machine and ensure that they will not clash and (2.48) does the same for the robot moves.

After solving the linear program and calculating the starting time of each operation in a specific cycle, the repetition number for each starting time in the resulting schedule can be obtained using a variation of Procedure 3.2.1 on page 84. One simply has to substitute the check whether $\tau_{suc(i)}$ precedes τ_i in R by checking whether $S_{suc(i)} \leq S_i$.

By nature formulation (4.1) - (4.11) is not linear, since constraints (4.5) and (4.7) are quadratic. However, it can be rewritten by dividing constraints (4.3) - (4.5), (4.7), (4.10) and (4.11) by α. One can afterwards substitute $1/\alpha = \bar{\alpha}$, $S_i/\alpha = \bar{S}_i$ and $p_i/\alpha = \bar{p}_i$ and change the objective to maximise $\bar{\alpha}$.

4.1.2 A New Mixed Integer Programming Model

In this part, we will present a new formulation for the CJSPT that is more tailored to the problem compared to the model in the previous section. The major difference between cyclic and non-cyclic problems is that the precedence constraints between the operations are slightly relaxed. So, in a specific cycle, operation i does not have to be scheduled before its successor $suc(i)$. In this case, the precedence constraints are, of course, not violated since both operations must belong to different repetitions of $J(i)$, but they provide a more flexible layout of the schedule. This property relies on the existence of overlapping operations. To model overlapping operations, we introduce a

set of binary variables γ_i with $i \in \Omega^\star$ which are defined as

$$\gamma_i = \begin{cases} 1, & \text{if } i \text{ is overlapping;} \\ 0, & \text{else.} \end{cases}$$

The main idea behind this model is to specify the relations between operations in the same cycle, rather than looking ahead into the next one. As before, we make the assumption that we start the cycle at machine M_0 with unloading operation $i = 1$ at time 0. Furthermore, at the end of the cycle, the robot has to drive back to the input machine. Since we are only concentrating on all operations in one specific cycle, we can ignore the repetition numbers. Hence, the following constraints must hold:

$$S_1 = t_1, \tag{4.14}$$

$$S_i + e_{i0} \leq \alpha, \tag{4.15}$$

for all $i \in \Omega^\star$. To ensure that the no-wait and precedence constraints ((2.44) and (2.45)) hold, we have to distinguish between overlapping and non-overlapping operations. For the non-overlapping case, it still holds that $S_i + p_i^{\min} + t_{suc(i)} \leq S_{suc(i)}$. For the overlapping case, we have $S_i + p_i^{\min} + t_{suc(i)} \leq S_{suc(i)} + \alpha$. Both cases can be combined in the following constraint:

$$S_i + p_i^{\min} + t_{suc(i)} \leq S_{suc(i)} + \alpha\gamma_i$$

for all $i \in \Omega$. If i is not overlapping, then $\gamma_i = 0$ and the term $\alpha\gamma_i$ disappears. Since this is not a linear constraint we can split it up into the following two constraints:

$$S_i + p_i^{\min} + t_{suc(i)} \leq S_{suc(i)} + C\gamma_i, \tag{4.16}$$

$$S_i + p_i^{\min} + t_{suc(i)} \leq S_{suc(i)} + \alpha, \tag{4.17}$$

for all $i \in \Omega$ and where $C \in \mathbb{N}$ is a sufficiently large constant.

The constraints for the transportation of each job can be modeled in a very similar way to the ones in the previous model. Therefore, we introduce a set of binary variables θ_{ij} with $i, j \in \Omega^\star$ which are defined as

$$\theta_{ij} = \begin{cases} 1, & \text{if } i \text{ is transported after } j; \\ 0, & \text{else.} \end{cases}$$

The robot constraints (2.48) can then be formulated as

$$S_i + e_{i,pre(j)} + t_j \leq S_j + C\theta_{ij}, \tag{4.18}$$

$$\theta_{ij} + \theta_{ji} = 1, \tag{4.19}$$

for all $i, j \in \Omega^\star$ and where $C \in \mathbb{N}$ again is a sufficiently large constant. Note that these constraints are logically identical to constraints (4.5) and (4.6). Another fact is that a transport move itself will never overlap. This is because we assume that the cycle starts with unloading a job from the input machine and that the cycle finishes with the robot arriving empty at the input machine.

Finally, we will formulate the blocking constraints for operations that have to go on the same machine. We again introduce a set of binary variables β_{ij} with $i, j \in \Omega^\star$ which define the processing order of the jobs on the same machine in the current cycle:

$$\beta_{ij} = \begin{cases} 0, & \text{if } j \text{ is processed after } i \text{ on } M(i) = M(j); \\ 1, & \text{else.} \end{cases}$$

for all $i, j \in \Omega$ with $M(i) = M(j)$.

A first set of constraints is similar to (4.5) and (4.6) and ensures that an operation cannot start its processing before the previous job on the same machine has been

transported to its succeeding one:

$$S_{suc(i)} + e_{suc(i),pre(j)} + t_j \leq S_j + C\beta_{ij}, \tag{4.20}$$

$$\beta_{ij} + \beta_{ji} = 1, \tag{4.21}$$

for all $i, j \in \Omega$ with $M(i) = M(j)$. Considering only one specific cycle, makes it more difficult to deal with overlapping operations. Before we start to tackle this problem, it is worth mentioning that there can be at most one overlapping operation on each machine and this operation always has to be loaded first off and last onto the machine in the cycle. Such an operation i has two processing periods in the cycle. One is at the very beginning, i.e. from time 0 to $S_{suc(i)} - t_{suc(i)}$. The other one is from S_i to α. Even if those two periods physically do not belong to the same repetition of the job, the sum of these processing times needs to be at least $p_i^{\min}$. Therefore, in the following, we have to distinguish between an overlapping and a non-overlapping operation on each machine. For the remaining constraints, consider i, j in Ω, with $i \neq j$ and $M(i) = M(j)$. In the case of i being overlapping, its successor has to start before any other operation on $M(i)$ in the cycle. This leads to the following constraints:

$$S_{suc(i)} + e_{suc(i),pre(j)} + t_j \leq S_j + C(1 - \gamma_i). \tag{4.22}$$

Furthermore, all other operations must have finished their processing and been unloaded before the last operation on a machine can start in the current cycle:

$$S_{suc(j)} \leq S_i + C(1 - \gamma_i), \tag{4.23}$$

where $M(i) \neq M(suc(j))$, $j \neq i$ and $M(i) = M(j)$. Since there is at most one overlapping operation per machine, all other operations on this machine must have

stayed (at least) for their minimal processing times, which leads to:

$$S_j + p_j^{\min} + t_{suc(j)} \leq S_i + C(1 - \gamma_i), \tag{4.24}$$

where $M(i) \neq M(suc(j))$, $j \neq i$ and $M(i) = M(j)$. Finally, constraints (2.46) and (2.47) have to hold as before, which gives the same constraints as in the model from the previous section:

$$\begin{aligned} & e_{i,pre(i)} + t_i \leq \alpha \text{ for } i \in \Omega^\star \setminus \Omega \\ \text{and} \quad & p_i^{\min} + t_{suc(i)} + e_{suc(i),pre(i)} + t_i \leq \alpha \text{ for } i \in \Omega. \end{aligned} \tag{4.25}$$

The model can be summarised by the following mixed integer program.

$$\min \ \alpha \tag{4.26}$$

s.t.

$$S_1 = t_1 \tag{4.27}$$

$$S_i + e_{i0} \leq \alpha \qquad i \in \Omega^\star \tag{4.28}$$

$$S_i + p_i^{\min} + t_{suc(i)} \leq S_{suc(i)} + C\gamma_i \qquad i \in \Omega \tag{4.29}$$

$$S_i + p_i^{\min} + t_{suc(i)} \leq S_{suc(i)} + \alpha \qquad i \in \Omega \tag{4.30}$$

$$S_i + e_{i,pre(j)} + t_j \leq S_j + C\theta_{ij} \qquad i,j \in \Omega^\star \tag{4.31}$$

$$\theta_{ij} + \theta_{ji} = 1 \qquad i,j \in \Omega^\star \tag{4.32}$$

$$S_{suc(i)} + e_{suc(i),pre(j)} + t_j \leq S_j + C\beta_{ij} \qquad i,j \in \Omega, i \neq j, M(i) = M(j) \tag{4.33}$$

$$\beta_{ij} + \beta_{ji} = 1 \qquad i,j \in \Omega, i \neq j, M(i) = M(j) \tag{4.34}$$

$$S_{suc(i)} + e_{suc(i),pre(j)} + t_j \leq S_j + C(1-\gamma_i) \qquad i,j \in \Omega, i \neq j, M(i) = M(j) \tag{4.35}$$

$$S_{suc(j)} \leq S_i + C(1-\gamma_i) \qquad i,j \in \Omega, i \neq j, \ M(i) = M(j) \neq M(suc(j)) \tag{4.36}$$

$$S_j + p_j^{\min} + t_{suc(j)} \leq S_i + C(1-\gamma_i) \qquad i,j \in \Omega, i \neq j, M(i) = M(j) \tag{4.37}$$

$$e_{i,pre(i)} + t_i \leq \alpha \qquad i \in \Omega^\star \setminus \Omega \tag{4.38}$$

$$p_i^{\min} + t_{suc(i)} + e_{suc(i),pre(i)} + t_i \leq \alpha \qquad i \in \Omega, \tag{4.39}$$

$$\gamma_i, \theta_{ij}, \beta_{ij} \in \{0,1\} \tag{4.40}$$

for all $i,j \in \Omega^\star$ and $C \in \mathbb{N}$ is a sufficiently large constant. Since modeling the overlapping operations on each machine is the key property of this model, we will refer to it as the *CJSPTB-MIP-OO*.

The last constraint we have to specify is the height restriction. This, as before, depends on the different models. We will only consider the cyclic job-shop problem model and the one with job repetition.

We start with the cyclic job-shop problem model and its height $h_{\star,0}$. As we have mentioned before, in an arbitrary but fixed feasible cycle, we can set the repetition

number of the first operation of every job to a fixed number. In our case, we chose $h_{\star,0}$ for every job. For every overlapping operation i of a job J_j the repetition number of the successor $suc(i)$ in the current cycle will decrease by one. This means the last operation $\star^j$ of job J_j will have the repetition number $h_{\star,0} - o_j$. According to the definition of $h_{\star,0}$ the $(r + h_{\star,0})$-th repetition of a job can only start if the r-th repetition of any job has finished. Thus, in our case using Theorem 3.1.1 we have to ensure, that for two jobs J_j and J_k, the end of the last operation of J_j (which is $S_{\star^j}$) can only happen before the start of the first operation 0^j of the job, if

$$\begin{aligned} o_j &\leq h_{\star,0}, \quad \text{for } k \neq j; \\ o_j &< h_{\star,0}, \quad \text{else.} \end{aligned}$$

This can be done with the following constraints:

$$S_{\star^j} - (h_{\star,0} - o_j)C > S_{0^k}, \tag{4.41}$$

$$S_{\star^j} - (h_{\star,0} - o_j - 1)C > S_{0^j}, \tag{4.42}$$

for all $j, k = 1, \ldots, N$ with $j \neq k$. The number of overlapping operations is given by

$$\sum_{i \in J_j} \gamma_i = o_j, \tag{4.43}$$

for all $j = 1, \ldots, N$.

For the model with job-chain repetition, we have to remember that the number of overlapping operations of a specific job J_j is equal to $h^*_{J_j} - 1$ (cf. Theorem 3.1.2). Since $h^*_{J_j} \leq h_{J_j}$ must hold, the number of overlapping operations per job J_j has to be less than or equal to $h_{J_j} - 1$, which leads to the additional constraints

$$\sum_{i \in J_j} \gamma_i \leq h_{J_j} - 1, \tag{4.44}$$

for all $j = 1, \ldots, N$.

The repetition numbers for a resulting schedule can be obtained in the same way as in Section 4.1.1.

4.2 A Branch and Bound Procedure

Within this section, we want to present a new branch and bound procedure for solving the CJSPTB. Branch and bound algorithms are general solutions methods to find the optimal solution for a discrete or combinatorial optimisation problem. The idea is based on the work by Land and Doig (1960) and contains two main steps. A *branching* part, in which the algorithm systematically evaluates all possible solutions. "Systematically" in this case means, that first of all no solution is visited more than once, and that the solutions are ordered in specific way, usually using a *tree* structure. The search tree has the property, that a solution of a parent node has a relation to the solution of its child nodes. This relation is used in the second part, the *bounding*. For a minimisation problem, every feasible solution provides an upper bound for the optimal solution. Furthermore, the structure of the search tree is usually set up in a way, that one can also determine a lower bound for all possible child nodes (by using some relaxation method). If the lower bound for the children of a specific node is greater than the best upper bound (solution value) discovered so far, then there is no need to evaluate those nodes, since they cannot provide a better solution and therefore can be "chopped off" the tree. This way of pruning the search tree decreases the number of solutions to evaluate and therefore speeds up the search procedure. The algorithm stops returning the optimal solution, when all solutions have been visited or chopped off the tree.

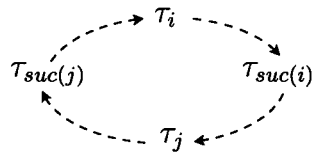
As we have seen before, a solution for the CJSPTB can be represented by a robotic cycle (a permutation of all transport moves). However, we have shown in Section 3.2 that not all possible permutations lead to a feasible robotic cycle. Thus, we will show in the next part an efficient way of constructing feasible robotic cycles.

4.2.1 Constructing Feasible Robotic Cycles

In Section 3.2, Definition 3.2.1, we have defined a blocking-feasible robotic cycle R as one, in which before executing τ_i, job $J(i)$ must be loaded and finished its processing on machine $M(pre(i))$ and the robot is never required to transport a job to an already loaded machine. From this definition, we can state the following:

Lemma 4.2.1. *For a robotic cycle R the following statements are equivalent.*

1. *R is blocking-feasible.*
2. *For every machine M_k $(k = 1, \ldots, m)$ and any operations $i \neq j$ with $M(i) = M(j) = M_k$ and existing succeeding operations, none of the transport moves τ_j or $\tau_{suc(j)}$ occur between τ_i and $\tau_{suc(i)}$ in R.*
3. *For every machine M_k $(k = 1, \ldots, m)$ and any operations $i \neq j$ with $M(i) = M(j) = M_k$ and existing succeeding operations, the order of $\tau_i, \tau_j, \tau_{suc(i)}, \tau_{suc(j)}$ in R has to be compatible with the following graph:*



where $\tau_i \dashrightarrow \tau_j$ means that τ_i occurs before τ_j in R in a cyclic manner.

Proof. $1 \Rightarrow 2$: Assume that R contains the sequence $\tau_i \dashrightarrow \tau_j \dashrightarrow \tau_{suc(i)}$. Since R is blocking-feasible, $J(i)$ must have been unloaded off $M(i)$ before another job $J(j)$ with $M(i) = M(j)$ can be loaded onto $M(i)$. Since the unloading operation $\tau_{suc(i)}$ occurs after τ_j, the robotic cycle cannot be blocking-feasible. On the other hand, if we assume that R contains a sequence $\tau_i \dashrightarrow \tau_{suc(j)} \dashrightarrow \tau_{suc(i)}$, the robot (while executing $\tau_{suc(j)}$) tries to unload job $J(j)$ off the machine. This is not possible since $J(i)$ is currently

loaded on $M(i)$. Thus, again R is not blocking-feasible.

$2 \Rightarrow 1$: Let $\tau_{suc(i)}$ be the next transport move to be performed by the robot. Assume that $J(i)$ is not available on $M(i)$. Hence, $M(i)$ must either be occupied by a different job or empty. The case of $M(i)$ being occupied cannot arise, since τ_i must be executed before $\tau_{suc(i)}$ without any other transport move τ_j $(M(i) = M(j))$ executed in between $(\tau_i \dashrightarrow \tau_{suc(i)} \dashrightarrow \tau_j)$. On the other hand, $M(i)$ cannot be empty, since otherwise either τ_i would not have been executed or $J(i)$ has been unloaded by a different transport operation $\tau_{suc(j)}$ which is also a contradiction to Statement 2.
Finally, the machine that is going to be loaded must be empty. If τ_i is the next transport move to be performed, then $M(i)$ must be empty. This is the case, due to the fact that after performing any other transport move τ_j loading $M(j) = M(i)$, its succeeding transport move $\tau_{suc(j)}$ unloading $M(j)$ will have been executed according to the graph.
$2 \Leftrightarrow 3$: This is easy to see, since 3 is simply a graphic interpretation of 2 and vice versa. □

For a cyclic schedule this means that two transport moves τ_i and τ_j loading the same machine and their successors have to be executed in the following order.

$$\ldots \tau_i(r_i) \dashrightarrow \tau_{suc(i)}(r_i) \dashrightarrow \tau_j(r_j) \dashrightarrow \tau_{suc(j)}(r_j) \dashrightarrow$$
$$\tau_i(r_i + 1) \dashrightarrow \tau_{suc(i)}(r_i + 1) \dashrightarrow \tau_j(r_j + 1) \ldots .$$

The next question is, how can we construct those feasible cycles? With this question in mind, we define a *partial robotic cycle* (*PRC*) R^p that consists of a list R^{done} containing the so far scheduled (i.e. finished) transport moves and a set R^{todo} of unscheduled transport moves. The list R^{done} is a robotic cycle, where not necessarily all transport moves have been scheduled yet, and the set R^{todo} contains all elements, which are not

in R^{done}. During the construction phase, the unscheduled transport moves in R^{todo} will be added to the end of R^{done} until R^{done} is a complete robotic cycle and R^{todo} contains no more elements. One can think of different strategies to construct blocking-feasible solutions. One, that we will discuss here, is based on a depth-first-search tree where the nodes are partial robotic cycles which are blocking-feasible. The method works as follows.

1. Initialisation: The root of the tree is an initial *PRC*, where R^{done} contains only transport move τ_1 and R^{todo} contains all remaining transport moves in an arbitrary order. (Note that since we are considering a cyclic problem, we can always fix the first operation in every cycle to be τ_1).

2. Construction of a blocking-feasible child: Given a blocking-feasible *PRC* (node) of the tree, a blocking-feasible child (if it exists) can be constructed as follows. We test for every unscheduled transport move τ_i in R^{todo} whether it can be placed at the end of R^{done}, so that the *PRC* could still lead to a blocking-feasible robotic cycle (we will see in the next paragraph how this can be done efficiently).

3. For every transport move τ_i that can be placed, we add a new child to the tree wherein τ_i is deleted from R^{todo} and placed at the end of R^{done}.

4. For all children recursively continue with the procedure. If there are no children left the procedure stops.

While shifting transport moves from R^{todo} to R^{done} one can check with Definition 3.2.1 or property 3 of Lemma 4.2.1 whether this *PRC* could potentially lead to a blocking-feasible robotic cycle or not. Therefore, we can also apply the concept of blocking-feasibility to partial robotic cycles.

If the scheduled transport moves in a *PRC* are incompatible with the graph in Item 3 of Lemma 4.2.1 then this *PRC* is not blocking-feasible. However, there is no need to check

the complete PRC all over again after inserting a transport operation. During every transport move τ_i, the robot gets in touch with two machines: $M(pre(i))$ and $M(i)$. After inserting τ_i into R^{done} the blocking-feasibility is *violated* if one of the following cases occurs.

