Globally Optimal Schedules for Cyclic Systems with Non-Blocking Specification and Time Window Constraints

Eckart Mayer
Stuttgart

von der Fakultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuss:

Vorsitzender:	Prof. Dr.-Ing. Clemens Gühmann
Gutachter:	Prof. Dr.-Ing. Jörg Raisch
	Prof. Alessandro Giua

Tag der wissenschaftlichen Aussprache: 08. Juni 2007

Berlin 2007
D 83

Forschungsberichte aus dem Max-Planck-Institut
für Dynamik komplexer technischer Systeme

Band 18

Eckart Mayer

Globally Optimal Schedules for Cyclic Systems with Non-Blocking Specification and Time Window Constraints

D 83 (Diss. TU Berlin)

Shaker Verlag
Aachen 2007

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Zugl.: Berlin, Techn. Univ., Diss., 2007

Printed in Germany.

ISBN 978-3-8322-6587-8
ISSN 1439-4804

Shaker Verlag GmbH • P.O. BOX 101818 • D-52018 Aachen
Phone: 0049/2407/9596-0 • Telefax: 0049/2407/9596-9
Internet: www.shaker.de • e-mail: info@shaker.de

Vorwort

Als einer der ersten Mitarbeiter der Fachgruppe System- und Regelungstheorie am neu gegründeten Max-Planck-Institut für Dynamik komplexer technischer Systeme in Magdeburg konnte ich am hochinteressanten Aufbau neuer Infrastruktur, Forschung und Lehre mitwirken. Ich bedanke mich sehr herzlich bei meinem Betreuer Professor Jörg Raisch für diese Möglichkeit, für das gute Arbeitsklima, und besonders für die wertvollen Beiträge zu meinem Forschungsthema auf dem Gebiet der ereignisdiskreten und hybriden Systeme, aus dem die hier vorliegende Dissertation hervorgegangen ist.

Mein Dank geht an Professor Alessandro Giua, Università di Cagliari, für die Übernahme des Mitberichts, und an die Mitarbeiter der Fakultät IV der Technischen Universität Berlin, insbesondere an Herrn Professor Clemens Gühmann für die Leitung der Prüfungskommission.

Vielen Dank an alle Weggefährten in Magdeburg und Berlin, besonders an meinen Freund und Kollegen Ulrich Vollmer mit Familie. Für fruchtbare Kooperation möchte ich mich bei Herrn Thomas Haenel und Herrn Bernd Prause von der Firma CyBio AG in Jena, bei Utz-Uwe Haus und Professor Robert Weismantel vom Institut für mathematische Optimierung der Otto-von-Guericke-Universität Magdeburg, und bei Danjing Li und Franziska Geyer bedanken.

Die Arbeit wurde teilweise gefördert durch das Bundesministerium für Wirtschaft und Technologie im Rahmen des Programms PRO INNO.

Und damit schließlich ein privates, besonders liebes Dankeschön für die vielfältige Unterstützung an meine Frau Sandra, an meine Eltern und meinen Bruder Steffen – und für großartigen Service unterwegs auf der Zielgeraden an meine Freunde Florian Bessler und Barbara und Harald Schiller.

Magdeburg / Berlin / Stuttgart, Juni 2007 Eckart Mayer

Contents

Zusammenfassung

Eine wichtige Eigenschaft vielfältiger Systeme in Natur und Technik ist der *zyklische Betrieb*. Bei zyklischen Systemen wiederholen sich die Ereignisse periodisch über eine große oder beliebige Anzahl von Zyklen hinweg. Der natürliche Vorteil solcher zyklischen Abläufe liegt in ihrer einfachen Struktur, im regelmäßigen Systemfortschritt und in der gleichmäßigen Verteilung der Ressourcennutzung.

Das wichtigste Merkmal eines zyklischen Systems ist dessen *Zyklus-* bzw. *Taktzeit*, also die Dauer des einzelnen Zyklus bzw. der (mittlere) Zeitversatz zwischen dem Start aufeinander folgender Zyklen. Der Kehrwert der Taktzeit liefert den *Durchsatz* des Systems. Die Zielsetzung der meisten Anwendungen ist die Maximierung des Durchsatzes, also die Bearbeitung möglichst vieler Einheiten pro Zeit.