1. R^{done} contains one of the following sequences

$$\tau_j \dashrightarrow \tau_i \text{ and } \tau_{suc(j)} \in R^{todo} \quad \text{or} \quad \tau_{suc(j)} \dashrightarrow \tau_i \text{ and } \tau_{pre(i)} \in R^{todo}, \tag{4.45}$$

   for all j with $M(pre(i)) = M(j)$;

2. R^{done} contains one of the following sequences

$$\tau_j \dashrightarrow \tau_i \text{ and } \tau_{suc(j)} \in R^{todo} \quad \text{or} \quad \tau_{suc(i)} \dashrightarrow \tau_i \text{ and } \tau_j \in R^{todo}, \tag{4.46}$$

   for all j with $M(j) = M(i) \neq M_\star$.

Note that these cases are all violating the graph in Lemma 4.2.1. Therefore, those situations cannot lead to a blocking-feasible robotic cycle.

The first condition in (4.45) needs to be checked only for the last operation j with $\tau_j \in R^{done}$ processed on $M(pre(i))$. If $\tau_{suc(j)}$ is also contained in R^{done} and there is another operation k with τ_k in R^{done} processed on $M(pre(i))$ with $\tau_{suc(k)} \in R^{todo}$ then the corresponding blocking-infeasibility must have been detected when τ_j was added to R^{done}. This comment also applies to the first condition in (4.46). Therefore the conditions in (4.45) and (4.46) can be checked in constant time if

- for each machine the last operation j with $\tau_j \in R^{done}$ is stored, and
- a data structure is used which allows for any operation j to check whether τ_j is in R^{done} in constant time.

Example 4.2.1. *Consider the following job-shop problem with two jobs J_1, J_2 and three machines M_1, M_2, M_3. The following table shows the processing times and the machine allocations.*

Job	J_1					J_2			
Operation	1	2	3	4	5	6	7	8	9
Processing time	3	7	8	4	8	3	12	6	4
Machine	M_1	M_2	M_1	M_2	M_3	M_1	M_2	M_1	M_2

Thus, a robotic cycle consists of 11 *transport moves. As mentioned before, we assume that the first operation in a robotic cycle is always τ_1. In the following, we will simply write i instead of τ_i. Thus, the PRC in the root of the search tree consists of*

$$R^{done} = (1) \text{ and } R^{todo} = \{\star^1, 2, 3, 9, 5, 7, \star^2, 6, 4, 8\}.$$

The only machine status at this point is $M_1 : 1$, $M_2 : -$ and $M_3 : -$, where $-$ means the machine is empty. In the next steps we will place all feasible transport moves in R^{todo} at the second position in R^{done}. This leads to the following PRC's:

$$(1, \star^1), \quad (1, 2), \quad (1, 5), \quad (1, \star^2).$$

Transport moves τ_9, τ_4, τ_7 cannot be scheduled at the second position, since they would try to unload machine M_1 without being the successor of the operation currently loaded on M_1 (cf. (4.45)*). On the other hand, transport moves τ_8, τ_6, τ_3 cannot be scheduled since all of them will load M_1 which is currently blocked by operation* 1 *(cf.* (4.46)*).*

In the next generation, we start with the first of the previous generated PRC where

$$R^{done} = (1, \star^1) \text{ and } R^{todo} = \{2, 3, 9, 5, 7, \star^2, 6, 4, 8\}.$$

The machine status is still $M_1 : 1$, $M_2 : -$ and $M_3 : -$, since $\tau_{\star^1}$ has brought a job to

the output station $M_\star$. The next possible moves are

$$(1, \star^1, \star^2), \qquad (1, \star^1, 5), \qquad (1, \star^1, 2).$$

Since M_1 was still occupied by operation 1 only the same operations as before could be scheduled at the next index. The complete tree search until finding a feasible solution is presented in Figure 4.1. From the PRC with $R^{done} = (1, \star^1, 2, 3, 4, 5)$ onwards there is always only one possible PRC in every next generation, which leads to the following blocking-feasible robotic cycle:

$$R = \tau_1, \tau_{\star^1}, \tau_2, \tau_3, \tau_4, \tau_5, \tau_6, \tau_7, \tau_8, \tau_9, \tau_{\star^2}.$$

The whole problem only has 40 feasible robotic cycles. The presented method needs to create 529 different partial robotic cycles to find these 40 solutions. Note that the number of all possible (not necessarily feasible) robotic cycles is $10! = 3,628,800$. (Note that the transport move τ_1 can be fixed at the first position which means there are only $(n-1)!$ different robotic cycles.)

We want to point out, that this constructive method is generating all possible blocking-feasible robotic cycles. This is easy to see, since in every iteration, the algorithm tries to schedule every transport move left in R^{todo} at the next position in R^{done} and only skips those ones, that cannot lead to a blocking-feasible robotic cycle. For a non-reentrant single job problem, for instance, every permutation of the transport moves is a blocking-feasible robotic cycle. And our algorithms would generate all of them.

Table 4.1 shows the number of blocking-feasible robotic cycles of some small instances and the number of nodes generated during the search needed to determine these feasible cycles. The general format is jspt-Nxm-a where as before N is the number of jobs, m the number of machines, and a an index to enumerate different data sets of the same

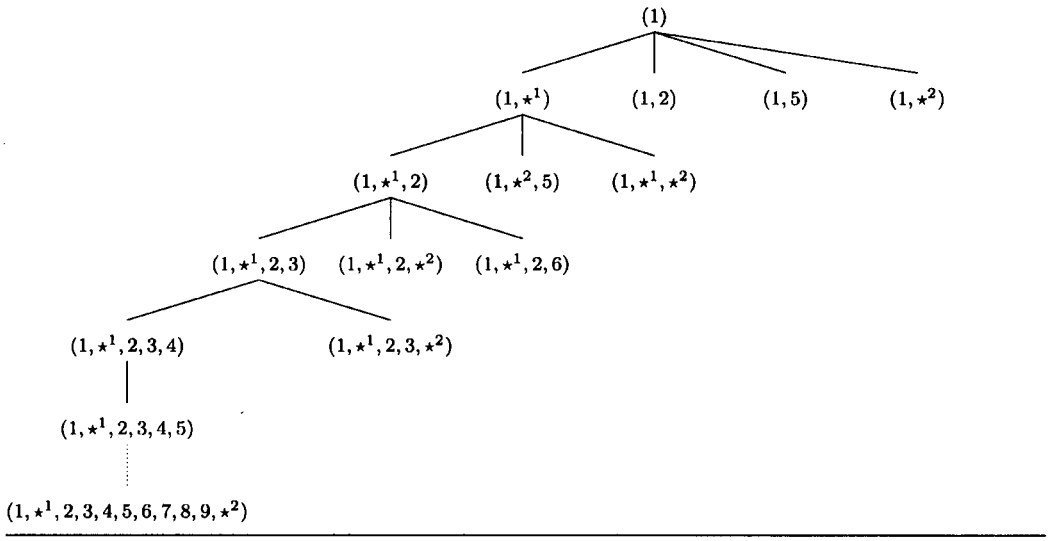


Figure 4.1: Search tree for Example 4.2.1

size. More detailed information about the data-sets are given in Section 4.4. Note that the explored nodes are partial robotic cycles which are generated to build up the feasible solutions.

A feasible robotic cycle must, in addition to the blocking-feasibility, also fulfill the height constraint. We will also consider the cyclic job-shop problem model and the one with job repetition. Thus we have upper bounds $h_{\star,0}$ or h_{J_j} for $j = 1, \ldots, N$. During the construction of the different robotic cycles, we can calculate minimum actual heights for every partial route R^{done}. Since, the height will not decrease by inserting more transport operations to R^{done} the PRC becomes infeasible as soon as an actual height is bigger than the given maximum height. We can either use a variation of Procedure 3.2.1 to determine the height of a PRC or, more efficiently, remember the actual heights for each PRC and simply check if it could have changed after adding the next transport move to R^{done}.

Instance	Transport moves	Feasible solutions	Nodes explored	$(N+n-1)!$
Example 4.2.1	11	40	238	3,628,800
jspt-4x5-1	16	440	4,129	> 1e12
jspt-4x5-2	16	302	2,706	> 1e12
jspt-4x5-3	16	2,300	12,287	> 1e12
jspt-5x5-1	20	7,938	70,912	> 1e17
jspt-5x5-2	20	11,328	68,827	> 1e17
jstp-5x5-3	20	17,166	108,249	> 1e17
jspt-6x5-1	24	200,676	1,542,938	> 1e22
jspt-6x5-2	24	205,894	50,803,949	> 1e22
jstp-6x5-3	24	7,521,903	30,318,854	> 1e22

Table 4.1: Blocking Feasible Routes for Small Instances

4.2.2 A Lower Bound for the Construction Phase

In branch and bound algorithms every node in the search tree can be complemented by a lower bound which usually is a relaxation of a potential feasible solution. Those lower bounds can then be compared to an upper bound (e.g. from already explored feasible solutions) and the branch can be 'chopped off' the search tree if the lower bound is greater or equal than any upper bound. So, the bounding can reduce the search space by pruning the tree. In general, bounding methods are meant to be computationally cheap to keep the search as quick as possible.

For every PRC, we will now calculate a lower bound for the cycle time in two stages. For the already scheduled operations in R^{done} we can determine a lower bound LB^{done} for the time needed to process these operations. For the non-scheduled operations in R^{todo}, we present several different lower bounds, that are dependent on the machines, the robot, and the height and combine them to an overall lower bound LB^{todo}. In the following, we will explain how those bounds can be calculated.

We start with considering the list R^{done} of a partial robotic cycle which contains $n + N - |R^{todo}|$ elements. Starting with S_0, we can successively calculate an earliest starting

point of all operations corresponding to the transport moves in R^{done}. An operation i can only start if its predecessor operation $pre(i)$ has been finished and the robot has transported it to its machine $M(i)$. This is at time

$$S_{pre(i)} + p^{\min}_{pre(i)} + t_i. \tag{4.47}$$

In addition to that, the robot must have

- finished the transport move $\tau_{pre^R(i)}$ of the previous operation according to the robotic cycle (which is equivalent to the time when $pre^R(i)$ starts its processing),
- driven empty to $M(pre(i))$ and
- transported the job to its next machine $M(i)$.

This is at time

$$S_{pre^R(i)} + e_{pre^R(i),pre(i)} + t_i. \tag{4.48}$$

In the case that $pre^R(i) = pre(i)$, the robot has waited at $M(pre(i))$ until the processing is finished, and then transported the job to its next machine. (This is also covered by (4.47) since $e_{pre^R(i),pre(i)} = 0$). Finally, the earliest point in time, at which operation i can start, is the maximum of (4.47) and (4.48), since both corresponding conditions have to be fulfilled. This leads to

$$S_i = \max\left(S_{pre^R(i)} + e_{pre^R(i),pre(i)}, S_{pre(i)} + p^{\min}_{pre(i)}\right) + t_i.$$

It follows that, a minimal time needed to schedule the already planned jobs is equivalent to the earliest starting point of the last scheduled operation, which is

$$LB^{done} = S_{R^{done}[n+N-|R^{todo}|]},$$

where $R^{done}[k]$ $(k = 1, \ldots, n+N)$ is the operation belonging to the transport move planned at position k in R^{done}. Setting $S_0 = 0$ to be the earliest starting time of an already planned operation i can be calculated in an iterative way assuming that the starting times of the previously scheduled operations have already been calculated. A formal description to successively calculate LB^{done} is given in Procedure 4.2.1.

Procedure 4.2.1 Calculates the earliest starting time of an operation in R^{done}

1: **procedure** `CalculateEarliestStartingTime`()
2: $S_0 = 0$
3: $S_i = -\infty$ for $i = 1, \ldots, n+N$
4: **for** $k = 1$ to $n + N - |R^{todo}|$ **do**
5: $i = R^{done}[k]$
6: $pre^R(i) = R^{done}[k-1]$
7: $S_i = \max\left(S_{pre^R(i)} + e_{pre^R(i),pre(i)}, S_{pre(i)} + p^{\min}_{pre(i)}\right) + t_i$
8: **end for**
9: $LB^{done} = S_{R^{done}[n+N-|R^{todo}|]}$
10: **end procedure**

Note that if we add another transport move to R^{done} there is no need to calculate the complete bound again. Instead, one can simply update it, by taking the old starting time of the previous bound and carrying out another iteration in the loop, for the last inserted move.

Let us exemplify this on a PRC with

$$R^{done} = (1, \star^2, 2, \star^1) \quad \textit{and} \quad R^{todo} = \{3, 6, 8, 4, 7, 9, 5\}$$

that is based on the data in Example 4.2.1. *The transport times* t_i *are* 2 *for all operations* i *and all empty moving times are set to* 1. *A corresponding schedule to follow the bound calculations more easily can be found in Figure* 4.2.

Starting with $S_0 = 0$, *the earliest time operation* 1 *can start is* $S_1 = \max(S_0 + e_{0,0}, S_0 +$

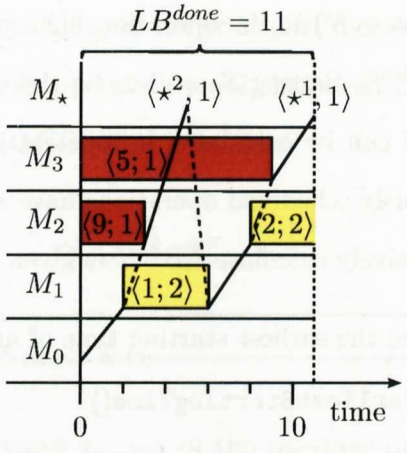


Figure 4.2: Gantt-chart to exemplify calculation of LB^{done}

$p_0^{\min}) + t_1 = \max(0, 0) + 2 = 2$. *Since the predecessor of* $\star^2$ *has not been scheduled yet, the only time we can add to the lower bound after planning* $\star^2$ *is* $e_{1,9} + t_{\star^2} = 3$. *For the next operation* 2, *the predecessor* 1 *has already been started at time* $S_1 = 2$ *which means its earliest starting time is* $S_2 = \max(S_{\star^2} + e_{\star^2,1}, S_1 + p_1^{\min}) + \tau_2 = \max(6, 5) + 2 = 8$. *Finally, we get* $LB^{done} = S_{\star^1} = 11$.

The next step is to give a lower bound for the remaining operations in R^{todo}. They are divided into three different types depending on machines, robot, and height. Each of these bounds will be exemplified using the same *PRC* as before.

Machine Dependent Bound

Lower bounds LB_k^M $(k = 1, \ldots, m)$ for scheduling the remaining time needed on each machine M_k can be obtained as follows. Let Ω_k be the set of all operations i on a specific machine $M(i) = M_k$ and $\Omega_k^{todo} = \{i \in \Omega^\star \mid M(i) = M_k \wedge \tau_i \in R^{todo}\}$ the subset of the operations that have not been scheduled yet ($\tau_i \in R^{todo}$). The bound relies on the minimal processing times, transport times and possible empty moving and waiting times of the operations on a machine.

- **processing times:** All operations have to stay for at least $p_i^{\min}$ time units on their machine. Hence, if neither i nor $suc(i)$ have been started yet, the complete minimal processing time still has to be done. Since the problem is cyclic, it is possible that, at the beginning of a cycle, a job that has started in the previous cycle is still on the machine. Therefore, let $\gamma_i \in \{0,1\}$ $(i \in \Omega)$ be equal to 1, if operation i is such an overlapping operation and 0 else. In our example $\gamma_5 = \gamma_9 = 1$ and all others are 0. In the following we consider four different cases, distinguishing whether τ_i and $\tau_{suc(i)}$ are in R^{todo} or R^{done}. Each of these cases is then divided into classes determined by whether i is overlapping or not.

  1. $\tau_i \in R^{done}$: In this case, we also have to distinguish if $\tau_{suc(i)}$ has been scheduled or not.

     - $\tau_{suc(i)} \in R^{done}$: If i is not overlapping then the minimal processing time is already contained in the bound LB^{done}, so the interesting case is the one, where i is overlapping. Thus, i has been processed from the beginning of the cycle until it has been unloaded at time $S_{suc(i)} - t_{suc(i)}$. This leaves a remaining processing time of

       $$p_i^{\min} - (S_{suc(i)} - t_{suc(i)}).$$

       We, moreover, have to decrease the remaining processing time by the time span between the start of the operation (S_i) and the current finishing time for all operations in R^{done} (LB^{done}), which then adds up to

       $$p_i^{\min} - \underbrace{(S_{suc(i)} - t_{suc(i)})}_{\text{start of cycle}} - \underbrace{(LB^{done} - S_i)}_{\text{end of cycle}}. \tag{4.49}$$

     - $\tau_{suc(i)} \in R^{todo}$: In this case, operation i has already started its processing

($\tau_i \in R^{done}$), but has not been taken off the machine yet ($\tau_{suc(i)} \in R^{todo}$). Note that this is only possible if i is not overlapping. Otherwise, $\tau_{suc(i)}$ must have been scheduled before. In the non overlapping case, the remaining processing time is

$$p_i^{\min} - (LB^{done} - S_i), \tag{4.50}$$

where $LB^{done} - S_i$ indicates the time that the operation has already been processed for.

In our previous example, operation $i = 2$ is one of those. It did start at time $S_2 = 5$ and has been processed for $7 - 5 = 2$ time units, which leaves a remaining minimum processing time of 5 time units.

Note that the last part of the (4.49) is the same as in (4.50). Because of this, we can combine these two differences by using the overlapping indication variable γ_i. In both cases, i has already started its processing in the actual cycle. If i is overlapping ($\gamma_i = 1$) then $\tau_{suc(i)}$ must also have been started, and we additionally have to decrease $p_i^{\min}$ by $S_{suc(i)} - t_{suc(i)}$. Hence, a joint formulation is given by

$$p_i^{\min} - \gamma_i(S_{suc(i)} - t_{suc(i)}) - (LB^{done} - S_i). \tag{4.51}$$

Note that this term can be negative. Since we want to include it in a lower bound, we define

$$\Delta_1^M(i) = \max(p_i^{\min} - \gamma_i(S_{suc(i)} - t_{suc(i)}) - (LB^{done} - S_i), 0). \tag{4.52}$$

2. $\tau_i \in R^{todo}, \tau_{suc(i)} \in R^{done}$: This case can only occur, when i is overlapping, since τ_i will appear after $\tau_{suc(i)}$ in the final robotic cycle. The time span i

has been processed for so far is $S_{suc(i)} - t_{suc(i)}$ (from the beginning of the cycle until the time i has been unloaded). Thus, the remaining processing time is given by

$$p_i^{\min} - (S_{suc(i)} - t_{suc(i)}), \tag{4.53}$$

Note that $S_{suc(i)} - t_{suc(i)}$ can be greater than $p_i^{\min}$, because $p_i^{\min}$ is the minimal processing time and not the actual time p_i the job stays on machine $M(i)$. Hence, for the lower bound we again define an expression $\Delta_2^M(i)$ that keeps (4.53) non-negative:

$$\Delta_2^M(i) = \max(p_i^{\min} - (S_{suc(i)} - t_{suc(i)}), 0). \tag{4.54}$$

3. $\tau_i, \tau_{suc(i)} \in R^{todo}$: In this last case τ_i and $\tau_{suc(i)}$ have not been planned yet. For a non-overlapping operation i which has not been started yet, the whole minimum processing time $p_i^{\min}$ is still needed. We can imply that i is not overlapping if on the same machine $M(i)$ an operation j has already been processed in this cycle. Otherwise, if i is overlapping then i must occupy $M(i)$ during the time interval $[0, LB^{done}]$ and no other job could have been processed in that time. Therefore we define $\nu_k \in \{0, 1\}$ which is 0 if no operation in R^{todo} can be overlapping on machine M_k and 1 otherwise (in the previous example $\nu_1 = \nu_2 = \nu_3 = 0$ since every machine had already processed an operation). There are two situtations in which $\nu_k = 0$ holds:

   - A job has already been processed on M_k in this cycle.
   - For any operation i with $M(i) = M_k$ either Case 1 ($\tau_i \in R^{done}$) or Case 2 ($\tau_i \in R^{todo}, \tau_{suc(i)} \in R^{done}$) occurs.