Die vorliegende Arbeit befasst sich mit der Ablaufplanung (*Scheduling*) für zyklisch betriebene Systeme. Scheduling beschreibt die (optimale) Zuordnung von Aktivitäten bzw. Aufgaben zu limitierten Ressourcen. In dieser Arbeit werden *streng zyklische Systeme* betrachtet, bei denen eine vorgegebene Anordnung von Aktivitäten (*Batch*) vielfach wiederholt wird. Dabei folgen alle Batches demselben zeitlichen Muster, darüberhinaus besteht zwischen aufeinander folgenden Batches ein immer gleicher Zeitabstand (Taktzeit). Beim streng zyklischen Ablauf startet in jedem Zyklus immer genau ein Batch, entsprechend wird in jedem Zyklus auch genau ein Batch abgeschlossen. Während seiner Bearbeitung benötigt der einzelne Batch möglicherweise dieselbe Ressource mehrmals, d. h. er kehrt nach Aktivitäten auf anderen Ressourcen erneut zu einer Ressource zurück. Einige der Aktivitäten/Arbeitsschritte hängen vom Abschluss vorhergehender Aktivitäten des Batches ab, andere können auch parallel ausgeführt werden (Präzedenznetzwerk). Ein wichtiges Anwendungsgebiet der vorliegenden Arbeit sind so genannte *High throughput screening-Anlagen*. Dort werden Substanzen auf Mikrotitrationsplatten transportiert; eine Ressource steht erst für eine nachfolgende Aktivität zur Verfügung, wenn die Platte die Ressource verlassen hat. Da dafür umgekehrt die Nachfolgeressource frei sein muss,

Abbildung 1. Beispiel für ein Batch-Zeitschema mit sechs Aktivitäten (Gantt chart).

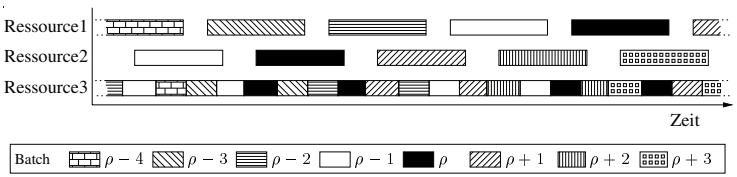

Abbildung 2. Optimaler zyklischer Ablaufplan für das Schema aus Abb. 1 (Ausschnitt).

weist das System die Eigenschaft des *Blockierens* auf (Deadlockfähigkeit). Beim High throughput screening treten darüberhinaus Unter- und Obergrenzen für bestimmte Zeitdauern im Ablauf des einzelnen Batches auf, die vom Prozess vorgegeben sind (*Zeitfenster*, *obere Zeitschranken*). Für die beschriebenen zyklischen Systeme übersteigt die Batch-Bearbeitungszeit üblicherweise die Taktzeit. Dementsprechend befinden sich zu jedem Zeitpunkt mehrere Batches gleichzeitig im System: Eine Ressource führt (frühe) Arbeitsschritte eines Batches vor (späten) Arbeitsschritten vorhergehender Batches aus.

In dieser Arbeit wird eine Methode vorgestellt, mit der zyklische Scheduling-Probleme für die beschriebene Systemklasse global optimal gelöst werden. Die Methode stützt sich auf ein dafür entworfenes *ereignisdiskretes* Systemmodell. Verfahren der *Graphentheorie* erlauben die systematische Vereinfachung des Systemmodells, ohne dass dabei die global optimale Lösung abgeschnitten wird. Auf diesem Weg kann das zyklische Scheduling-Problem als gemischt-ganzzahlige nichtlineare Optimierungsaufgabe (mixed-integer nonlinear program, MINLP) formuliert werden. Der wichtigste Schritt dabei ist die Herleitung der disjunktiven Reihenfolgebedingungen, mit denen der Tatsache Rechnung getragen wird, dass jede Ressource nur von einer Aktivität gleichzeitig belegt sein kann. Das mathematische Optimierungsproblem kann durch eine geschickte Transformation in eine *lineare* gemischt-ganzzahlige Formulierung überführt werden (mixed-integer linear program, MILP). Etablierte Verfahren der mathematischen Optimierung ermöglichen dann die Berechnung global optimaler Lösungen – für reale Aufgabenstellungen (beispielsweise aus dem High throughput screening) innerhalb weniger Sekunden.

Abbildung 1 zeigt für ein einfaches Beispiel die Aktivitäten des einzelnen Batches als Gantt Chart. Für dieses Batch-Schema liefert Abbildung 2 einen Ausschnitt aus dem ermittelten durchsatzoptimalen Ablaufplan: Das zeitliche Muster für den einzelnen Batch wurde (durch Einfügen künstlicher Wartezeiten) modifiziert; eine global optimale zyklische Reihenfolge der Aktivitäten auf den einzelnen Ressourcen wurde ermittelt.

Über die hier beschriebene Systemklasse hinaus behandelt die Arbeit mehrere Erweiterungen, unter anderem reihenfolgeabhängige Maschinen-Umrüstzeiten, Ressourcen mit einer Kapazität größer eins und hierarchisch überlagerte Periodizität. Die abgedeckte Systemklasse ist damit sehr allgemeingültig und umfasst den größten Teil der in der Praxis auftretenden Scheduling-Probleme für zyklische Systeme. Die Anwendung von Methoden der ereignisdiskreten Systemtheorie (*Graphentheorie*, *Max-Plus-Algebra*) erlaubt eine kompakte Formulierung des mathematischen Optimierungsproblems, mit der auch große Instanzen in wenigen Sekunden gelöst werden können. Die Anwendbarkeit der Methoden wird durch Beispiele aus unterschiedlichen Bereichen gezeigt. Die betrachteten Anwendungen umfassen neben realen Aufgabenstellungen des High throughput screening weitere Beispiele aus der Fertigungsplanung und der Verkehrstechnik.

Chapter 1

Introduction

1.1 Cyclic Systems

Cyclically operated systems are characterized by the repetition of events in a *periodic scheme*. A set of activities is repeated a large or unlimited number of times (cycles). In engineering technology, application areas for cyclic operation include production systems (flexible manufacturing, FMS), transportation systems (timetables, traffic lights), communication systems, or chemical batch processing. Numerous examples for periodic phenomena can be found beyond technical systems, e. g. in nature (cell cycles) or music (canon). Natural advantages of cyclic operation in technical processes are structural simplicity, steady progress and uniformly distributed utilization of resources.

The most important characteristic of a cyclic system is the *cycle time*, i. e. the (mean) time distance between the occurrence of the same events in consecutive cycles. The inverse of the cycle time is equivalent to the *throughput* of the system. For most applications, the aim is to maximize throughput, i. e. to handle a maximum number of entities per time. In *discrete event systems* (DES) theory, *max-plus algebra* has been established as a convenient tool to simulate and analyze cyclic systems [9]. The eigenvalue of a system matrix in max-plus algebra determines the minimum possible cycle time of the system. However, in its basic form, max-plus algebra only covers systems that can be modeled by use of event graphs (also known as synchronization graphs), which represent systems that do not exhibit open sequence decisions. If the periodically repeated activities compete for resources that can only be used by one activity at a time, the order of the activities within the cycle is of essential influence on the cycle time and therefore has to be set in an optimal way.

The problem of determining sequences and timing for activities on limited resources is a *scheduling* problem. In this thesis, the idea of solving a scheduling problem is to find *proven globally optimal solutions* using a system theoretic approach (rather than searching for any 'good', but possibly suboptimal solution). Due to their inherent complexity, scheduling problems are often difficult to solve efficiently. Therefore, suitable mathematical models are needed in order to obtain optimal solutions for large systems within reasonable calculation time. One of the main tasks of current research on DES theory is to manage the exponential state space explosion of complex systems by improved architectures for structured discrete event models [14]. In this thesis, we present a DES modeling approach for cyclic systems in which activities compete for common resources. Based on this modeling approach, we derive a method to efficiently solve a broad class of cyclic scheduling problems.