   In case that the machine could have an overlapping operation ($\nu_k = 1$) we

obviously do not know, which operation this might be. Therefore, we have to assume that it is the one with the longest processing time since this will decrease the lower bound the most. If $LB^{done} \geq p_i^{min}$ then operation i can completely be processed from the beginning of the cycle until LB^{done}. Thus we have to decrease the lower bound by the complete minimal processing time. On the other hand, if $LB^{done} < p_i^{min}$ then we only have to decrease the lower bound by LB^{done}. Summarising this, the lower bound has to be decreased by

$$\Delta_3^M = \min\left(\max_{i\in\Omega_k^{todo}} p_i^{\min}, LB^{done}\right). \tag{4.55}$$

Combining these bounds we finally get the bound

$$LB_k^p = \sum_{\substack{i\in\Omega_k \\ \tau_i\in R^{done}}} \Delta_1^M(i) + \sum_{\substack{i\in\Omega_k \\ \tau_i\in R^{todo} \\ \tau_{suc(i)}\in R^{done}}} \Delta_2^M(i) + \sum_{\substack{i\in\Omega_k \\ \tau_i,\tau_{suc(i)}\in R^{todo}}} p_i^{\min} - \nu_k\Delta_3^M.$$

for all $k = 1,\ldots,m$. Note that the substraction of $\nu_k\Delta_3^M$ ensures that in case the machine could have an overlapping operation, at most this processing time from the beginning of the cycle until now will be taken off the lower bound.

In our example the lower bounds are

$$LB_1^p = p_3^{\min} + p_6^{\min} + p_8^{\min} = 17,$$
$$LB_2^p = p_2^{\min} - (LB^{done} - S_2) + p_4^{\min} + p_7^{\min} + p_9^{\min} - (S_{suc(9)} - t_{suc(9)}) = 21,$$
$$LB_3^p = p_5^{\min} - (S_{suc(5)} - t_{suc(5)}) = -1.$$

Since a negative lower bound is not of any use in this case, we set $LB_3^p = 0$.

The complexity of calculating this bound is $O(n)$, since in the worst case $\Omega_k^{todo} =$

Ω_k for all $k = 1, \ldots, m$ and therefore every operation $i \in \Omega$ has to be incorporated to the bound calculation. Note that there is no need to check if a machine already has processed a job for the current PRC. One can simply keep track of this in constant time during the search.

- **transport times:** Every job on M_k also needs to be transported to M_k and after its processing it has to be transported to its succeeding machine. During this time, no other job can be processed on M_k. Thus, we can increase the lower bound by the sum of all transport moves to and away from the machine.

$$LB_k^t = \sum_{i \in \Omega_k^{todo}} \begin{cases} t_i, & \text{if } \tau_{suc(i)} \in R^{done}; \\ t_i + t_{suc(i)}, & \text{else;} \end{cases}$$

for all $k = 1, \ldots, m$. In our example, these lower bounds are

$$LB_1^t = t_3 + t_4 + t_6 + t_7 + t_8 + t_9 = 12,$$
$$LB_2^t = t_3 + t_4 + t_5 + t_7 + t_8 + t_9 = 12,$$
$$LB_3^t = t_5 = 2.$$

Again, the complexity to calculate these bounds is $O(n)$.

- **empty moving and waiting:** After the robot has transported a job to a machine it can either stay and wait at the machine or move empty to another one. During that time the machine, to which the robot will transport a job next, must be empty. We distinguish between the two cases

  a. empty move: Assume τ_i is the next task of the robot, transporting a job from $M(pre(i))$ to $M(i)$. If the robot is not already waiting at $M(pre(i))$ then it has to perform an empty move from its actual machine to $M(pre(i))$ which is at least as long as the shortest possible move from any machine

M_k to $M(pre(i))$. This machine M_k can only be one, on which a non yet scheduled operation will be processed, or the one where the last operation $l = R^{done}[n+N-|R^{todo}|]$ has been brought to. Therefore let $M(j) = M_k \neq M_i$ with $j \in l \cup \Omega^{todo}$. For every unscheduled operation, the robot needs at least to perform such a minimal empty move, which is of length:

$$\min_{\substack{j \in l \cup \Omega^{todo} \\ M(i) \neq M(j)}} e_{j,pre(i)}.$$

Note that this value is the same for all operations $j \in \Omega_k^{todo}$, since they are all processed on the same machine $M(j) = M_k$ and i is fixed. In our example, the minimal empty moving time is always 1.

b. waiting: In the case that two succeeding transport operations belong to the predecessor of an operation and the operation itself $(\tau_{pre(i)}, \tau_i)$, the robot would have waited at machine $M(pre(i))$ after performing $\tau_{pre(i)}$. During the processing time of $pre(i)$, no other job could have been on machine $M(i)$, since otherwise $M(i)$ would have been blocked. Hence, there is an additional waiting time for $M(i)$ of length

$$p_{pre(i)}^{\min}.$$

In the Gantt-chart of Figure 4.2 this is for instance the case on machine M_2, where operation 3 on M_1 has to be finished before any other operation on M_2 can start. So, there is a gap of length $p_3^{min} = 8$ between operation 2 and 4.

For every operation on M_k that is left to schedule, one of these two cases will occur. A lower bound for every transport move is given by the minimum of those cases. Finally, after the last operation has been scheduled, the robot needs to

return to the machine of the predecessor of the first operation in the robotic cycle, which we can assume is operation 1, and therefore the input station M_0. Summing up, a lower bound for the empty moving and waiting time is given by

$$LB_k^e = \left(\sum_{i \in \Omega_k^{todo}} \min \left(p_{pre(i)}^{\min}, \min_{\substack{j \in l \cup \Omega^{todo} \\ M(i) \neq M(j)}} e_{j,pre(i)} \right) \right) + \min_{i \in \Omega^{todo}} e_{i,pre(1)}.$$

Again, we take the minimum, because we don't know which case might occur in the final schedule.

In our previous example, these empty moving bounds are dominated by the short empty moving times and therefore

$$LB_1^e = 4, \qquad LB_2^e = 4 \qquad \textit{and} \qquad LB_3^e = 2.$$

The complexity of this bound is $O(nm)$. There are at most n operation left in Ω^{todo}. For each i of Ω^{todo} one need to find the minimum empty moving time from a machine $M(j)$ to $M(pre(i))$. (Note that $e_{j,pre(i)}$ refers to the empty moving time between two machines and does not directly depend on the operations itself.) Since there are at most m machines $M(j)$ to check, this leads to a complexity of $O(nm)$. However, one can decrease it to $O(n \log m)$ by ordering these empty moving times $e_{j,pre(i)}$ in advance by their durations for every machine $M(pre(i))$. This initial sorting would have a complexity of $O(m \cdot m \log m)$. (For every machine the distances to every other machine need to be sorted.) The last minimum empty moving time $\min_{i \in \Omega^{todo}} e_{i,pre(1)}$ can also be calculated in $O(\log m)$.

It follows that an overall lower bound for a machine M_k $(k = 1, \ldots, m)$ is

$$LB_k^M = LB_k^p + LB_k^t + LB_k^e \tag{4.56}$$

and the overall machine depending lower bound is

$$LB^M = \max_{k=1,\ldots,m} LB_k^M. \tag{4.57}$$

For our example this leads to $LB_1^M = 17 + 12 + 4 = 33$, $LB_2^M = 21 + 12 + 4 = 37$, $LB_3^M = 0 + 2 + 2 = 4$ and therefore $LB^M = 37$.

The complexity of this bound is the sum over the complexities of all sub bounds which is $O(n \log m)$ plus an initial one-time cost of $O(m \cdot m \log m)$ for sorting the empty moving times.

Robot Dependent Bound

The robot can also be seen as a machine that can only process (transport) one job at a time. Every job needs to be transported to its remaining machines. Thus, there is a minimum time of $\sum_{i \in \Omega^{todo}} t_i$. After a transport operation, the robot either waits at the machine or drives empty to another one. So, it at least needs the minimum time between the processing time of i and the empty moving time to another (different) machine for picking up a job, that still has to be processed. This is

$$\min \left(p_i^{\min}, \min_{\substack{j \in l \cup \Omega^{todo} \\ M(j) \neq M(i)}} e_{i,pre(j)} \right).$$

Finally, at the end of each cycle the robot needs to drive back to the input station, which adds a time of $\min_{i \in \Omega^{todo}} e_{i,pre(1)}$. Thus, the overall minimal time the robot needs to

perform the leftover tasks is

$$LB^R = \left(\sum_{i \in \Omega^{todo}} t_i + \min \left(p_i^{\min}, \min_{\substack{j \in l \cup \Omega^{todo} \\ M(j) \neq M(i)}} e_{i,pre(j)} \right) \right) + \min_{i \in \Omega^{todo}} e_{i,pre(1)}.$$

In our example, this bound would be $LB^R = 14 + 7 + 1 = 22$.

The robot dependent bound has the same complexity as the empty moving bound which is $O(nm)$.

Height Dependent Bound

If we have an overlapping operation this operation is partially processed at the beginning and at the end of the cycle. Both parts belong to different repetitions of the same job. In difference to a machine a job can have more than one overlapping operation. If we have the situation that the maximum height has been reached for a specific job in R^{done} then the remaining operations of this job have to be planned according to their precedence constraints. Note that from this point on, scheduling the remaining operations has to be done in the same way as in a non-cyclic job-shop scheduling problem with makespan minimisation. All operations of the same job have to be processed in order of their precedence constraints and no additional operation is allowed to be overlapping. There are also cases in which no additional operations of a job can be overlapping anymore (and therefore has to be scheduled according to their precedence constraints) even if the maximum height of a job J_j has not been reached yet. This case occurs when every machine M_k, on which the job has still to be processed on, had already processed an operation at an earlier point in time.

In the following we only calculate the height dependent bound for jobs J_j for which all values γ_i with $J(i) = J_j$ are known at this point. This means that for operations i of these jobs with $\tau_i \in R^{done}$ the value for $\gamma_i \in \{0, 1\}$ is known and for all remaining

operations i with $\tau_i \in R^{todo}$ it holds $\gamma_i = 0$. Note again that we exclude the cases in which we do not know whether i is overlapping or not.

The height dependent bound LB_j^H is calculated individually for each of such jobs J_j. The basic idea is as follows. For a non-cyclic problem a bound for each job is given by the sum of all processing times plus the corresponding transport moves, since all operations have to be done according to their precedence constraints. For the cyclic case assume that operation i' is the only overlapping operation of J_j and 0^j (respectively $\star^j$) indicates the first (respectively last) operation of J_j. Then in the current cycle

- all operations from 0^j to i' have to be processed in order of their precedence constraints and
- all operation from $suc(i')$ to $\star^j$ have to be processed in order of their precedence constraints.

Each of those two *sequences* builds a lower bound for the minimal completion time of the job. The first one is given by $\sum_{k=0^j}^{pre(i')}(t_k + p_k^{\min}) + t_{i'} + \max\left(p_{i'}^{\min} - (S_{suc(i')} - t_{suc(i)}, 0\right)$ whereas the second one is given by $(S_{suc(i')} - t_{suc(i)}) + \sum_{k=suc(i')}^{\star^j}(t_k + p_k^{\min})$. Thereby the parts

$$\max\left(p_{i'}^{\min} - (S_{suc(i')} - t_{suc(i)}), 0\right) \text{ and } (S_{suc(i')} - t_{suc(i)})$$

indicate the two partial processings of operation i' at the end and at the beginning of the cycle.

In case a job has more than one overlapping operation, there are more than two sequences where each builds a lower bound for the completion time of the job. Therefore, we calculate those lower bounds for each sequence and finally take the maximum as the final height dependent bound LB_j^H of job J_j.

Procedure 4.2.2 shows how such a bound can be calculated. We start with setting LB_j^H and a variable LB^{temp} for a temporary bound to 0. In the for-loop (lines 4-19)

Procedure 4.2.2 Calculates the height dependent lower bound for a job J_j

1: **procedure** `CalculateHeightBound`(J_j)
2: $LB_j^H = 0$
3: $LB^{temp} = 0$
4: **for** i = first operation of J_j to $pre(\star^j)$ **do**
5: **if** $\tau_i, \tau_{suc(i)} \in R^{done}$ and $\gamma_i = 1$ **then**
6: $LB_j^H = \max\left(LB_j^H, p_i^{\min} - (S_{suc(i)} - t_{suc(i)})\right)$
7: **end if**
8: **if** $\tau_i \in R^{done}$, $\tau_{suc(i)} \in R^{todo}$ and $\gamma_i = 0$ **then**
9: $LB^{temp} = p_i^{\min} - (LB^{done} - S_i)$
10: **end if**
11: **if** $\tau_i, \tau_{suc(i)} \in R^{todo}$ and $\gamma_i = 0$ **then**
12: $LB^{temp} = LB^{temp} + t_i + p_i^{\min}$
13: **end if**
14: **if** $\tau_i \in R^{todo}$, $\tau_{suc(i)} \in R^{done}$ and $\gamma_i = 1$ **then**
15: $LB^{temp} = LB^{temp} + t_i + \max\left(p_i^{\min} - (S_{suc(i)} - t_{suc(i)}), 0\right)$
16: $LB_j^H = \max\left(LB_j^H, LB^{temp}\right)$
17: $LB^{temp} = 0$
18: **end if**
19: **end for**
20: **if** $\tau_{\star^j} \in R^{todo}$ **then**
21: $LB^{temp} = LB^{temp} + t_{\star^j}$
22: $LB_j^H = \max\left(LB_j^H, LB^{temp}\right)$
23: **end if**
24: **return** LB_j^H
25: **end procedure**

we consider all operations of J_j in order of their precedence constraints. We add the processing and transport times of every sequence to LB^{temp} until an overlapping operation (or the last operation) has been reached and then continue with a new sequence. In particular for every operation i five different cases can occur.

1. $\tau_i, \tau_{suc(i)} \in R^{done}$ and i is overlapping (line 5): This case is the same as in (4.49). Thus we update LB_j^H by the maximum of itself and $p_i^{\min} - (S_{suc(i)} - t_{suc(i)})$.

2. $\tau_i \in R^{done}$, $\tau_{suc(i)} \in R^{todo}$ and i is not overlapping (line 8): This case is the same as in (4.50). Thus we set the temporary bound LB^{temp} to $p_i^{\min} - (LB^{done} -$

S_i). Note that after operation i has been finished the succeeding operations $suc(i), suc(suc(i)), \ldots, \star^j$ of i have to be processed as well. If none of these operations is an overlapping one (remember that we know this at this point) then all operations have to be executed one after each other in the remaining cycle. Therefore, we store the current lower bound in LB^{temp} and we will increase this value (cf. Case 3) until we have reached another overlapping operation (cf. Case 4) or the last operation of the job (cf. Case 5).

3. $\tau_i, \tau_{suc(i)} \in R^{todo}$ and i is not overlapping (line 11): In this case, we are in the middle of a sequence and know that i is completely processed in the remaining part of the cycle after its predecessor has been finished. Therefore the transport time as well as the minimal processing time can be added to the temporary bound ($LB^{temp} = LB^{temp} + t_i + p_i^{\min}$).

4. $\tau_i \in R^{todo}$, $\tau_{suc(i)} \in R^{done}$ and i is overlapping (line 14): Since i is overlapping, the sequence of operations processed after each other in the cycle will finish at operation i. Thus we cannot add the complete minimal processing time $p_i^{\min}$ to LB^{temp} but only the transport time and the remaining processing time which is $t_i + \max\left(p_i^{\min} - (S_{suc(i)} - t_{suc(i)}), 0\right)$. Again this is the same case as in (4.49). Since $p_i^{\min} - (S_{suc(i)} - t_{suc(i)})$ can be negative we use the maximum of it and 0. Because i has been the last operation in a sequence of operations processed one after each other in the cycle, the succeeding operations of i $(suc(i), suc(suc(i)), ...)$ do not start after i in this cycle. Thus, these operation cannot influence the temporary lower bound LB^{temp} which know builds the bound for one complete sequence. Hence, we update the best lower bound LB_j^H by the maximum of itself and LB^{temp} and reset LB^{temp} to 0 again to possibly calculate another bound for a sequence of operations that have to be processed according to their precedence constraints in the remaining cycle. Note that after this case, either Case 1 or 2

will occur.

5. $\tau_{\star^j} \in R^{todo}$ (line 20): If the last operation $\star^j$ of J_j has been reached we add its transport time $t_{\star^j}$ to the current temporary lower bound LB^{temp} and update LB_j^H as before.

The overall bound for the height is given by $\max_{j=1}^{N} LB_j^H$.

Considering our example the following holds:

$$LB_1^H = \underbrace{p_2^{\min} - (LB^{done} - S_2)}_{Case\ 2} + \underbrace{t_3 + p_3^{\min} + t_4 + p_4^{\min}}_{Case\ 3} + \underbrace{t_5 + p_5^{\min} - (S_{\star^1} - t_{\star^1})}_{Case\ 4} = 21$$

$$LB_2^H = \underbrace{t_6 + p_6^{\min} + t_7 + p_7^{\min} + t_8 + p_8^{\min}}_{Case\ 3} + \underbrace{p_9^{\min} - (S_{\star^2} - t_{\star^2})}_{Case\ 4} = 22,$$

and therefore $LB^H = 22$.

The complexity of this bound is as follows. Every operation of a job is considered in the for loop of Procedure 4.2.2. The if-statements are all checked in constant time. Since there are n operations to be checked the bound can be calculated in $O(n)$ time.

Summarising, a lower bound for the complete *PRC* is given by

$$LB = LB^{done} + \max(LB^M, LB^R, LB^H), \tag{4.58}$$

which in our case is $LB = 11 + \max(37, 15, 22) = 48$.

The efficiency of each of these these bounds is mostly depending on the problem instance. The robot dependent bound for example is not of much use if the travel times of the robot are very small compared to the processing times. That means the robot would not be a bottleneck in the production process. On the other hand, if the moving

times are large compared to the processing times and the fact that between each two operations a transport move has to be done, this bound has a lot more influence on the overall lower bound.

The height dependent bound is only of any use, if the maximum height is relatively small (or will be reached fast), since then the lower bound will be not just dependent on the chains of operations on each machine, but also on the chain of processing times of the operations of each job.

For the instances we have considered, one can in general say that the machine dependent bound is the most dominant one. This holds especially at the beginning of the construction phase.

4.2.3 Search Strategy

In branch and bound methods the solving time and solution quality significantly depend on the strategy with which the branching tree is explored. On the one hand, we want a good quality solution and, on the other hand we need solutions fast enough to make the bounding more efficient.

The strategy we have chosen for our approach works as follows. To every *PRC* we assign a score, which gives an indication of the potential quality of a complete resulting robotic cycle. Obviously, the quality of the solution depends on the extra time each operation stays on its machine ($p_i - p_i^{\min}$) and the machine idle times (the time the machines are empty). For example, let us examine two extreme cases. Consider a solution, where all jobs are sequentially performed after each other. The order of the operations for each job is the same as the order of the precedence constraints. This means that there is at most one machine loaded at any time during the production process and all other machines are empty at this time. The robot is always waiting at a machine while a job is processed and only performs empty moves from the output to

the input station. Furthermore, each job stays on a machine exactly for its minimal processing time. Note that such a solution is always feasible. However, the machine utilisation is the worst possible one (assuming that the cycle time is minimised) which means, that the machines have long idle times.

On the other hand, if we try to keep the machine idle time very low, by reloading a machine as soon as it has been unloaded, the possibility to get an infeasible *PRC* is very high since many machines are blocked. Also, the jobs on a machine might stay there for longer than needed. E.g. if a set of machines is loaded with jobs whose successors have all to be processed on the same machine M_k, then they obviously have to visit M_k one after each other. This means that a part of this machine set will be blocked for a (most likely) longer time than needed since the jobs cannot be unloaded. Furthermore, no other job can go on those machines in the meantime, which also increases the cycle time.

Based on this, we assume that a schedule with short cycle time has less idle time and less operation waiting time than a schedule with a long cycle time. Therefore, we define the overall waiting time ω of a partial robotic cycle as the sum of the machine idle times and the operation waiting times in R^{done}. The smaller this time is, the potentially better the final solution will be.

The order in which the partial robotic cycles are explored during the search is based on a priority rule. It depends on the waiting time ω and the lower bound LB^{done} of every *PRC*. The later a *PRC* appears in the search tree, the more likely it is to have a high waiting time, since more operations have been scheduled already. Therefore, we want to relativise the waiting time of a *PRC* to its depth in the tree. A simple way of doing this is to set it in relation to the lower bound LB^{done}. The deeper a *PRC* is in the tree, the higher this value gets. Therefore, the priority rule is based on a *scoring*

function that is given by

$$\frac{\omega}{LB^{done}}.$$

After every iteration, we branch on the node containing the PRC with the lowest score. If two PRCs have the same score than the one with the smallest lower bound is chosen next.

An advantage of the search procedure is, that it has great potential to be parallelised. Different partial robotic cycles can be updated by different threads, where the only variables to be synchronised are the best global solution and the global lower bound. Since multi core computers are very common those days and concurrent containers are available in most standard libraries of common programming languages like Java, C++ or Python, parallel computing becomes an important factor in modern algorithms.

4.3 A Heuristic Approach

In the area of operational research heuristics refer to algorithms which deliver acceptable solutions to a problem. They are usually based on a 'rule of thumb' or a set of rules derived from some experience or insight into the problem by an expert. Although, tending to work well in practice heuristics do not give any guarantee to the quality of obtained solutions. There are two main types of heuristics: constructing and improving ones. A constructive heuristic builds solutions from scratch and compares them against each other, while improving heuristics (or *local search methods*) take an existing solution and try to find a better solution by perturbing a previous one. The construction method for robotic cycles in Section 4.2.1 can be seen as a constructing heuristic. However, in the remaining part of this section we will consider a local search method (more precisely, a tabu search heuristic) for the CJSPTB. A detailed overview about heuristics in general can be found in Michalewicz and Fogel (2004) and Burke and Kendall (2005).

4.3.1 Neighbourhoods

As before, solutions are presented as robotic cycles, which means our search space is the set of all feasible robotic cycles. A very important component of local search methods is the *neighbourhood function*. For a given solution the neighbourhood function determines a set of different solutions (neighbours) that have been derived from the original one. From this set of neighbours one solution (often one with a good objective value) is chosen as the current one. Then the neighbours of this current solution are calculated and evaluated. One selects a solution within these neighbours as the new current one and continues.

Since the CJSPTB can be regarded as a permutation problem one can move from a given robotic cycle R to another one, by swapping two transport moves in R, for

instance. Common changes for the non-cyclic job-shop problem with transport and no blocking (JSPT) are to

- move all operations of one job to the end of the schedule (job shift),
- swap the order of two successive transport operations by the robot (robot swap) or
- swap the order of two successive operations processed on the same machine (machine swap).

In the following we are going to adapt these neighbourhood moves to the CJSPTB. However, by randomly swapping the position of two transport moves in R, the resulting neighbour might be an infeasible robotic cycle. Thus, we have to be a bit more 'careful' by choosing which transport operations are being moved or swapped. Since we are aiming to generate feasible neighbours, we always have to check whether they are blocking-feasible and fulfill the height restriction. In our case, the blocking feasibility is the more restricting property and, therefore, we will discuss this part more intensively. The height of a blocking-feasible robotic cycle can easily be calculated by keeping track of the individual repetition numbers during the different neighbourhood moves.

Job Shift

For the non-cyclic job-shop problem with transportation and blocking one can always move from a given feasible solution to another feasible one, by shifting all operations of one job to the end of the schedule in order of their precedence constraints. We can use this idea for the CJSPTB as well. Let J_j be the job to be shifted to the end of the schedule. The pseudo code of such a shift is shown in Procedure 4.3.1.

In the first step, we remove all transport moves belonging J_j from the robotic cycle (cf. line 2). Note that the remaining robotic cycle R' is still feasible. Firstly, the

Procedure 4.3.1 Performing a job shift

1: **procedure** `JobShift`(J_j)
2: remove transport moves of job J_j from R
3: let R' be the remaining robotic cycle
4: consider $\mathcal{M}_j \subseteq M$ of all blocking machines on which J_j is processed on
5: **while** there exists $l \in \Omega$ with $M(l) \in \mathcal{M}_j$ and $\tau_{suc(l)} \dashrightarrow \tau_l$ in R' **do**
6: move $\tau_{suc(i)}$ to the last position of R'
7: **end while**
8: reinsert operations of J_j at the end of R' in order of precedence constraints
9: **return** R'
10: **end procedure**

height will not be increased by removing a job. And secondly, for every removed transport move τ_i its successor $\tau_{suc(i)}$ will also be removed (if it exists) and therefore with Lemma 4.2.1 (page 148) the robotic cycle is still blocking-feasible. Before we can add the eliminated transport moves of J_j at the end of the R', we have to make sure that this would not violate the conditions of Lemma 4.2.1. In particular, we have to make sure that for all machines M_k, job J_j is going to be processed on, the following holds. Consider an operation $l \notin J_j$ with $M(l) = M_k$. If the order of τ_l and its successor in R' is $\tau_l \dashrightarrow \tau_{suc(l)}$ then inserting $\tau_i, \tau_{suc(i)}$ with $J(i) = J_j$ and $M(i) = M_k$ at the end of R' would lead to $\tau_l \dashrightarrow \tau_{suc(l)} \dashrightarrow \tau_i \dashrightarrow \tau_{suc(i)}$. This would be blocking-feasible on M_k according to Lemma 4.2.1. On the other hand, if the order in R' is $\tau_{suc(l)} \dashrightarrow \tau_l$ then inserting $\tau_i, \tau_{suc(i)}$ in order of their precedence constraints would lead to $\tau_{suc(l)} \dashrightarrow \tau_l \dashrightarrow \tau_i \dashrightarrow \tau_{suc(i)}$ which cannot lead to a blocking-feasible robotic cycle. Hence, for all those transport moves τ_l one has to make sure that its successor appears in front of τ_i. This can be done by moving the successor of τ_l to the end of R' (cf. line 6). However, by moving a transport move $\tau_{suc(l)}$ to the end of the robotic cycle it can happen that R' contains the order $\tau_{suc(suc(l))} \dashrightarrow \tau_{suc(l)}$ and $M(suc(l))$ is a machine where J_j has to be processed on. Thus, $\tau_{suc(suc(l))}$ also has to be moved to the end of R'. In the worst case this is done for all succeeding transport moves of a job. Finally the transport moves of J_j are inserted at the end of R'.

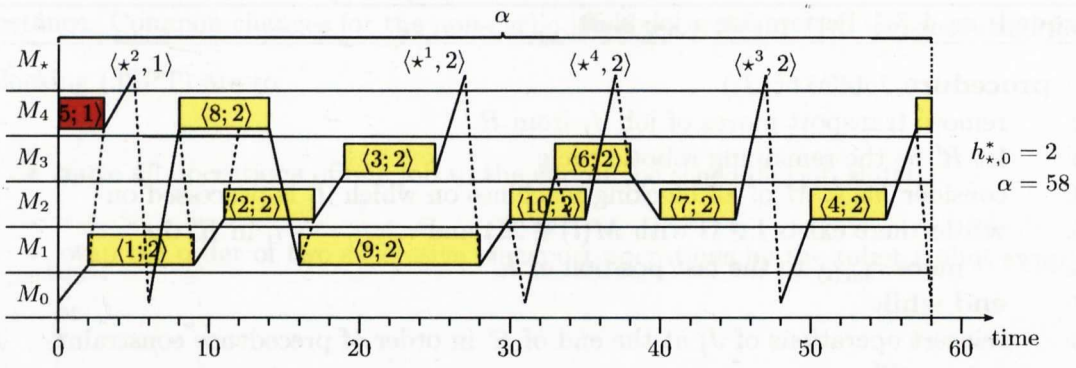


Figure 4.3: Gantt-chart for Example 4.3.1 with robotic cycle R

We will demonstrate the job shift with an example.

Example 4.3.1. *Consider the following data of a CJSPTB with* 4 *jobs and* 4 *machines.*

Job	J_1			J_2		J_3		J_4		
Operation	1	2	3	4	5	6	7	8	9	10
Processing time	5	3	6	5	4	5	6	6	8	5
Machine	M_1	M_2	M_3	M_2	M_4	M_3	M_2	M_4	M_1	M_2

The transport times are $t_i = 2$ *for all* $i \in \Omega^\star$ *and the empty moving times between any two different machines are* 1. *Figure* 4.3 *shows a feasible schedule for the problem. The robotic cycle in this solution is*

$$R = \begin{array}{cccccccccccccc} M_1 & M_\star & M_4 & M_2 & M_1 & M_3 & M_\star & M_2 & M_3 & M_\star & M_2 & M_\star & M_2 & M_4 \\ \tau_1, & \tau_{\star^2}, & \tau_8, & \tau_2, & \tau_9, & \tau_3, & \tau_{\star^1}, & \tau_{10}, & \tau_6, & \tau_{\star^4}, & \tau_7, & \tau_{\star^3}, & \tau_4, & \tau_5. \end{array}$$

(The corresponding machine is written above each transport move to help the reader identifying possible blocking situation in the next steps.) We now want to perform a job shift for J_4 *in this solution. We start with removing transport moves* $\tau_8, \tau_9, \tau_{10}, \tau_{\star^4}$ *from the robotic cycle. This leaves a remaining robotic cycle*

$$R' = \begin{array}{cccccccccc} M_1 & M_\star & M_2 & M_3 & M_\star & M_3 & M_2 & M_\star & M_2 & M_4 \\ \tau_1, & \tau_{\star^2}, & \tau_2, & \tau_3, & \tau_{\star^1}, & \tau_6, & \tau_7, & \tau_{\star^3}, & \tau_4, & \tau_5. \end{array}$$

The machines J_4 has to be processed on are $\mathcal{M}_4 = \{M_1, M_2, M_4\}$. Considering machine M_4 we have the situation that τ_5 with $M(5) = M_4$ is done after $\tau_{\star^2}$ in R'. If we would insert τ_8, τ_9 at the end of R' in its current state then the order $\tau_{\star^2}$ --→ τ_5 --→ τ_8 --→ t_9 would violate the conditions in Lemma 4.2.1 for machine M_4. According to our procedure we move $\tau_{\star^2}$ to the end of R' and then insert the transport moves of J_4. The resulting feasible robotic cycle is

$$\begin{array}{rccccccccccccc} & M_1 & M_2 & M_3 & M_\star & M_3 & M_2 & M_\star & M_2 & M_4 & M_\star & M_4 & M_1 & M_2 & M_\star \\ R'' = & \tau_1, & \tau_2, & \tau_3, & \tau_{\star^1}, & \tau_6, & \tau_7, & \tau_{\star^3}, & \tau_4, & \tau_5, & \boldsymbol{\tau_{\star^2}}, & \boldsymbol{\tau_8}, & \boldsymbol{\tau_9}, & \boldsymbol{\tau_{10}}, & \boldsymbol{\tau_{\star^4}}. \end{array}$$

Note that all jobs are processed one after each other and there are no overlapping operations. However, this is not necessarily the case. If we apply a job shift to J_3 in R, the resulting robotic cycle would be

$$\begin{array}{rccccccccccccc} & M_1 & M_\star & M_4 & M_2 & M_1 & M_3 & M_\star & M_2 & M_\star & M_2 & M_4 & M_3 & M_2 & M_\star \\ R''' = & \tau_1, & \tau_{\star^2}, & \tau_8, & \tau_2, & \tau_9, & \tau_3, & \tau_{\star^1}, & \tau_{10}, & \tau_{\star^4}, & \tau_4, & \tau_5, & \boldsymbol{\tau_6}, & \boldsymbol{\tau_7}, & \boldsymbol{\tau_{\star^3}}. \end{array}$$

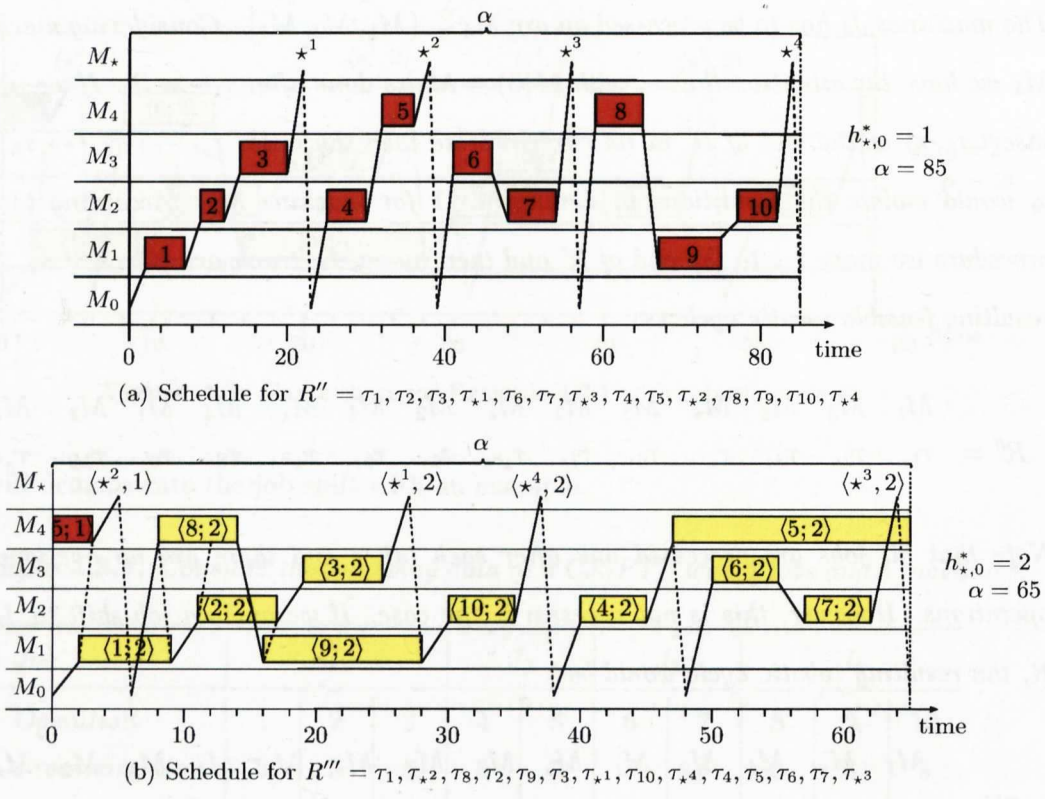
The corresponding schedule for both robotic cycles are shown in Figure 4.4. In Figure 4.4(a) we omit the repetition numbers since they are all identical.

Robot Swap

For a given robotic cycle $R = \tau_{\sigma(1)}, \tau_{\sigma(2)}, \ldots, \tau_{\sigma(k)}, \tau_{\sigma(k+1)}, \ldots, \tau_{\sigma(N+n)}$ a robot swap between transport moves $\tau_{\sigma(k)}$ and $\tau_{\sigma(k+1)}$ leads to the neighbour robotic cycle

$$R' = \tau_{\sigma(1)}, \tau_{\sigma(2)}, \ldots, \tau_{\sigma(k+1)}, \tau_{\sigma(k)}, \ldots, \tau_{\sigma(N+n)}.$$

Formally, we swap the transport moves at position k and $k+1$. Note that we only swap transport moves that are direct successors of each other. In other words, τ_i is swapped with $\tau_{suc^R(i)}$. In case $k = N + n$ is the last position in the robotic cycle, we cannot



(a) Schedule for $R'' = \tau_1, \tau_2, \tau_3, \tau_{\star^1}, \tau_6, \tau_7, \tau_{\star^3}, \tau_4, \tau_5, \tau_{\star^2}, \tau_8, \tau_9, \tau_{10}, \tau_{\star^4}$

(b) Schedule for $R''' = \tau_1, \tau_{\star^2}, \tau_8, \tau_2, \tau_9, \tau_3, \tau_{\star^1}, \tau_{10}, \tau_{\star^4}, \tau_4, \tau_5, \tau_6, \tau_7, \tau_{\star^3}$

Figure 4.4: Gantt-charts for Example 4.3.1 after job shift

swap it with its "right hand neighbour". Because of the cyclic property the right hand neighbour of $\tau_{\sigma(N+n)}$ would be $\tau_{\sigma(1)}$. Since we have fixed the robotic cycle to start with transport move $\tau_{\sigma(1)} = \tau_1$, we cannot just swap these two moves, but also have to readjust the robotic cycle, so that it starts with τ_1 again. Hence, swapping transport moves $\tau_{\sigma(1)}$ and $\tau_{\sigma(N+n)}$ in the previously given robotic cycle R would lead to the new robotic cycle

$$R'' = \tau_{\sigma(1)}, \tau_{\sigma(N+n)}, \tau_{\sigma(2)}, \ldots, \tau_{\sigma(N+n-1)}.$$

However, not every robot swap provides a feasible neighbour. Let us recall that a feasible robotic cycle needs to be blocking-feasible and has to fulfill the height restriction (cf. Section 3.2). We are now going to discuss in which cases a swap will lead to a feasible or infeasible robotic cycle.