Cyclic operation can either be enforced by the process requirements or can be deliberately chosen to benefit from the advantages of periodic behavior. A cyclic schedule, among the whole set of feasible schedules for a plant or a process, does not necessarily come with the highest possible throughput. The set of acyclic schedules offers more degrees of freedom and therefore often includes a schedule with better throughput (actually, a cyclic schedule may be seen as a special case of an acyclic schedule). However, cyclic schedules have the advantage of structural simplicity. Whereas the degrees of freedom describing all acyclic schedules for a problem can often not be handled in a systematic way any more, the set of feasible cyclic schedules will often still be manageable, and a globally optimal schedule with respect to this deliberately imposed prerequisite can be determined. The examples in this thesis show that the globally optimal cyclic schedule often outperforms the best schedule found manually or by heuristics, searching within the set of all, cyclic and acyclic schedules.

The approach developed in this thesis covers a broad class of cyclically operated systems, especially those that exhibit some or all of the following requirements:

- *revisited resources:* while an entity is processed in the system, it may visit the same resource not only once (*flow shop*) but several times.
- *non-blocking specification:* after a workstep on one resource is finished, the resource will not be released before the resource for the next workstep is allocated. In scheduling, this behavior is referred to as *blocking*. Any feasible cyclic schedule has to obey a specification that excludes the occurrence of *deadlocks*, which may otherwise arise from this property.

- *time window constraints:* timing constraints may be imposed on the time scheme that has to be followed by the entities while they are processed in the system. These constraints may include lower as well as upper bounds on time intervals.

- In practice, processing of an entity may often not simply consist of a strictly ordered set of resource allocations (*job shop*), but may require parallel branches of worksteps with split-up points, synchronization points and transfer processes. Such a process cannot be described just by monolithic activities on resources. Instead, a DES graph model is needed, where the process is defined by a *set of events*, together with their event times and dependencies.

- Furthermore, several extensions allow the method to deal with generalized problem structure, such as systems that include resources with multiple capacity or sequence dependent setup times.

This thesis is arranged as follows: in this chapter, we give a literature review on past and current research activities on scheduling for cyclic systems. Chapter 2 then provides the basic mathematical background for the thesis. *Graph theory* and *max-plus algebra* are briefly reviewed; an introduction to *mathematical programming* is given from the engineering point of view. Subsequently, we give an exact definition of the cyclic scheduling problem in Chapter 3. We also define the special requirements and discuss possible objective functions.

In Chapter 4 we present a mathematical modeling framework for cyclically operated timed discrete event systems. Based on this modeling approach, the cyclic scheduling problem is solved in Chapter 5. After deriving a mathematical program for the cyclic scheduling problem, the method is illustrated by a simple example. Additionally imposed bounds and constraints allow for an accelerated computation of a globally optimal solution. Chapter 6 extends the modeling framework as well as the scheduling method towards a generalized class of problems, including cyclic processes with a finite number of cycles or a hierarchical cyclic scheme, resources with multiple capacity or sequence dependent setup times.

The results presented in this thesis have originally been motivated by the cyclic scheduling problem for so-called *high throughput screening* systems. The method has been used to solve many real-world problems from the high throughput screening industry with very good results [73, 74, 75]. Cyclic schedules are as well advantageous for flexible manufacturing systems (FMS): there, a small set of different products is often manufactured a large number of times. Especially if setup times for the machines are negligible, cyclic

schedules can be applied and are superior to conventional schedules [67]. In Chapter 7, the method is applied to a large number of problem instances from these areas. In addition, the application of cyclic scheduling to some railway transportation problems is discussed by considering a small example. The thesis finishes with a summary and some ideas on further research in the field of cyclic scheduling.

1.2 Literature review

Scheduling, in general, is the assignment of jobs to limited resources. Given a set of activities and resources, decision has to be taken for each activity, on which resource and at what time it should take place. For a detailed classification of scheduling problems and application examples, the reader is referred to the corresponding text books [17, 94, 96].

The *cyclic scheduling problem* is about determination of schedules with periodic behavior, i. e. under cyclic operation. A large number of cyclic scheduling studies have been published in literature during the last years, covering a broad variety of different problem types. We will first consider cyclic systems *without blocking possibilities or time window constraints*. In the *cycle shop* problem, the work that has to be done repetitively is divided into jobs that consist of a fixed sequence of activities (See Middendorf and Timkovsky [79] for a classification of cycle shops that covers a wide area of cyclic scheduling problem types). The cyclic job shop problem as a subclass of cycle shops is analyzed, e. g., by Roundy [101], Ohl et al. [90, 91], Camus et al. [19], Korbaa et al. [63, 64], Aldakhilallah and Ramesh [4] as well as by Hall et al. [46] and Seo and Lee [104]. Beneath the cycle shop type, a more general type of cyclic scheduling problems results if dependencies of activities within jobs are represented by a *precedence network* rather than just a fixed sequence. Discussion of such systems include the work by Hanen [49] and Huh et al. [55] as well as Trouillet et al. [110] and Chretienne [23]. Hanen and Munier [50] give an overview of cyclic scheduling problems with identical parallel machines, which arise for parallel processors (see also [22, 24, 51, 83, 84]). Di Febbraro et al. [35] propose a solution method for a fairly general model for cyclic manufacturing systems with the existence of limited buffers and machine alternatives.

A second class of cyclic scheduling problems arises if the system exhibits the possibility of *blocking*. Blocking occurs if, due to the absence of buffers, a job remains on a resource until the next resource is available, therefore blocking the first resource even after completion of the respective workstep. Problems with blocking occur in flow shops

[1, 59, 65] as well as for most robotic work-cells. Scheduling for robotic work-cells, with the sequence of machines being a flow shop, has been studied by Kats, Levner, Crama et al. [26, 27, 62, 68], Chen et al. [21] as well as Ng and Leung [87] for one robot and by Kats and Levner [60] and Ioachim et al. [57] for multiple robots. Also, the robotic cycle shop, called *re-entrant flow shop*, is studied by Kats and Levner [61]. One should note that many of the robotic work-cell problems imply *no-wait constraints* ([60, 62, 68]) or fixed sequences of robot moves ([87]) which means that they can be solved by algorithms of polynomial order.

Other specific cyclic scheduling problems include the assumption of a prescribed cycle time [89, 105], flexibilities in the set of jobs per batch (*minimal part set*, MPS) [95], or in the assignment of resources to activities [20]. In general, the focus of cyclic scheduling is different if the sequences of activities on the resources are fixed [66, 67].

Further extensions include stochastic processing times [102], separation scheduling problems [8] or pipeline operations [3]. Boudoukh et al. [15] analyze the cyclic job shop with different processing times for consecutive jobs, Frey [38] studies sequencing problems in a flow line where processing times also vary between consecutive jobs, Giua et al. [41] solve the problem of finding the minimal number and optimal allocation of transportation resources in a job shop, and Chudy [25] analyzes the basic case of two cyclic processes sharing a common resource.

Application areas of cyclic scheduling include flexible manufacturing (FMS) and robotic systems [27, 104], traffic and transportation [69, 89, 105], chemical batch plants [5, 80, 81, 97, 103, 106], and high throughput screening [31]. Scheduling problems from high throughput screening are closely related to those originating from flexible manufacturing systems. Nevertheless, as there are specific requirements (e. g. timing conditions for chemical reactions), the cyclic scheduling problem for high throughput screening incorporates constraints that differ structurally from the abovementioned problems.