We start with the height restriction. We know that the actual height $h^*_{\star,0}$ depends on the different job repetitions processed at a time. A solution can only become infeasible if a job J_j with repetition number r will finish after another job with repetition number $r + h_{\star,0}$ has already started processing. Thus we simply need to keep track of the repetition numbers during the neighbourhood moves.

To ensure that the neighbour is blocking-feasible, for any two operation $i, j \in \Omega$ with $M(i) = M(j)$ the cyclic order $\tau_i \dashrightarrow \tau_{suc(i)} \dashrightarrow \tau_j \dashrightarrow \tau_{suc(j)}$ must be preserved in a robotic cycle (cf. Lemma 4.2.1). Thus, we can conclude the following.

Corollary 4.3.1. *Consider a blocking-feasible robotic cycle R containing τ_i, τ_j with $M(i) = M(j) \neq M_\star$. Then swapping any two transport moves in*

$$\tau_i \dashrightarrow \tau_{suc(i)} \dashrightarrow \tau_j \dashrightarrow \tau_{suc(j)}$$

leads to an infeasible robotic cycle.

Due to Corollary 4.3.1 in the following cases a swap between τ_k and τ_{k+1} leads to a blocking-infeasible sequence if for $M(i) = M(j)$ one of the following cases occurs:

Case A: $\tau_k = \tau_i$ and $\tau_{k+1} = \tau_{suc(i)}$, or

Case B: $\tau_k = \tau_{suc(i)}$ and $\tau_{k+1} = \tau_j$, or

Case C: $\tau_k = \tau_j$ and $\tau_{k+1} = \tau_{suc(j)}$, or

Case D: $\tau_k = \tau_{suc(j)}$ and $\tau_{k+1} = \tau_i$.

Assuming that R was blocking-feasible before, the resulting robotic cycle R' will be blocking-feasible after a robot swap if none of the cases above will occur. We will again use an example to illustrate a robot swap.

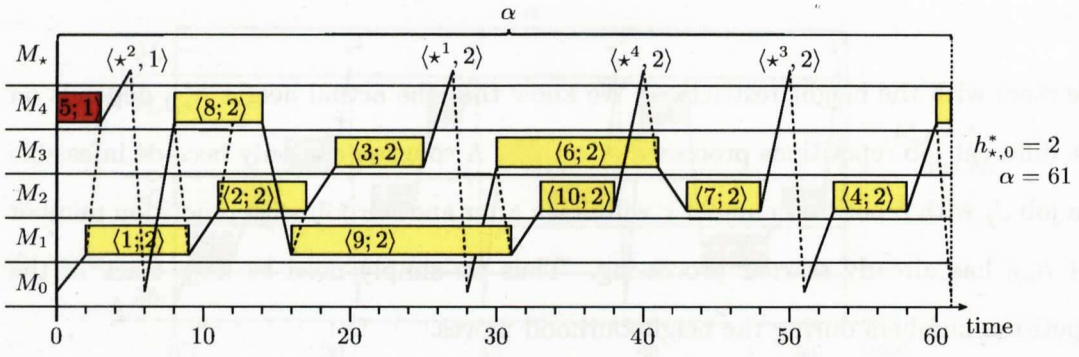


Figure 4.5: Gantt-chart for Example 4.3.2 with robotic cycle R'''

Example 4.3.2. *We will use the same data as in Example* 4.3.1 *and start with the same feasible solution given by*

$$R = \begin{array}{cccccccccccccc} M_1 & M_\star & M_4 & M_2 & M_1 & M_3 & M_\star & M_2 & M_3 & M_\star & M_2 & M_\star & M_2 & M_4 \\ \tau_1, & \tau_{\star^2}, & \tau_8, & \tau_2, & \tau_9, & \tau_3, & \tau_{\star^1}, & \tau_{10}, & \tau_6, & \tau_{\star^4}, & \tau_7, & \tau_{\star^3}, & \tau_4, & \tau_5 \end{array}$$

and shown in Figure 4.3. *We want to shift transport move* τ_6 *as far to the left as possible by using robot swaps. Operation* 6 *is processed on* M_3. *The first swap is between* τ_6 *and* τ_{10}, *where operation* 10 *is processed on* M_2. *This swap is feasible. The next swap would be with* $\tau_{\star^1}$. *However,* $\star^1$ *is the direct successor of operation* 3 *which is also processed on* M_3 *(cf. Case* B*). Thus, a swap would lead to an infeasible sequence*

$$\tau_3 \dashrightarrow \tau_6 \dashrightarrow \tau_{suc(3)} \dashrightarrow \tau_{suc(6)}.$$

Hence, a the resulting robotic cycle after swapping τ_6 *to the left is given by*

$$R''' = \begin{array}{cccccccccccccc} M_1 & M_\star & M_4 & M_2 & M_1 & M_3 & M_\star & M_3 & M_2 & M_\star & M_2 & M_\star & M_2 & M_4 \\ \tau_1, & \tau_{\star^2}, & \tau_8, & \tau_2, & \tau_9, & \tau_3, & \tau_{\star^1}, & \boldsymbol{\tau_6}, & \boldsymbol{\tau_{10}}, & \tau_{\star^4}, & \tau_7, & \tau_{\star^3}, & \tau_4, & \tau_5. \end{array}$$

The corresponding schedule is shown in Figure 4.5. *In the same way, one can obviously also shift a transport move to the right in a robotic cycle. Moreover, for every transport move* τ_i *there is an interval of transport moves* τ_i *can be shifted in using some robot*

swaps.

Machine Swap

Another common swap in job-shop scheduling is to change the order of two operations processed on the same machine. While this swap is rather trivial for the general job-shop problem, it is not that obvious how to retrieve a feasible robotic cycle for the CJSPTB after a machine swap.

Assume that τ_i appears before τ_j in R where $M(i) = M(j)$. Instead of swapping τ_i and τ_j we move $J(i)$ immediately after $\tau_{suc(i)}$ (see steps 1, 2 below). Usually this leads to a robotic cycle that is not blocking-feasible. Therefor we need to repair the robotic cycle which is described in Steps 3 and 4 of the following.

1. Remove all operations of $J(i)$ from the robotic cycle. The remaining partial robotic cycle is still feasible.

2. Insert all transport moves of $J(i)$ after $\tau_{suc(j)}$ in order of their precedence constraints. We denote the resulting (most likely infeasible) robotic cycle with R'.

3. Let τ_{i^0} resp. $\tau_{i^\star}$ be the first resp. last transport move of $J(i)$. Shift all transport moves $\tau_{suc(j')}$ belonging to $J(j)$ with $\tau_{i^\star} \dashrightarrow \tau_{suc(j')}$ in front of τ_{i^0} by keeping the original order between themselves, if they fulfill the following conditions:

   - $J(i)$ has to be processed on $M(j')$,
   - R' contains the order $\tau_{j'} \dashrightarrow \tau_{i^0}$.

4. Consider a transport move τ_k neither belonging to $J(i)$ nor $J(j)$. After Steps 2 and 3 have been executed operation k can collide with another operation l on machine $M(k) = M(l)$ such that the current robotic cycle is infeasible (the first time l belongs to either $J(i)$ or $J(j)$). To correct this, we shift $\tau_{suc(k)}$ as far to the left as

needed until the there is a blocking-feasible order $\tau_k \dashrightarrow \tau_{suc(k)} \dashrightarrow \tau_l \dashrightarrow \tau_{suc(l)}$ on $M(k) = M(l)$. During this shift, there could occur new collisions on some machines. We continue with this correction until the robotic cycle is blocking-feasible (cf. explanation below).

Step 1 is the same as in the job shift. Step 2 is similar to the job shift, except job $J(i)$ will inserted immediately after $\tau_{suc(j)}$ rather than at the end of the robotic cycle.

In Step 3, we make sure that all operations belonging to $J(j)$, that collide with $J(i)$ on any machine in the current schedule are finished before $J(i)$ starts. Assume that i' and j' are operations belonging to $J(i)$ and $J(j)$, respectively that are processed on the same machine $M(i') = M(j')$. After reinserting the operation of $J(i)$ we know that the resulting robotic cycle R' contains the sequence $\tau_{i^0}, \ldots, \tau_{pre(i')}, \tau_{i'}, \tau_{suc(i')}, \ldots, \tau_{i^\star}$. Together with the two transport moves $\tau_{j'}$ and $\tau_{suc(j')}$ the following cyclic orders could be possible in R':

$$\tau_{j'} \dashrightarrow \tau_{i^0}, \ldots, \tau_{i^\star} \dashrightarrow \tau_{suc(j')} \quad \text{or} \quad \tau_{j'} \dashrightarrow \tau_{suc(j')} \dashrightarrow \tau_{i^0}, \ldots, \tau_{i^\star}.$$

The latter case would be feasible for machine $M(i') = M(j')$, whereas the first case would be infeasible. However, we can make this robotic cycle blocking-feasible for this machine by moving $\tau_{suc(j')}$ in front of τ_{i^0} to retrieve the situation in the latter case. In the most extreme case the complete job $J(j)$ will be finished before $J(i)$ starts. Note that the order between the moved transport moves will remain the same as before. If we would remove all other transport moves not belonging to $J(i)$ or $J(j)$ the remaining robotic cycle would be feasible. However, there could be other transport moves τ_k not belonging to $J(i)$ or $J(j)$ which are now in conflict with transport moves of $J(i)$ or $J(j)$.

In step 4 let τ_k be such a transport move. The following two cases can occur between

τ_k and τ_l and their successors with $M(k) = M(l)$:

$$\tau_k \dashrightarrow \tau_l \dashrightarrow \tau_{suc(k)} \dashrightarrow \tau_{suc(l)} \text{ or } \tau_k \dashrightarrow \tau_l \dashrightarrow \tau_{suc(l)} \dashrightarrow \tau_{suc(k)}.$$

We shift the successor $\tau_{suc(k)}$ to the left such that the resulting order in the robotic cycle is $\tau_k \dashrightarrow \tau_{suc(k)} \dashrightarrow \tau_l \dashrightarrow \tau_{suc(l)}$ with $M(k) = M(l)$. We repeat this procedure until R' is blocking-feasible. Note that this method will eventually lead to a blocking-feasible robotic cycle. Any shift of a transport move $\tau_{suc(k)}$ will, in the worst case, result in the order $\ldots, \tau_k, \tau_{suc(k)}, \ldots$ in R'. If every operation of $J(k)$ is colliding with another operation on a machine then the final sequence of all transport moves belonging to $J(k)$ in R' would be $\tau_{k^0}, \ldots, \tau_k, \tau_{suc(k)}, \ldots, \tau_{k^\star}$ which means all operations of the job are processed one after each other. If this case happens for all jobs except $J(i)$ and $J(j)$ the result would be a blocking-feasible robotic cycle in which all jobs are processed directly one after each other.

Since this move provides a lot of diversification between the original robotic cycle and its resulting neighbour many repetition numbers can change. Hence the height restriction needs to be checked again.

Example 4.3.3. *We again will use the same data as in Example* 4.3.1. *The given robotic cycle is* $R = \tau_1, \tau_{\star^2}, \tau_8, \tau_2, \tau_9, \tau_3, \tau_{\star^1}, \tau_{10}, \tau_6, \tau_{\star^4}, \tau_7, \tau_{\star^3}, \tau_4, \tau_5$. *We are going to swap operations* 8 *and* 5 *(resp.* τ_8 *and* τ_5*) that are both processed on* M_4. *The swap can be done as follows*

1. *We remove the transport moves belonging to* J_2 *(*$\tau_4, \tau_5, \tau_{\star^2}$*) from the robotic cycle. The route of the remaining partial robotic cycle is*

$$\begin{array}{rccccccccccc} & M_1 & M_4 & M_2 & M_1 & M_3 & M_\star & M_2 & M_3 & M_\star & M_2 & M_\star \\ R' = & \tau_1, & \tau_8, & \tau_2, & \tau_9, & \tau_3, & \tau_{\star^1}, & \tau_{10}, & \tau_6, & \tau_{\star^4}, & \tau_7, & \tau_{\star^3}. \end{array}$$

2. *We reinsert the transport moves of* J_2 *after* $\tau_{suc(8)} = \tau_9$ *in order of their precedence*

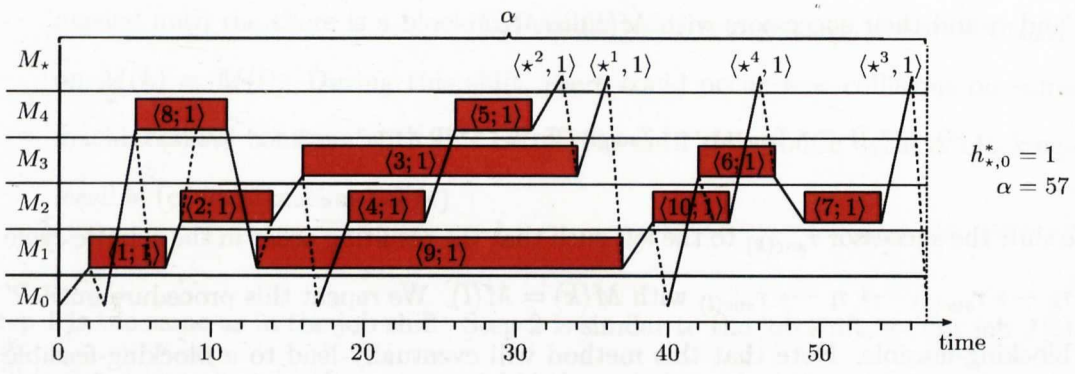


Figure 4.6: Gantt-chart for Example 4.3.3 with robotic cycle R'''

constraints. The resulting robotic cycle is

$$R'' = \begin{array}{cccccccccccccc} M_1 & M_4 & M_2 & M_1 & M_2 & M_4 & M_\star & M_3 & M_\star & M_2 & M_3 & M_\star & M_2 & M_\star \\ \tau_1, & \tau_8, & \tau_2, & \tau_9, & \boldsymbol{\tau_4}, & \boldsymbol{\tau_5}, & \boldsymbol{\tau_{\star^2}}, & \tau_3, & \tau_{\star^1}, & \tau_{10}, & \tau_6, & \tau_{\star^4}, & \tau_7, & \tau_{\star^3}. \end{array}$$

3. *Job $J(5) = J_2$ is processed on machine M_2 and M_4 and $J(8) = J_4$ on M_4, M_1 and M_2. The only non-dummy transport move belonging to J_4 scheduled after $\tau_{\star^2}$ in R' is τ_{10}. However the predecessor of operation 10 is processed on M_1 which is not in the set of machine J_2 is processed on. Formally said, there are no transport moves $\tau_{suc(j)}$ with $M(j) \in \{M_2, M_4\}$ of J_4 that appear after $\tau_{\star^2}$ in R'.*

4. *The remaining transport move not belonging to J_4 is τ_3. Note that for M_2 we have the order $\tau_2 \dashrightarrow \tau_4 \dashrightarrow \tau_5 \dashrightarrow \tau_3$ which is infeasible according to Lemma* 4.2.1. *Thus we move τ_3 to the first position such that the order on M_2 is blocking-feasible again. This position is in front of τ_4, since $\tau_2 \dashrightarrow \tau_3 \dashrightarrow \tau_4 \dashrightarrow \tau_5$ is a blocking-feasible order for M_2. After that, there is no blocking-infeasibility on any machine. Thus, the final blocking-feasible robotic cycle is given by*

$$R''' = \begin{array}{cccccccccccccc} M_1 & M_4 & M_2 & M_1 & M_3 & M_2 & M_4 & M_\star & M_\star & M_2 & M_3 & M_\star & M_2 & M_\star \\ \tau_1, & \tau_8, & \tau_2, & \tau_9, & \boldsymbol{\tau_3}, & \tau_4, & \tau_5, & \tau_{\star^2}, & \tau_{\star^1}, & \tau_{10}, & \tau_6, & \tau_{\star^4}, & \tau_7, & \tau_{\star^3}. \end{array}$$

The corresponding schedule can be found in Figure 4.6.

The presented neighbourhood moves can now be used to explore the search space. As mentioned before, there are various strategies of how this could be done. In the following section we will present a tabu search strategy to solve the CJSPTB.

4.3.2 A Tabu Search

The possible simplest way of a local search method is hill climbing. For a given feasible solution, one calculates all its neighbours and choose the one with the best objective value as the new solution. One continues until no further improvement can be made. A disadvantage of this method is that it will usually stuck in a local optimum. There are different ways of escaping such a local optimum. A common one is to accept the best solution of all neighbours as the new solution, even if it is not better than the best one found so far (global optimum), which will be stored separately. In addition to that a quite promising method is the tabu search (cf. Glover (1997)). Briefly said, it is a hill climber with memory. There is a list of fixed length l in which the last l previously visited 'states' of the search are stored. If during the search, a state will be reached, that is in the tabu list, one will ignore it. Even then it will be ignored if the resulting neighbour is the best of the current set of neighbours. This will avoid going in short cycles and possibly escape local optimas. In addition to this one can also use an aspiration criteria. This means, we will accept solutions if they are better than the global optimum, even if they are tabu.

The next step is to decide, which neighbours are calculated for a current solution. Of course, one could calculate all possible neighbours, using the neighbourhood moves provided. However, this might take a long time. Since the aim is to improve the current solution, it is manifest to apply those neighbourhood moves that might improve the current solution. The objective value does not necessarily depend on all operations in a solution. Moreover, it is dominated by those operations that build the bottlenecks

in the current solution. The property of those operations is that the solution cannot be improved if none of the bottleneck operations will be moved. Thus we restrict the neighbourhood search, to only swap and shift those transport moves, where the corresponding operations builds a bottleneck for the current solution.

The last important point is the tabu list. One has to decide, what is going to be stored in it, and how long the list should be. The simplest elements to store in the list are complete solutions, in our case this would be a robotic cycle. However, checking whether the current neighbour is contained in the list could take up some time, depending how long the list is. Therefore, instead of this, we store the previous positions of the swapped or shifted transport moves. For our approach we chose three different tabu lists TL_J, TL_R and TL_M, each defined for the three different neighbourhood moves.

- TL_J stores the jobs that have been moved last to the end of the schedule after a job shift.
- TL_R stores the pairs of transport moves that have been swapped in a robot swap.
- TL_M stores the pairs of jobs that have been swapped in a machine swap.

The search strategy is as follows. We start with a feasible robotic cycle R, which in our case is simply all jobs are processed one after each other in ascending order of their indices. This robotic cycle is then evaluated and defines the current best global solution. Then, the operations on the critical path(s) are calculated. For every operation i on a critical path, the following neighbours are calculated and evaluated.

- $J(i)$ is shifted to the end of the schedule (this is at most done once for every job).
- τ_i is swapped with the transport moves left and right of it in R.
- i is swapped with the preceding and the succeeding operation on $M(i)$ (if it exists).

All neighbours are evaluated and the best neighbour, that is not tabu, is stored as the best local solution and added to the corresponding tabu list. In case there are neighbours with the same objective value, we choose the one that has been calculated first during our search. Note that this keeps the method purely deterministic, which is good for reproducing the results. However, a random choice of the neighbour could act as a good tiebreaker and might influence the final solution in either a good or bad way. If the current solution is better than the best one found so far (global optimum), then we replace this solution by the current one. Note that we also store a solution that actually was tabu, if it is better than the global optimum. The search stops, if after a specific number of iterations no improving solution could be found, or a time limit has been reached. (We will specify these values in Section 4.5.)

4.4 Generating Problem Instances

For the cyclic job-shop problem with one transport robot, there are, as far as we know no standard benchmarks available. Most authors have considered standard job-shop benchmarks and have randomly added additional times for transportation. To be able to test any algorithm on different classes of problem instances (especially different ratios between processing times and transport times), we developed a problem generator (Brucker et al. (2009)).

The underlying pseudo random number generator (PRNG) is based on Park and Miller (1988). Thus, every problem instance can be reproduced if the program is run with the same parameters. The input parameters are the number of jobs N, the number of machines m (including an input and output machine) and a seed number for the PRNG. For every job, the generator randomly assigns a machine to each operation. Furthermore, one can set minimal and maximal values for the processing, setup, empty moving and transport time. A reasonable assumption is that the transport time cannot be smaller than the corresponding empty moving time between the same machines. Therefore, we add a random value between a minimal and maximal additional transport time to the corresponding empty moving time.

The processing times are calculated as follows. There is a lower bound for the smallest minimal processing time and an upper bound for the largest maximal processing time. Additionally, a minimal distance between minimal and maximal processing time can be set (processing time window).

Since the triangle inequality has to hold, we determine the distances between the machines in the following way. We create a 2-dimensional quadratic area with diameter equal to the difference between minimum and maximum empty moving time. Then, we randomly place all machines on this area, calculate their euclidian distances to each

other and add the minimal empty moving time. This guarantees a distance between the machines according to the given limits. We assume that every job starts at input machine M_0 and finishes at the output machine $M_\star = M_{m-1}$. In our case, the minimal processing and setup times of the output machine are set to 0 and the maximum processing time to a big enough number.

Here is an example of a small problem instance with 2 jobs and 5 machines.

```
*SEED 212121
*MIN,MAX_PROCESSING_TIME 10,99
*MIN_PROCESSING_TIME_WINDOW 20
*MIN,MAX_TRANSPORT_TIME 1,4
*MIN,MAX_EMPTY_MOVE_TIME 1,8
*MIN,MAX_SETUP_TIME 4,8
*The first line represents the numbers of jobs (2) followed by the number
*of machines (5). Each of the next 2 line(s) represent all operations of one job.
*Each operation has assigned 5 values which are in the following order:
*Machine | minProcessingTime | maxProcessingTime | transportTime | setupTime
*The other lines representing the time distance
*between the machines are of the form:
*machine-A | machine-B | distance
2 5                                                    // #jobs #machines
3 42 74 6 8 2 75 97 9 6 1 73 95 8 7 4 0 9999 5 0 // operations job 1
2 27 56 5 7 3 25 68 7 8 1 48 94 8 5 4 0 9999 5 0 // operations job 2
0 0 0                                                  // empty moving times
0 1 5
0 2 4
0 3 5
0 4 5
1 1 0
1 2 6
1 3 5
1 4 7
2 2 0
2 3 3
2 4 2
3 3 0
3 4 4
4 4 0
```

4.5 Computational Results

In this section we describe some implementation details and report some computational results. All approaches have been tested on an Intel Xeon E5472 3.0GHz computer with 16GB memory, single threaded, running Linux 64bit. The mathematical programming models have been solved with CPLEX 12.2 using the default parameters. The branch and bound method (Section 4.2) and the tabu search (Section 4.3) have been implemented in C++ using the Intel compiler version 11.1.

We will start with presenting the test data used for our experiments. Afterwards, the MIP-model from the literature (Section 4.1.1) is compared with our new MIP-formulation (Section 4.1.2). In addition to the best solutions, we will also provide the lower bounds found during the search and the memory used by the solver, since this is an important criteria, what problem sizes the solver can handle. Afterwards, we will compare the best solutions and bounds of the two MIP-models with our branch and bound method. Finally, we will present the best results out of all methods and compare it with the tabu search heuristic. To model the cyclic constraint, we have chosen the cyclic job-shop model with height $h_{\star,0}$. The reason for this is, that the model with job-chain repetition does not have much variation in the solution for different heights for our problem instances. This is, because in our instances (and also the ones in the literature for problems without transportation) every job is processed on every machine. Hence, even for a small number of jobs, every job usually overlaps at most once. Thus, there is no real point in testing different parameters for the height, since the optimal solutions stay the same. For the complexity of solving the problem it also does not make much of a difference. All three models have constraints that restrict a specific repetition of a job to start before another one has finished. They do not change anything substantial in the overall problem formulation. The parameters for the height are set to $h_{\star,0} = 1$ and $h_{\star,0} = 2$. We have also done all experiments with a maximum height of $h_{\star,0} = 3$

(and even higher where possible), but the optimal results kept the same, in case we could find it.

The time limit that every method had to solve each problem instance was set to 3600 seconds, which is also a standard value in the literature.

4.5.1 Test Data

The problem instances used in the experiments are generated with the problem generator proposed in Section 4.4. We have generated 27 instances of various sizes that are shown in Table 4.2. The complete data of the instances can be found in Brucker et al. (2009).

Instance	#Jobs	#Machines	#Operations
5x5-1	5	5	20
5x5-2	5	5	20
5x5-3	5	5	20
5x10-1	5	10	45
5x10-2	5	10	45
5x10-3	5	10	45
6x6-1	6	6	30
6x6-2	6	6	30
6x6-3	6	6	30
7x7-1	7	7	42
7x7-2	7	7	42
7x7-3	7	7	42
8x8-1	8	8	56
8x8-2	8	8	56
8x8-3	8	8	56
9x9-1	9	9	72
9x9-2	9	9	72
9x9-3	9	9	72
10x5-1	10	5	40
10x5-2	10	5	40
10x5-3	10	5	40
10x10-1	10	10	90
10x10-2	10	10	90
10x10-3	10	10	90
15x15-1	15	15	210
15x15-2	15	15	210
15x15-3	15	15	210

Table 4.2: Data sets

4.5.2 MIP-Models

In this part, we compare the mixed integer programming formulation from the literature, presented in Section 4.1.1, to the new developed one presented in Section 4.1.2. The results can be found in Tables 4.3 and 4.4. The first two columns contain the problem instance and the corresponding height $h_{\star,0}$. The first set of columns contains the results for the model from the literature (CJSPT-MIP-LIT) and the second set (CJSPT-MIP-OO) those from the newly developed model presented within this work. The minimal cycle time obtained by CPLEX ('α') is given in the next column, followed by the lower bound ('LB') and the corresponding gap ('GAP'). The last column ('Memory') contains the space in megabyte for storing the tree in case the problem could not be solved to optimality when the time limit has been reached. If no value for a specific column could be obtained (e.g. no solution has been found) we write 'inf'.

As we can see, our new model mostly outperforms the model from the literature. It finds a better solution for 16 instances and a better lower bound in 21 cases. The model from the literature on the other hand finds a better solution value for 2 instances and a better lower bound for 11 instances. Furthermore, the differences in the memory needed for the branching trees are enormous. The average memory needed for the CJSPT-MIP-LIT model is 2751MB, not taking into account instances solved to optimality, whereas the CJSPT-MIP-OO model only uses an average of 289MB per unsolved instance. This new formulation especially benefits scenarios, where computers with less memory or 32Bit operating systems are used. However, for large instance with more than 200 operations, none of the models was able to find any feasible solution. One can also notice, that the problem seems to become more complicated, when the height has been increased. Obviously, the main reason is that the search space has been increased.

Instance	$h_{\star,0}$	CJSPT-MIP-LIT α	LB	Gap	Memory	CJSPT-MIP-OO α	LB	Gap	Memory
5x5-1	1	519	519	0.0%	-	519	519	0.0%	-
	2	508	508	0.0%	-	508	508	0.0%	-
5x5-2	1	446	446	0.0%	-	446	446	0.0%	-
	2	432	432	0.0%	-	432	432	0.0%	-
5x5-3	1	519	519	0.0%	-	519	519	0.0%	-
	2	482	482	0.0%	-	482	482	0.0%	-
5x10-1	1	694	694	0.0%	-	694	694	0.0%	-
	2	560	512	9.4%	-	**558**	**558**	0.0%	-
5x10-2	1	765	765	0.0%	-	765	765	0.0%	-
	2	**545**	**545**	0.0%	-	558	461	15.5%	178
5x10-3	1	696	696	0.0%	-	696	696	0.0%	-
	2	524	504	4.0%	665	524	**524**	0.0%	-
6x6-1	1	616	616	0.0%	-	616	616	0.0%	-
	2	575	575	0.0%	-	575	575	0.0%	-
6x6-2	1	605	605	0.0%	-	605	605	0.0%	-
	2	548	548	0.0%	-	548	548	0.0%	-
6x6-3	1	559	559	0.0%	-	559	559	0.0%	-
	2	532	532	0.0%	-	532	532	0.0%	-
7x7-1	1	683	682.7	0.0%	-	683	683	0.0%	-
	2	623	532	17.1%	1242	**620**	**620**	0.0%	-
7x7-2	1	735	735	0.0%	-	735	735	0.0%	-
	2	644	612	5.2%	339	**634**	**634**	0.0%	-
7x7-3	1	676	676	0.0%	-	676	676	0.0%	-
	2	621	539	15.2%	1350	**618**	**618**	0.0%	-
8x8-1	1	948	947	0.1%	3	948	**948**	0.0%	-
	2	880	572	53.8%	3540	**865**	**611**	29.4%	327
8x8-2	1	887	792	12.0%	400	**884**	**884**	0.0%	-
	2	811	634	27.9%	1818	**804**	**649**	19.3%	456
8x8-3	1	852	802	6.2%	1665	852	**852**	0.0%	-
	2	800	**524**	52.7%	3337	**796**	487	38.9%	427
9x9-1	1	1175	722	62.7%	3839	**1050**	**1033**	1.6%	418
	2	inf	498	inf	4387	inf	**656**	inf	494
9x9-2	1	1188	**728**	63.2%	3867	**1034**	714	30.9%	453
	2	inf	**501**	inf	4463	inf	480	inf	387
9x9-3	1	1119	764	46.5%	3808	**1097**	**1097**	0.0%	-
	2	**1272**	**665**	91.3%	3582	inf	525	inf	408
10x5-1	1	819	819	0.0%	-	819	819	0.0%	-
	2	779	649	20.0%	373	779	**779**	0.0%	-
10x5-2	1	818	818	0.0%	-	818	818	0.0%	-
	2	810	551	47.0%	1350	**808**	**808**	0.0%	-
10x5-3	1	844	844	0.0%	-	844	844	0.0%	-
	2	798	572	39.5%	1597	798	**798**	0.0%	-

Table 4.3: Results for MIP models part 1

Instance	$h_{\star,0}$	CJSPT-MIP-LIT α	LB	Gap	Memory	CJSPT-MIP-OO α	LB	Gap	Memory
10x10-1	1	1403	803	42.8%	5749	**1389**	803	42.2%	93
	2	inf	**446**	inf	2910	inf	346	inf	270
10x10-2	1	inf	**822**	inf	5367	**1444**	800	44.6%	54
	2	inf	362	inf	967	inf	**383**	inf	297
10x10-3	1	inf	720	inf	6572	**1441**	**777**	46.1%	521
	2	inf	310	inf	3863	inf	**318**	inf	391
15x15-1	1	inf	**953**	inf	2979	inf	927	inf	85
	2	inf	**722**	inf	3122	inf	289	inf	201
15x15-2	1	inf	942	inf	3099	inf	**959**	inf	89
	2	inf	**483**	inf	4322	inf	330	inf	209
15x15-3	1	inf	830	inf	2255	inf	**860**	inf	100
	2	inf	**401**	inf	5204	inf	295	inf	209

Table 4.4: Results for MIP models part 2

4.5.3 Branch and Bound

In this section, we have applied the branch and bound method from Section 4.2 to solve the problem instances. For comparison, we have used the best solutions and best bounds found in the previous experiments. The results are presented in Tables 4.5 and 4.6.

As we can see from the results, our method is competitive with CPLEX. For large instances, we were able to find (non trivial) solutions were CPLEX was not able to find any solution. Moreover, we could also find better solutions for some instances. Even by changing the parameter of CPLEX to find feasible instead of optimal solution ($CPX_PARAM_MIPEMPHASIS$) the overall result did not change much. It in fact became worse than the current one that uses the default parameter. Moreover, by studying the final schedules, we could see that our solutions (especially for large instances) are also not trivial ones, which in this case means they are not of the type in which the jobs are processed one at a time. However, CPLEX outperforms our method in the bounding. One reason for that is definitely that CPLEX explores the tree more in its breadth, where we are more concentrated on the depth to find more solutions. Another indicator for this is the memory used by the algorithm. As we have seen in the

Instance	$h_{\star,0}$	CJSPT-MIP-Best α	CJSPT-MIP-Best LB	CJSPT-MIP-Best Gap	Branch and Bound α	Branch and Bound LB	Branch and Bound Gap
5x5-1	1	519	519	0.0%	519	519	0.0%
	2	508	508	0.0%	508	508	0.0%
5x5-2	1	446	446	0.0%	446	446	0.0%
	2	432	432	0.0%	432	432	0.0%
5x5-3	1	519	519	0.0%	519	519	0.0%
	2	482	482	0.0%	482	482	0.0%
5x10-1	1	694	694	0.0%	694	694	0.0%
	2	558	558	0.0%	558	558	0.0%
5x10-2	1	765	765	0.0%	765	765	0.0%
	2	**545**	**545**	0.0%	558	461	17.3%
5x10-3	1	696	696	0.0%	696	696	0.0%
	2	524	524	0.0%	524	524	0.0%
6x6-1	1	616	616	0.0%	616	616	0.0%
	2	575	575	0.0%	575	575	0.0%
6x6-2	1	605	605	0.0%	605	605	0.0%
	2	548	548	0.0%	548	548	0.0%
6x6-3	1	559	559	0.0%	559	559	0.0%
	2	532	**532**	0.0%	532	530	0.4%
7x7-1	1	683	683	0.0%	683	683	0.0%
	2	**620**	**620**	0.0%	629	363	42.3%
7x7-2	1	735	**735**	0.0%	735	608	17.3%
	2	634	**634**	0.0%	634	538	15.1%
7x7-3	1	676	**676**	0.0%	676	454	32.8%
	2	**618**	**618**	0.0%	621	377	39.3%
8x8-1	1	948	**948**	0.0%	948	823	13.1%
	2	**865**	**611**	29.4%	884	415	53.1%
8x8-2	1	884	**884**	0.0%	884	702	20.6%
	2	**804**	**649**	19.3%	837	521	37.8%
8x8-3	1	852	**852**	0.0%	852	504	40.8%
	2	**796**	**524**	34.2%	906	349	61.5%
9x9-1	1	1050	**1033**	1.6%	1050	613	41.6%
	2	inf	**656**	inf	**1109**	316	71.5%
9x9-2	1	1034	**728**	30.9%	1034	589	43.0%
	2	inf	**501**	inf	**1028**	480	53.3%
9x9-3	1	1097	**1097**	0.0%	1097	619	43.6%
	2	1272	**665**	91.3%	**1268**	386	62.5%
10x5-1	1	819	**819**	0.0%	819	640	21.9%
	2	779	**779**	0.0%	779	464	17.1%
10x5-2	1	818	**818**	0.0%	818	570	30.3%
	2	808	**808**	0.0%	808	439	46.7%
10x5-3	1	844	**844**	0.0%	844	539	39.0%
	2	798	**798**	0.0%	798	507	36.5%

Table 4.5: Results for best MIP model and branch and bound part 1

Instance	$h_{\star,0}$	CJSPT-MIP-Best α	LB	Gap	Branch and Bound α	LB	Gap
10x10-1	1	1389	**803**	42.2%	**1291**	511	60.4%
	2	inf	**446**	inf	**1396**	346	75.2%
10x10-2	1	1444	**822**	43.1%	**1224**	515	57.9%
	2	inf	**383**	inf	**1788**	228	87.7%
10x10-3	1	1441	**777**	46.1%	**1208**	509	57.9%
	2	inf	**318**	inf	**1665**	288	82.7%
15x15-1	1	inf	**953**	inf	**10681**	599	94.4%
	2	inf	**722**	inf	**7763**	458	94.1%
15x15-2	1	inf	**959**	inf	**10543**	589	94.4%
	2	inf	**483**	inf	**7489**	296	96.0%
15x15-3	1	inf	**860**	inf	**7254**	602	91.7%
	2	inf	**401**	inf	**5428**	351	93.5%

Table 4.6: Results for best MIP model and branch and bound part 2

previous section, the CJSPT-MIP-OO formulation had and average tree size of 289MB after the time limit has been reached, whereas the branch and bound method used less than 20MB.

4.5.4 Tabu Search

The last method to test is the tabu search presented in Section 4.3. An important variable for such a heuristic is the length of the tabu list, or in our case the lengths of the three lists TL_J, TL_R and TL_M. A too short tabu list could lead to getting stuck in the area of a local optimum, whereas a too long tabu list more likely leaves out the neighbours that would lead to an optimal solution. (Note that we are not storing complete solutions, but only the last neighbourhood moves.) We have tested various sizes for the lengths l_{TL} of the tabu lists. In particular:

- length of TL_J: $l_{TL_J} = N \cdot k$,
- length of TL_J: $l_{TL_R} = n \cdot k$,
- length of TL_J: $l_{TL_M} = m \cdot N \cdot k$,

where $k \in \{0.1, 0.15, 0.2, 0.25, 0.3\}$ was used as a scaling factor. The quality of the results was varying for different values of k. Also, there was not a specific value for k that led to the best solution for every instance. However, to get a fair comparison, and not choosing the best parameter for each instance, we have chosen $k = 0.15$, since it on average gave the best solutions.

In Table 4.7 we compare our tabu search method with the best results from all previous methods.

The results clearly show that the tabu search is outperforming all previously applied exact methods, on the data sets we have tested them on. Especially for larger instances where CPLEX and our branch and bound procedure start to struggle. Only the branch and bound method wins on instance 15x15-3 for height 2. However, we do not know anything about the quality of the solution, by just using a heuristic, since no lower bounding is done during the search. Only for the small instances, that the exact methods also solve, we know that the solution is optimal.