The solution methods proposed for cyclic scheduling problems are manifold: for some problem classes, *polynomial algorithms*, i. e. algorithms whose calculation time depends 'only' in polynomial order on problem size, have been found (e. g. [3, 26, 46, 57, 60, 61, 62, 68]). Problems that are known to be NP-hard, i. e. for which polynomial solution algorithms do not exist, may nevertheless be efficiently solved by formulating them as (mixed integer) *optimization problems*. Acceptable calculation times for globally optimal solutions can then be achieved by use of problem-specific formulations and branch-and-bound techniques (e. g. [21, 35, 55, 59, 101, 103, 104]). Besides, problem specific

heuristics have been found for a large number of problems. Heuristics may either be used to do systematic search for globally optimal solutions [63, 87] or to find reasonably good solutions in relatively short calculation time [1, 4, 51, 90, 102]. The mathematical modeling procedure for cyclic discrete event systems typically makes use of systematic graph theory based methods [22, 35, 41, 110] or methods from discrete event systems theory like max-plus algebra [34, 40, 66, 88, 104] or Petri nets [19, 20, 38, 90, 91].

From the study of research work on cyclic scheduling, it becomes clear that even for very closely related types of scheduling problems, very different approaches have to be chosen. For NP-hard problems, the suitability of an algorithm may even depend simply on problem size: if problem size is small to medium, exact algorithms based on mixed integer programming can provide a proven optimal solution and intermediate solutions with guaranteed bounds on the distance to the potential global optimum. However, the same algorithms may fail if problem size becomes large.

Whereas there are a large number of contributions on cyclic flow shops and job shop problems, some also including blocking (robotic cells [27], [1]) or time window constraints [21, 27], up to now, there are no studies on the cyclic scheduling problem with general precedence network structure together with blocking and time window constraints. Nevertheless, this very general class of cyclic scheduling problems reflects many real-world problems, especially robotic systems with chemical processes (high throughput screening) and will therefore be discussed in this thesis.

Chapter 2

Mathematical background

2.1 Max-plus algebra and graph theory

The mathematical model for the scheduling problem used in this thesis is based on a timed *graph model*. *Max-plus algebra* as a very powerful tool for the simulation and analysis of cyclic discrete event systems allows for a compact representation of weighted graphs. This section briefly reviews basic terms of graph theory and max-plus algebra, limited to the topics needed to understand the graph models used in this thesis. For further details on graph theory, the reader is referred to respective text books, e. g. [44]. An extensive description of max-plus algebra and its properties can be found in [9].

Max-plus algebra

Definition 1 (Max-plus algebra) *Max-plus algebra [9, 29] consists of two operations* $\oplus$ *and* $\otimes$ *on the set* $\mathbb{R}_{max} = \mathbb{R} \cup \{-\infty\}$.[1] *The operations are defined as follows:* $\forall a, b \in \mathbb{R}_{max}$ *:*

$$\begin{aligned} a \oplus b &= \max(a, b)\,, \\ a \otimes b &= a + b\,, \\ a \otimes -\infty &= -\infty\,, \end{aligned}$$

[1] Alternatively, $\mathbb{R}_{max}$ can be extended to $\mathbb{R}_{max} = \mathbb{R} \cup \{-\infty, +\infty\}$.

The operation $\otimes$ is called *multiplication* of max-plus algebra, the operation $\oplus$ is called *addition* of max-plus algebra.

For multiplication $\otimes$ and addition $\oplus$, some standard properties hold similar to conventional algebra (associativity, distributivity of $\otimes$ over $\oplus$). The neutral element of max-plus multiplication $\otimes$ is 0, also denoted as e, the multiplicative inverse of $a \in \mathbb{R}_{max}$ is $-a$. The neutral element of addition $\oplus$ is $-\infty$, also denoted as ϵ. There exists no inverse with respect to max-plus addition. Max-plus algebra is therefore a dioid, but not a ring.

Max-plus operations can be defined for matrices $\in \mathbb{R}_{max}^{p\times q}$ as follows: the *matrix sum* $C = A \oplus B$ with $A, B, C \in \mathbb{R}_{max}^{p\times q}$ is defined by

$$c_{ij} = a_{ij} \oplus b_{ij} = \max(a_{ij}, b_{ij}) \ .$$

Max-plus *matrix multiplication* $C = A \otimes B$ with $A \in \mathbb{R}_{max}^{p\times r}$, $B \in \mathbb{R}_{max}^{r\times q}$, $C \in \mathbb{R}_{max}^{p\times q}$, is defined by

$$c_{ij} = \bigoplus_{k=1}^{r} (a_{ik} \otimes b_{kj}) = \max_{k}(a_{ik} + b_{kj}) \ .$$