Instance	$h_{\star,0}$	Best previous solution	Tabu search solution
5x5-1	1	519	519
	2	508	508
5x5-2	1	446	446
	2	432	432
5x5-3	1	519	519
	2	482	482
5x10-1	1	694	694
	2	558	558
5x10-2	1	765	765
	2	545	545
5x10-3	1	696	696
	2	524	524
6x6-1	1	616	616
	2	575	575
6x6-2	1	605	605
	2	548	548
6x6-3	1	559	559
	2	532	532
7x7-1	1	683	683
	2	620	620
7x7-2	1	735	735
	2	634	634
7x7-3	1	676	676
	2	618	618
8x8-1	1	948	948
	2	865	865
8x8-2	1	884	884
	2	804	804
8x8-3	1	852	852
	2	769	769
9x9-1	1	1050	1050
	2	1109	**947**
9x9-2	1	1034	1034
	2	1268	**953**
9x9-3	1	1097	1097
	2	1268	**931**
10x5-1	1	819	819
	2	779	779
10x5-2	1	818	818
	2	808	808
10x5-3	1	844	844
	2	798	798
10x10-1	1	1291	**1172**
	2	1396	**1163**
10x10-2	1	1224	**1177**
	2	1788	**1101**
10x10-3	1	1208	**1143**
	2	1665	**1252**
15x15-1	1	10681	**2447**
	2	7763	**2440**
15x15-2	1	10543	**2391**
	2	7489	**2547**
15x15-3	1	7254	**3641**
	2	**5428**	5803

Table 4.7: Comparison tabu search and best known solution

Chapter 5

Concluding Remarks

In this thesis, we have studied the cyclic job-shop problem with blocking and one transport robot. We started with presenting the classical non-cyclic job-shop problem, showed how additional constraints like blocking and transportation can be added and finally ended with describing the main problem of the thesis. In the literature review we have seen, that the problem is not well studied by other researchers, which was one reason why we have chosen it for this thesis. Since even evaluating the cycle time for a feasible solution is not a trivial problem, we presented in Chapter 3 two algorithms from the literature and additionally proposed a new algorithm, that outperforms the existing ones in theoretical complexity (except for some special cases) as well as in actual running times on various tested benchmarks. As we have mentioned before the strength of this algorithms seems to be the separation between solving a relaxed problem very fast and correcting the solution if necessary, whereas the correction phase is hardly reached. In our case, the algorithm itself seems to be more complicated (especially the correction parts) and the proof of correctness also is a lot more complex compared to the PCP, for instance. However, it finally seems to pay off. Since the correction phase only applies so rarely, it would be interesting to find out, whether those cases can be

characterised. With this information, one could try to adjust the algorithm in a way such that those cases are already captured during the insertion of the arc pairs and therefore save the time for corrections.

Another question would be, what other problems could be presented with such a graph? Or, if there are similar representations, how can one adjust the algorithm to be applied to other graphs? Additionally, we have studied some properties of the problem. One parameter, that probably is the most characteristic one for cyclic scheduling problem is the "height". In Section 3.1, different interpretations of various height models as well as their influences in feasible solutions have been discussed. An interesting point would be to find the minimum height for a problem with a given cycle time. The height is a very interesting parameter, which has a great range of interpretation. Especially, in connection with the flow time of the jobs, it builds an important factor for practical purposes.

In Chapter 4, we investigated solution methods for the CJSPTB. For comparative reasons, we have used an existing mixed integer programming model for cyclic scheduling problems from the literature and adapted it to model and solve the CJSPTB. Furthermore, we have developed three other solution methods: another integer programming model, a branch and bound procedure and a tabu search heuristic. All of them have been applied to the same data sets and the solutions have been compared.

The computational results have shown, that our developed methods are able to solve reasonably large instances of the CJSPTB. However, every approach has advantages and disadvantages. The integer programming model has been solved with a powerful software (CPLEX). Although, for larger instances (around 10 jobs and 10 machines), CPLEX was sometimes not able to find any solution and if, this could have taken some time. Therefore, it is not appropriate for the case, that a solution is needed very fast. It also should not be forgotten that CPLEX is a quite expensive piece of software.

One possible future direction, would be the definition of cutting planes for the problem. There are many examples in the literature, where sophisticated cutting planes could enormously improve the solving process. A good overview is given in Marchand et al. (2002).

The advantage of branch and bound methods in general is, that they provide a (lower) bound and therefore can prove whether a final solution is optimal or not. However, the bounding part of our method is rather poor compared to CPLEX, for instance. It would be interesting to see, how the search strategy for the can be modified, such that the lower bounds become better and at the same time avoid loosing the ability of finding good solutions. An idea would be to incorporate the search strategy of our branch and bound algorithm into CPLEX. This would turn its general "black box behaviour" into a more sophisticated search strategy and could also make use of the computational power of CPLEX.

Out of all tested methods it is fair to say that the heuristic performed best on the given problem instances. However, there is still room for improvement. A promising adjustment could be the introduction of randomness, which has been shown to work as a good tiebreaker during the local search. Furthermore, it would be interesting to see, how different start solutions influence the performance of the heuristic. A constructive heuristic to create an initial starting solution, for instance, could be developed or even used from the non-cyclic version of the problem.

Even if we have presented three different approaches to solve the CJSPTB there are still many other possibilities to tackle the problem. Since we have been inspired by approaches for non-cyclic scheduling problems, one could try to adapt more existing techniques, which have already performed well for other problems. Constraint propagation, for instance, has shown to be a powerful tool which is used in connection with various solution methods for scheduling problems. It has been successfully applied to

the RCPSP and machine scheduling problems (cf. Brucker (2002)). One could also think of incorporating the presented neighbourhood moves from Section 4.3 into other meta heuristics such as simulated annealing.

Another point is the general model and the theoretical background. We have restricted ourselves to the cyclic job-shop scheduling with blocking and one transport robot. The problem is very specific and therefore only interesting for a small audience. How does the problem change, when the blocking restriction changes to (un)limited buffers? Can our methods easily be adjusted, or are they not of any use? What about flexible job-shops or even open-shop problems or parallel machines? Can we easily include multiple robots and what are additional constraints that might come up (e.g. physical collisions)? Those robots also do not need to be identical. They even could be limited to a fixed set of machines they could circulate between. It is also possible that the jobs do not need transportation after every processing step. One could imagine several working areas with several machines and conveyor belts. The number of variations of problems seems endless.

What about k-cyclic scheduling problems? It is easy to see that for a k-cyclic solution to a problem there is a minimum of one $k+1$ cyclic solution to the same problem that is at least as good as the other one (in the worst case, it is the k-cyclic solution itself). What is the trade off between increasing k, potentially improving the solution and increasing the solving time?

Solution robustness is also already an important factor for practical purposes. How long does is take to recover a production process after a machine failure? Especially, if those machine breakdowns are more likely, one might prefer a robust instead of an optimal solution. For real world applications, also uncertainty plays a big rule. For instance, the robot travel time between two machines can vary, due to wheel slipping

or other external influences. The robust shortest path problem, for instance, deals with those circumstances.

One last important question for us is, whether there are other non-scheduling research areas that could benefit from this work. This tends to be a problem for many research areas. The problems itself seem to be completely different and are based on various backgrounds, but the underlying models and formulations can be very similar. However, there are not many people that have such a comprehensive background to keep on top of all the research and finally see the link between then.

Finally, we again want to point out that, even if the problem is not completely new, not many people have worked on it. We hope to motivate other researchers with our work to look into this area and modify, improve or even just use our methods and solutions.

Appendix A

Pseudo-code for ε-Corrections

We start with Case A. The pseudo-code of this correction can be found in Procedure A.1. While the solution is not optimal we perform an ε-correction. Therefore, we save the arcs belonging to pairs that are critical on the left hand side of the graph (ending in a node of V_1) in list $critListX$ and the ones that are critical on the right hand side (starting in a node of V_2) in $critListY$. Since $(0,j)_{|j,i|}$ and $(i,n^\star)_{|j,i|}$ are critical for sure we initialise the lists as in line 4. In the next two lines we update the arc lengths of $(0,j)_{|j,i|}$ and $(i,n^\star)_{|j,i|}$ and store the multiplier of ε in a variable $k_{(i,n^\star)}$. Since the length of $(u,n^\star)$ cannot be changed, we follow the alternating path along arc $(i,n^\star)_{|j,i|}$. We store this current arc in a variable $curArcY$ and loop through all arcs that are critical with this arc, since their lengths have to be corrected (line 8). If any of these arcs is critical with an arc from $critListX$ or a constant arc $(0,c)$ then the solution is optimal according to Theorem 3.5.2 and we return without any changes. If not, we update the arc lengths as described in the second part of Section 3.5.2. We add all arcs that are critical with $curArcY$ to $critListX$ and also their partner arcs to $critListY$ in case it is itself critical. We repeat this procedure until either the solution is found to be optimal or all critical arcs on the right hand side have been corrected. (This means we have

reached the end of every alternating path.) Now we can calculate the best value for ε and update the graph. The length of the new critical path is supposed to be ε units shorter than the old length, which was

$$x_{|j,i|} + CP_{j,n^\star} = CP_{0,i} + y_{|j,i|}.$$

Therefore we calculate the maximal ε such that every path through the corrected arcs in *critListX* and *critListY* is at most the length of the new critical path length $x_{|j,i|} + CP_{j,n^\star} - \varepsilon$. We update the graph an continue until the solution is optimal.

The correction for Case B is similar. Its pseudo-code can be found in Procedure A.2 and it works in a comparable way as Procedure A.1. The main difference is, that we start with at least two critical paths that can be changed (one through $(0,j)_{|j,i|}$ and one through $(i,n^\star)_{|j,i|}$) so we have to correct along both directions of the alternating path (line 26). If during this correction one side ends in a constant arc $(0,c)$ or $(c,n^\star)$ we continue the correction with Procedure A.1, line 14.

Procedure A.1 ε-correction in case $(0,j)_{|j,i|}$ is critical with $(u,n^\star)$

1: **procedure** `correctLengthsA`(pair $\mid j,i \mid$)
2: given: $(0,j)_{|j,i|}$ is critical with $(u,n^\star)$
3: **while** not optimal **do**
4: $critListX = \{(0,j)_{|j,i|}\}$, $critListY = \{(i,n^\star)_{|j,i|}\}$
5: $x_{|j,i|} = x_{|j,i|} - \varepsilon$, $y_{|j,i|} = y_{|j,i|} + \varepsilon$
6: $k_{(i,n^\star)} = 1$
7: $curArcY$ = first element in $critListY$
8: **for** all arcs $(0,t)_{|t,s|}$ that are critical with $curArcY$ **do**
9: **if** $(s,n^\star)_{|t,s|}$ is critical with an arc from $critListX$ or an arc $(0,c)$ **then**
10: $\varepsilon = 0$
11: **return**
12: **end if**
13: $k_{(s,n^\star)} = k_{curArcY} + 1$
14: $x_{|t,s|} = x_{|t,s|} - k_{(s,n^\star)} \cdot \varepsilon$
15: $y_{|t,s|} = y_{|t,s|} + k_{(s,n^\star)} \cdot \varepsilon$
16: ADD $(0,t)_{|t,s|}$ to $critListX$
17: ADD $(s,n^\star)_{|t,s|}$ to $critListY$
18: $curArcY$ = next element in $critListY$
19: **end for**
20: **for** all arcs $(0,t)_{|t,s|}$ in $critListX$ and $(s,n^\star)_{|t,s|}$ in $critListY$ **do**
21: calculate maximal ε s.t.
22: $x_{|t,s|}(\varepsilon) + \widehat{CP}_{t,n^\star}(\varepsilon) \leq \widehat{CP}^{old}_{0,n^\star} - \varepsilon$
23: $\widehat{CP}_{0,s}(\varepsilon) + y_{|t,s|}(\varepsilon) \leq \widehat{CP}^{old}_{0,n^\star} - \varepsilon$
24: **end for**
25: UPDATE arc lengths
26: **end while**
27: **end procedure**

Procedure A.2 ε-correction in case $(0,j)_{|j,i|}$ is critical with $(u,n^\star)_{|v,u|}$

1: **procedure** `correctLengthsB`(pair $\mid j,i \mid$)
2: given: $(0,j)_{|j,i|}$ is critical with $(s,n^\star)_{|t,s|}$, $(i,n^\star)_{|j,i|}$ is critical with $(0,v)_{|v,u|}$
3: **while** not optimal **do**
4: $critListX = \{(0,j)_{|j,i|},(0,v)_{|v,u|}\}$, $critListY = \{(i,n^\star)_{|j,i|},(s,n^\star)_{|t,s|}\}$
5: $x_{|t,s|} = x_{|t,s|} + \varepsilon$, $y_{|t,s|} = y_{|t,s|} - \varepsilon$
6: $k_{(i,n^\star)} = 0$
7: $curArcY$ = first element in $critListY$
8: **for** all arcs $(0,t)_{|t,s|}$ that are critical with $curArcY$ **do**
9: **if** $(s,n^\star)_{|t,s|}$ is critical with an arc from $critListX$ **then**
10: $\varepsilon = 0$
11: **return**
12: **end if**
13: **if** $(s,n^\star)_{|t,s|}$ is critical with an arc $(0,c)$ **then**
14: `correctLengthsA`($\mid t,s \mid$)
15: **end if**
16: $k_{(s,n^\star)} = k_{curArcY} + 1$
17: $x_{|t,s|} = x_{|t,s|} - k_{(s,n^\star)} \cdot \varepsilon$
18: $y_{|t,s|} = y_{|t,s|} + k_{(s,n^\star)} \cdot \varepsilon$
19: ADD $(0,t)_{|t,s|}$ to $critListX$
20: ADD $(s,n^\star)_{|t,s|}$ to $critListY$
21: $curArcY$ = next element in $critListY$
22: **end for**
23: $x_{|v,u|} = x_{|v,u|} - \varepsilon$, $y_{|v,u|} = y_{|v,u|} + \varepsilon$
24: $k_{(0,j)} = 0$
25: $curArcX$ = first element in $critListX$
26: *repeat lines* 5 - 25 *symmetrically for* $(i,n^\star)_{|j,i|}$ *and* $(0,v)_{|v,u|}$
27: **for** all arcs $(0,t)_{|t,s|}$ in $critListX$ and $(s,n^\star)_{|t,s|}$ in $critListY$ **do**
28: calculate maximal ε s.t.
29: $x_{|t,s|}(\varepsilon) + \widehat{CP}_{t,n^\star}(\varepsilon) \leq \widehat{CP}^{old}_{0,n^\star} - \varepsilon$
30: $\widehat{CP}_{0,s}(\varepsilon) + y_{|t,s|}(\varepsilon) \leq \widehat{CP}^{old}_{0,n^\star} - \varepsilon$
31: **end for**
32: UPDATE arc lengths
33: **end while**
34: **end procedure**

Glossary

α ..... cycle time, page 38

$\mathcal{M}_j$ ..... Set of machines that J_j has to be processed on, page 179

A ..... selection of directed disjunctive arcs, page 15

A_{CP} ..... arc set of critical arcs, page 106

A_P ..... arc set of arcs belonging to a pair

S_i^* ..... Starting time of operation i in a specific cycle, page 79

TL_J ..... Tabu list storing jobs, page 199

TL_R ..... Tabu list storing pairs of transport moves, page 199

β_{ij} ..... binary variable defining if operation i is processed before j or not, page 141

$b(i)$ ..... $suc(i)$ if i is blocking and i else, page 49

$\widehat{CP}_{ji}$ ..... length of the longest paths from j to i in $\hat{G}_\nu$, page 109

C_i ..... completion time of operation i, page 16

$C_{\max}(S)$ ..... makespan of schedule S, page 16

$CP_{i,j}$ ..... length of longest critical path from node i to node j, page 101

D ..... set of disjunctive arcs representing machine constraints, page 15

$d(\mu)$ ..... lenght of circuit μ in graph G, page 87

$d_{i,j}$ ..... time lag between operation i and j, page 19

$(E \cup A)'$ ..... set of arc associated with arc pairs in P, page 103

A. PSEUDO-CODE FOR ε-CORRECTIONS

$\widehat{E \cup A}$ ..... Arc set of $\hat{G}$, page 104

E ..... set of arcs representing precedence constraints, page 14

E'_ν ..... set of arcs corresponding to P_ν, page 109

e_{ij} ..... empty moving time from machine $M(i)$ to machine $M(j)$, page 21

φ_j ..... flow time of job J_j, page 39

γ_i ..... binary variable defining if operation i is overlapping, page 140

$\hat{G}_\nu$ ..... relaxed graph based on pairs in P_ν, page 108

G ..... a graph, page 14

$h^*_{\star,0}$ ..... actual height of a problem in a feasible solution, page 72

$h^*_{J_j}$ ..... actual height of job J_j in a feasible solution, page 72

$h^*_{M_k}$ ..... actual height of machine M_k in a feasible solution, page 72

$h(\mu)$ ..... height of curcuit μ in graph G, page 87

h_{ij} ..... height of a constraint between i and j, page 40

$\langle i; r \rangle$ ..... r-th repetition of operation i, page 37

$i^\star$ ..... last operation of a job, page 44

$J(i)$ ..... the job in $J_1, \ldots, J_N$ operation i belongs to, page 13

J_j ..... j-th job, page 13

μ ..... circuit in a graph, page 87

m ..... number of machines, page 13

$M(i)$ ..... the machine operation i has to be processed on, page 13

M_k ..... the k-th machine, page 13

N ..... number of jobs, page 13

n ..... number of all non-dummy operations, page 13

n_j ..... number of operations job J_j has, page 13

Ω ..... set of all non-dummy operations, page 13

ω ..... overall waiting time of a partial robotic cycle, page 173

$\Omega^\star$ ..... $\Omega \cup \{\star^1, \ldots, \star^N\}$, page 21

o_k ..... number of overlapping operation of job J_k in a feasible solution of the CJSPTB, page 73

O_{ij} ..... i-th operation of job J_j, page 13

$pre(i)$ ..... direct predecessor of operation i, page 13

$pre^M(i)$ ..... predeceasing operation of i on $M(i)$, page 79

$pre^R(i)$ ..... operation transported directly before i by the robot, page 79

P ..... set of arc pairs, page 103

p_i ..... processing time of operation i, page 13

$p_i^{\max}$ ..... maximum processing time of operation i, page 18

$p_i^{\min}$ ..... minimal processing time of operation i, page 18

P_ν ..... subset of P, page 108

$p_{b(i)}$ ..... 0 if i is blocking and p_i else, page 50

R ..... robot route or cycle depending on the problem, page 22

r ..... repetition number, page 37

$suc(i)$ ..... direct successor of operation i, page 13

$suc^M(i)$ ..... succeeding operation of i on $M(i)$, page 79

$suc^R(i)$ ..... operation transported directly after i by the robot, page 79

$\star^j$ ..... dummy end activity of job J_j, page 21

S ..... a schedule, page 15

S_i ..... starting time of operation i, page 13

$S_i(r)$ ..... start processing of the r-th repetition of operation i, page 38

τ_i ..... transport move of operation i, page 21

$\tau_i(r)$ ..... r-th repetition of transport move τ_i, page 52

$\tau_i \prec \tau_j$ ..... τ_i precedes τ_j in R, page 79

θ_{ij} ..... binary variable defining if i is transported after j or not, page 141

T_i ..... start of transport move τ_i, page 23

t_i ..... time, the transport move τ_i takes, page 21

$T_i(r)$ ..... start of the r-th repetition of transport move τ_i, page 53

V ..... set of nodes, page 14

$v(\mu)$ ..... value of a circuit μ in graph G, page 87

References

E. Aarts, P.J.M. van Laarhoven, J. Lenstra, R. Vaessens, and N.L.J. Ulder. A computational study of local search algorithms for job shop scheduling. *OSRA Journal on Computing*, 6:118–125, 1994.

V.S. Aizenshtat. Multi-operator cyclic processes. *Doklady of the Byelorussian, Academy of Sciences*, 7 (4):224–227, 1963. (in Russian).

David Alcaide, Chengbin Chu, Vladimir Kats, Eugene Levner, and Gerard Sierksma. Cyclic multiple-robot scheduling with time-window constraints using a critical path approach. *European Journal of Operational Research*, 127(1):147–162, 2007.

D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3:149–156, 1991.

Christian Artigues, Sophie Demassey, and Emmanuel Néron, editors. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, chapter Resource constrained modulo scheduling, pages 267–279. ISTE, London, UK, 2010.

Francois Louis Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and Linearity: An Algebra for Discrete Event Systems.* John Wiley & Sons, 1992.

E. Balas and A. Vazacopoulos. Guided local search with shifting bottleneck for job-shop scheduling. *Management Science*, 1998.

Richard Bellman. On a routing problem. *Quarterly Applied Mathematics*, 16:87–90, 1958.

U. Bilge and G. Ulusoy. A time window approach to simultaneous scheduling of machines and material handling system in an FMS. *Operations Research*, 43:1058–1070, 1995.

REFERENCES

J. Blazewicz, W. Domschke, and E. Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93:1–33, 1996.

P. Brucker and T. Kampmeyer. Tabu Search Algorithms for Cyclic Machine Scheduling Problems. *Journal of Scheduling*, 8:303–322, 2005.

P. Brucker and T. Kampmeyer. A general model for cyclic machine scheduling problems. *Discrete Applied Mathematics*, 156:2561–2572, 2008a.

P. Brucker and T. Kampmeyer. Cyclic job shop scheduling problems with blocking. *Annals of Operations Research*, 159:161–181, 2008b.

P. Brucker, B. Jurisch, and B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49:107–127, 1994.

P. Brucker, S. Heitmann, J. Hurink, and T. Nieberg. Job-shop scheduling with limited capacity buffers. *OR Spectrum*, 28:151–176, 2006.

Peter Brucker. Scheduling and constraint propagation. *Discrete Applied Mathematics*, 123(13):227 – 256, 2002.

Peter Brucker and Sigrid Knust. Lower Bounds for Scheduling a Single Robot in a Job-Shop Environment. *Annals of Operations Research*, 115:147–172, 2002.

Peter Brucker and Sigrid Knust. *Complex Scheduling*. Springer, 2005.

Peter Brucker, Edmund K Burke, and Sven Groenemeyer. Problem generator for job-shop problem with transportation, 2009. URL `http://www.cs.nott.ac.uk/~svg/problemgenerator.html`.

E.K. Burke and G. Kendall. *Search methodologies: introductory tutorials in optimization and decision support techniques*. Springer, 2005. ISBN 9780387234601.

K. Caggiano and P. Jackson. Cyclic scheduling with acyclic job precedence constraints: improvement heuristics. Technical Report TR001348, Cornell University Operations Research and Industrial Engineering, 2002.

K. E. Caggiano and P.L.. Jackson. Finding minimum flow time cyclic schedules for non-identical, multistage jobs. *IIE Transactions*, 40(1):45 – 65, 2008. ISSN 0740817X.

P.-Y. Calland, A. Darte, and Y. Robert. Circuit retiming applied to decomposed software pipelining. *Parallel and Distributed Systems, IEEE Transactions on*, 9(1):24 –35, jan 1998. ISSN 1045-9219.

J. Carlier and E. Pinson. An Algorithm for Solving the Job-Shop Problem. *Management Science*, 35 (2):164 - 176, 1989.

Jacques Carlier and Eric Pinson. Jackson's pseudo-preemptive schedule and cumulative scheduling problems. *Discrete Applied Mathematics*, 145:80–94, 2004.

F. Chauvet, J.W. Herrmann, and J.-M. Proth. Optimization of cyclic production systems: a heuristic approach. *Robotics and Automation, IEEE Transactions on*, 19(1):150 - 154, 2003. ISSN 1042-296X.

A. Che and C. Chu. A polynomial algorithm for no-wait cyclic hoist scheduling in an extended electroplating line. *Operations Research Letters*, 33:274–284, 2005a.

A. Che and C. Chu. Multidegree cyclic scheduling of two robots in a no-wait flowshop. *IEEE Transactions on Automation Science and Engineering*, 2(2):173–183, 2005b.

Ada Che, Chengbin Chu, and Eugene Levner. A polynomial algorithm for 2-degree cyclic robot scheduling. *European Journal of Operational Research*, 127(1):31–44, February 2003.

P. Chrétienne. Transient and limiting behavior of timed event graphs. *RAIRO-TSI*, 4:127142, 1985.

C. Chu. A faster polynomial algorithm for 2-cyclic robotic scheduling. *Journal of Scheduling*, 9:453–468, 2006.

G. Cohen, P. Moller, J.-P. Quadrat, and M. Viot. Algebraic tools for the performance evaluation of discrete event systems. *Proceedings of the IEEE*, 77(1):39–85, January 1989. ISSN 0018-9219.

Y. Crama, V. Kats, J. van de Klundert, and E. Levner. Cyclic scheduling in robotic flow-shops. *Annals of Operations Research*, 96:97–124, 2000.

V.S. Aizenshtat D.A. Suprunenko and A.S. Metelsky. A multistage technological process. *Doklady Academy Nauk BSSR*, 6(9):541–522, 1962. in Russian.

G. B. Dantzig, W. Blattner, and M. R. Rao. Finding a cycle in a graph with minimum cost to times ratio with application to a ship routing problem. *Journal of Graph Theory*, 1967.

Ali Dasdan, Sandra S. Irani, and Rajesh K. Gupta. An experimental study of minimum mean cycle algorithms. Technical report, 1998.

Geismar H.N. Sethi S.P. Sriskandarajah C. Dawande, M.W. *Throughput Optimization in Robotic Cells*, volume 101. Springer, 2007.

REFERENCES

M. Dawande, H.N. Geismar, and C Sriskandarajah. Approximation algorithms for k-unit cyclic solutions in robotic cells. *European Journal of Operational Research*, 162:291–309, 2005a.

N. Dawande, H.N. Geismar, S.P. Sethi, and C. Sriskandarajah. Sequencing and Scheduling in Robotic Cells: Recent Developments. *Journal of Scheduling*, 8(5):387–426, 2005b.

Joaquim P. Marques de Sá. *Applied statistics using SPSS, STATISTICA, MATLAB and R.* Springer, 2007.

Alexandre E. Eichenberger and Edward S. Davidson. Efficient formulation for optimal modulo schedulers. In *In Proc. of the SIGPLAN 97 Conference on Programming Language Design and Implementation*, pages 194–205, 1997.

A. El-Bouria, N. Azizia, and S. Zolfaghari. A comparative study of a new heuristic based on adaptive memory programming and simulated annealing: The case of job shop scheduling. *European Journal of Operational Research*, 177:1894–1910, 2007.

Uriel Feige and Christian Scheideler. Improved bounds for acyclic job shop scheduling. In *IN PROCEEDINGS OF THE 30TH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING*, pages 624–633. ACM Press, 1998.

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

F. Gasperoni and Uwe Schwiegelshohn. Generating close to optimum loop schedules on parallel processors. *Parallel Processing Letters*, 4(4):391–404, December 1994.

F Glover. *Tabu search and adaptive memory programming - Advances, applications and challenges.* Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, MA 02061, 1997.

J. Grabowski, E. Nowicki, and S. Zdrzalka. A block approach for single-machine scheduling with release dates and due dates. *European Journal of Operational Research*, 26:278–285, 1986.

Heinz Gröflin and Andreas Klinkert. A new neighborhood and tabu search for the blocking job shop. *Discrete Applied Mathematics*, 157(17):3643 – 3655, 2009. ISSN 0166-218X. Sixth International Conference on Graphs and Optimization 2007.

Inc. Gurobi Optimization. *Gurobi Optimizer Reference Manual Version 4.5.* Houston, Texas, July 2011.

L. A. Hall. *Approximation Algorithms for NP-Hard Problems*, chapter Approximation algorithms for scheduling, pages 1–45. PWS Publishing Company, Boston, 1997.

N.G. Hall. *Handbook of Industrial Robotics*, chapter Operations research techniques for robotic system planning, design, control and analysis, pages 543–577. John Wiley, 1999.

N.G. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 45:510–525, 1997.

N.G. Hall, Hichem Kamoun, and Chelliah Sriskandarajah. Scheduling in robotic cells: complexity and steady state analysis. *European Journal of Operational Research*, 109(1):43–65, 1998.

N.G. Hall, T.E. Lee, and M.E. Posner. The complexity of cyclic shop scheduling problems. *Journal of Scheduling*, 5:307–327, 2002.

C. Hanen. Study of a NP-hard cyclic scheduling problem: the recurrent job-shop. *European Journal of Operational Research*, 72(1):82–101, 1994.

C. Hanen and A. Munier. *Scheduling Theory and its Applications*, chapter Cyclic scheduling on parallel processors: An overview. John Wiley & Sons, 1995.

S. Heitmann. *Job-shop scheduling with limited buffer capacities*. PhD thesis, University of Osnabrueck, Germany, 2007.

Ronald A. Howard. *Dynamic programming and markov processes*. MIT˜Press, Cambridge, Massachusetts, 1960.

J. Hurink and Sigrid Knust. Makespan minimization for flow-shop problems with transportation times and a single robot. *Discrete Applied Mathematics*, 112:199–216, 2001.

J. Hurink and Sigrid Knust. Tabu search algorithms for job-shop problems with a single transport robot. *European Journal of Operational Research*, 162:99–111, 2005.

Johann Hurink, Bernd Jurisch, and Monika Thole. Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spectrum*, 15:205–215, 1994.

IBM ILOG. *User's Manual for CPLEX V12.1*, 2010. URL `ftp://ftp.software.ibm.com/software/websphere/ilog/docs/optimization/cplex/ps_usrmancplex.pdf`.

I Ioachim and F Soumis. Schedule efficiency in a robotic production cell. *International Journal of Flexible Manufacturing Systems*, 7(1):5–26, MAR 1995. ISSN 0920-6299.

REFERENCES

J. Irwin and L. Wilkerson. An improved algorithm for scheduling independent tasks. *AIIE Transactions*, 3:239–245, 1971.

A.S. Jain and S. Meeran. Deterministic job-shop scheduling: past, present and future. *European Journal of Operational Research*, 113:390–434, 1999.

Klaus Jansen, Idsia Lugano, Roberto Solis-oba, Mpii Saarbrucken, and Maxim Sviridenko. Makespan minimization in job shops: A polynomial time approximation scheme. In *Proceedings of the 31th Annual ACM Symposium on Theory of Computing*, pages 394–399, 2001.

Albert Jones and Luis C. Rabelo. Survey of job shop scheduling techniques. Technical report, National Institute of Standards and Technology, 1998.

H. Kamoun and C. Sriskandarajah. The complexity of scheduling jobs in repetitive manufacturing systems. *European Journal of Operational Research*, 70:350–364, 1993.

T. Kampmeyer. *Cyclic Scheduling Problems.* PhD thesis, University of Osnabrueck, Germany, 2006.

V. Kats and E. Levner. A parametric critical path problem and an application for cyclic scheduling. *Discrete Applied Mathematics*, 87:149–158, 1998a.

V. Kats and E. Levner. Cyclic ccheduling of operations for a part type in an FMS handled by a single robot: a parametric critical-path approach. *International Journal of Flexible Manufacturing Systems*, 10:129–138, 1998b.

V. Kats and E. Levner. An efficient algorithm for multi-hoist cyclic scheduling with fixed processing times. *Journal of Scheduling*, 5:23–41, 2002.

V. Kats, E. Levner, and V.E. Levit. An improved algorithm for cyclic flowshop scheduling in a robotic cell. *European Journal of Operational Research*, 97:500–508, 1997.

V. Kats, E. Levner, and L. Meyzin. Multiple-part cyclic hoist scheduling using a sieve method. *IEEE Transactions on Robotics and Automation*, 15:704–713, 1999.

Vladimir Kats and Eugene Levner. A faster algorithm for 2-cyclic robotic scheduling with a fixed robot route and interval processing times. *European Journal of Operational Research*, 209(1):51–56, FEB 16 2011. ISSN 0377-2217.

Vladimir Kats, Lei Lei, and Eugene Levner. Minimizing the cycle time of multiple-product processing networks with a fixed operation sequence, setups, and time-window constraints. *European Journal of Operational Research*, 127(3):1196–1211, 2008.

H. Kise. On an automated 2-machine flowshop scheduling problem with infinite buffer. *Journal of the Operations Research Society of Japan*, 34:354–361, 1991.

Sigrid Knust. *Shop-Scheduling Problems with Transportation.* PhD thesis, Univeristy of Osnabrueck, 1999.

P. Lacomme and N. Tchernev. Resolution of a job-shop problem with a single transport robot and buffer facilities. In *International conference on service systems and service management*, volume 1 and 2, 2006.

A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, July 1960.

Chung-Yee Lee and Vitaly A. Strusevich. Two-machine shop scheduling with an uncapacitated interstage transporter. *IIE Transactions*, 37:725 – 736, 2005.

T.E. Lee. Stable earliest starting schedules for cyclic job shops: A linear system approach. *The International Journal of Flexible Manufacturing Systems*, 12:59–80, 2000.

T.E. Lee and M.E. Posner. Performance measures and schedules in periodic job shops. *Operations Research*, 45:72–91, 1997.

L. Lei. Determining the optimal starting times in a cyclic schedule with a given route. *Computers and Operations Research*, 20(8):807–816, OCT 1993. ISSN 0305-0548.

L. Lei and T.J. Wang. A proof: The cylic scheduling problem is NP-complete. Working Paper, Rutgers University, New Jersey, April 1989.

J. K. Lenstra and D. B. Shmoys. *Scheduling Theory and its Applications*, chapter Computing near-optimal schedules, pages 1–14. Wiley, Chichester, 1995.

J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 4:121–141, 1977.

E. Levner, K. Kogan, and I. Levin. Scheduling a 2-machine robotic cell - a solvable case. *Annals of Opeartions Research*, 57:217–232, 1995.

REFERENCES

Eugene Levner, Vladimir Kats, David Alcaide López de Pablo, and T. C. E. Cheng. Complexity of cyclic scheduling problems: A state-of-the-art survey. *Comput. Ind. Eng.*, 59(2):352–361, 2010.

E.M. Livshits, Z.N. Mikhailetsky, and E.V. Chervyakov. A scheduling problem in an automated flow line with an automated operator. *Computational Mathematics and Computerized Systems*, 5:151–155, 1974.

Hugues Marchand, Alexander Martin, Robert Weismantel, and Laurence Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(13):397 - 446, 2002. ISSN 0166-218X.

Paul Martin and David B. Shmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Proceedings of the 5th International Conference on Integer Programming and Combinatorial Optimization*, pages 389–403. Springer, 1996.

Alessandro Mascisa and Dario Pacciarelli. Job-shop scheduling with blocking and no-wait constraints. *European Journal of Operational Research*, 143:498–517, 2002.

Monaldo Mastrolilli and Luca Maria Gambardella. Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3:3–20, 2000.

H. Matsuo. Cyclic sequencing problems in the two-machine permutation flowshop: Complexity, worst case and average case analysis. *Naval Research Logistics*, 37:679–694, 1990.

S. T. McCormick and U. S. Rao. Some complexity results in cyclic scheduling. *Mathematical and Computer Modelling*, 20:107–122, 1994.

S.T. McCormick, Michael L. Pinedo, Scott Shenker, and Barry Wolf. Sequencing in an assembly line with blocking to minimize cycle time. *Operations Research*, 37(6):925–935, 1989.

C. Meloni, D. Pacciarelli, and M. Pranzo. A rollout metaheuristic for job shop scheduling problems. *Annals of Operations Research*, 131:215–235, 2004.

Z. Michalewicz and D.B. Fogel. *How to solve it: modern heuristics.* Springer, 2004. ISBN 9783540224945.

J.F. Muth and G.L. Thompson. *Industrial Scheduling.* Prentice-Hall, 1963.

WC Ng and J Leung. Determining the optimal move times for a given cyclic schedule of a material handling hoist. *Computers and Industrial Engineering*, 32(3):595–606, JUL 1997. ISSN 0360-8352.

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

E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91:160–175, 1996b.

E. Nowicki and C. Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8:145–159, 2005.

S. S. Panwalkar. Scheduling of a 2-machine flowshop with travel time between machines. *Journal of the Operational Research Society*, 42:609–613, 1991.

S. K. Park and K. W. Miller. Random number generators: good ones are hard to find. *Commun. ACM*, 31(10):1192–1201, 1988.

E. Pinson. *Scheduling Theory and its Applications*, chapter The job-shop scheduling problem: a concise survey and some recent developments, pages 277–293. John Wiley, 1995a.

E. Pinson. The job shop scheduling problem: a concise survey and some recent developments. In *PHOENIX Workshops*, 1995b.

C. N. Potts and V. A. Strusevich. Fifty years of scheduling: a survey of milestones. *The Journal of the Operational Research Society*, 60:41–68, 2009.

R Development Core Team. The R project for statistical computing. URL `http://www.r-project.org/`.

Yves Robert and Frédéric Vivien. *Introduction to Scheduling*. Chapman and Hall/CRC Press, 2009.

I. V. Romanovskil. Optimization of stationary control of a discrete deterministic process. *Cybernetics and Systems Analysis*, 3:52–62, 1967. ISSN 1060-0396.

R. Roundy. Cyclic schedules for job-shops with identical jobs. *Mathematics of Operations Research*, 17:842–865, 1992.

B. Roy and B. Sussman. Les problèmes dórdonnancement avec constraintes disjonctives. Technical report, SEMA, Note D.S., No. 9, Paris, 1964.

REFERENCES

Petra Schuurman and Gerhard J. Woeginger. Polynomial time approximation algorithms for machine scheduling: ten open problems. *Journal of Scheduling*, 2(5):203–213, 1999. ISSN 1099-1425.

S. S. Shapiro and M. B. Wilk. An analysis of variance test for normality (complete samples). *Biometrika*, 52:591611, 1965.

Avraham Shtub, Jonathan F. Bard, and Shlomo Globerson. *Project management: engineering, technology, and implementation.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994.

Helman I. Stern and Gad Vitner. Scheduling parts in a combined production-transportation work cell. *The Journal of the Operational Research Society*, 41:625–632, 1990.

Premysl Sucha, Zdenek Pohl, and Zdenek Hanzálek. Scheduling of iterative algorithms on FPGA with pipelined arithmetic unit. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 404–412, 2004.

V.S. Tanaev. A scheduling problem for a flowshop line with a single operator. *Engineering Physical Journal*, 7(3):111–114, 1964.

Benot Trouillet, Ouajdi Korbaa, and Jean-Claude Gentina. Formal approach of FMS cyclic scheduling. *IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews*, 37(1):126 –137, 2007. ISSN 1094-6977.

Peter J. M. van Laarhoven, Emile H. L. Aarts, and Jan Karel Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1):113 –125, 1992.

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1:80–83, 1945.

J.Y. Yen. An algorithm for finding shortest routes from all source nodes to a given destination in general network. *Quaterly of Applied Mathematics*, 27:526–530, 1970.

Chao Yong Zhang, PeiGen Lia, YunQing Raoa, and ZaiLin Guana. A very fast TS/SA algorithm for the job shop scheduling problem. *Computers and Operations Research*, 35:282–294, 2008.