The *neutral element of matrix addition* is the zero matrix N with $n_{ij} = -\infty$. The *neutral element of matrix multiplication* is the identity matrix I with

$$i_{ij} = \begin{cases} 0 & , i = j \\ -\infty & , i \neq j \ . \end{cases}$$

Based on multiplication $\otimes$, the *powers* of a square matrix A can be defined as

$$A^k = \underbrace{A \otimes A \otimes \cdots \otimes A}_{\text{k times}} \ .$$

The *closure* A^* of a square matrix $A \in \mathbb{R}_{max}^{n\times n}$ is the infinite sum of its powers:

$$A^* = \bigoplus_{k=0}^{\infty} A^k \ . \tag{2.1}$$

In [9] it is shown that, if this sum converges, it is equal to

$$A^* = I \otimes A \otimes A^2 \otimes \ldots \otimes A^{n-1} \ .$$

Similar to conventional algebra, an *eigenvalue* λ and eigenvectors v can be assigned to a square matrix $A \in \mathbb{R}_{max}^{n\times n}$ by solving the eigenvalue equation

$$A \otimes v = \lambda \otimes v \ .$$

Max-plus algebra can be used to build compact models for periodic systems (e. g. very descriptive models for railway systems [16, 69, 72, 108]). Such models are linear with respect to the algebra. However, in its basic form, max-plus algebra can only handle systems that are conflict-free, i. e. no open sequence decisions can be included in the model. Since the decision about optimal sequences is the main topic of scheduling, max-plus algebra is not suitable for all steps along the way to the scheduling problem solution. In this theses, max-plus algebra is used to simplify notation and to investigate the underlying graph model.

Graph theory

In this section, we provide the basics of graph theory that are needed to build up and simplify mathematical models for the cyclic scheduling problem.

Definition 2 (Graph) *A graph $\mathcal{G}$ consists of a pair $(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes a set of nodes ('vertices') and $\mathcal{E}$ denotes a set of edges. The set $\mathcal{E}$ consists of pairs of nodes. An edge may connect a node to itself (*self-loop*). A pair of nodes may be connected by more than one edge.*

Definition 3 (Directed graph) *A directed graph is a graph where the two nodes in the elements of $\mathcal{E}$ are sorted, i. e. the edges have a direction from a starting node (*tail*) to a target node (*head*). In a directed graph, the edges are also called* arcs*.*

Definition 4 (Simple graph) *A graph with a maximum of one edge connecting each pair of nodes $(q1, q2)$ is called simple.*[2]

Simple directed graphs are also called *digraphs*. A digraph can be illustrated in a line drawing: circles represent nodes and directed arcs represent edges. A small example with 5 nodes and 8 edges is given in Fig. 2.1. In digraphs, the ordered pair (q_1, q_2) resp. the matrix index $\cdot_{(q2,q1)}$ uniquely identifies the arc leading from node $q1$ to node $q2$.

Definition 5 (Predecessor, successor) *In a digraph, $q1$ is a* predecessor *of $q2$ iff there is an arc from node $q1$ to node $q2$. Node $q2$ is called* successor *of $q1$ iff $q1$ is a predecessor of $q2$.*

A node with a self-loop is a predecessor and a successor of itself.

[2] Note that, in a simple graph, we still allow for self-loops.

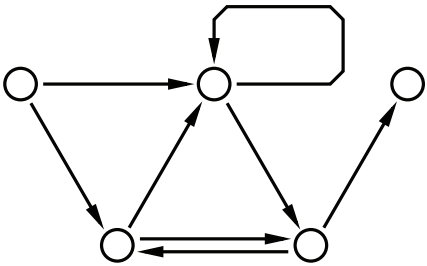

Figure 2.1. A simple digraph.

Definition 6 (Open graph) *An open graph is a digraph with two node attributes 'input' and 'output'. A node with the attribute 'input' is called* input node*, a node with the attribute 'output' is called* output node*. A node may hold either none of the two attributes, one of them, or both.*

Definition 7 (Indegree, outdegree) *The indegree of a node $q2$ in a digraph is the number of arcs that are directed to node $q2$ (i. e. have node $q2$ as their head). The outdegree of a node $q1$ is the number of arcs that are directed from node $q1$ (i. e. have node $q1$ as their tail).*

As we only allow for simple graphs, the indegree of a node q is identical to the number of predecessors of this node; the outdegree is equivalent to the number of successors.

Definition 8 (Path) *A path $\mathcal{P}$ is a sequence of nodes $(q_1, q_2, \ldots, q_p)$, $p > 1$, such that for all q_k, $1 \leq k < p$, there is an arc (q_k, q_{k+1}). Node q_1 is called initial node and q_p is called terminal node.*

The length of a path $\mathcal{P}$ is denoted by $|\mathcal{P}|_l$ with $|\mathcal{P}|_l = p - 1$.

Definition 9 (Circuit) *A path is called circuit,*[3] *if its initial node is also its terminal node. Apart from this node, no node occurs more than once in the path.*

Note that a self-loop is a circuit of length 1.

The terms *path* and *circuit* can equivalently be transferred to undirected graphs (at least one edge connecting each pair of nodes $\{q_k, q_{k+1}\}$, $1 \leq k < p$).

[3]We use the term circuit here, because the term 'cycle' is used in a different way in the cyclic scheduling context.

Definition 10 (Connected graph) *An undirected graph is called* connected*, if it contains at least one (undirected) path between each pair of vertices. A digraph $\mathcal{G}$ is said to be connected if its underlying undirected graph $\mathcal{UG}$ is connected. The underlying undirected graph $\mathcal{UG}$ is obtained by removing direction from the arcs of $\mathcal{G}$.*

Definition 11 (Tree) *An undirected* tree *is a graph $\mathcal{T} = (\mathcal{V}, \mathcal{E}_\mathcal{T})$ that does not contain an (undirected) circuit. It contains exactly one undirected path between each pair of vertices. A directed graph $\mathcal{G}$ is called a tree if the underlying undirected graph is a tree.*

A tree with n nodes is therefore connected and contains $n - 1$ edges. A connected graph with n nodes and $n - 1$ edges is always a tree. Note that this definition does not require a directed path between all pairs of nodes in the directed graph. A spanning tree $\mathcal{T} = (\mathcal{V}, \mathcal{E}_\mathcal{T})$ of a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ contains all nodes of $\mathcal{G}$ and a subset of edges which form a tree. Fig. 2.2 shows a spanning tree for the graph of Fig. 2.1.

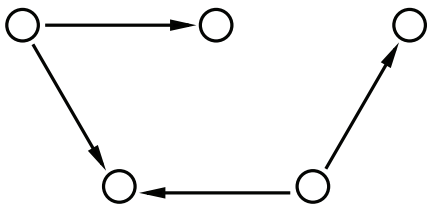

Figure 2.2. A spanning tree for the digraph from Fig. 2.1.

Definition 12 (Weighted graph) *A simple directed graph $\mathcal{G}$, with a real number $wt_{q2,q1} \in \mathbb{R}$ (weight) assigned to each arc $(q1, q2) \in \mathcal{E}$ is called a weighted graph. $wt_{q2,q1}$ is called weight of the arc $(q1, q2)$.*

As we will only consider weighted digraphs in this thesis, the term *graph* will be used in the following to indicate weighted simple directed graphs. An example for a weighted digraph is given in Fig. 2.3.

We always assume that the nodes of a graph are numbered in $\{1, \ldots, |\mathcal{V}|\}$. Expressions $q1, q2, \ldots$ therefore constitute labels as well as node numbers $\in \mathbb{N}$. A weighted graph can then be represented by a *precedence matrix* $WT \in \mathbb{R}_{max}^{n \times n}$ with $n = |\mathcal{V}|$. The entry $wt_{q2,q1}$ of WT represents the weight of the arc $(q1, q2)$ if such an arc exists. Otherwise it

is assigned $-\infty$:

$$wt_{q2,q1} = \begin{cases} \text{weight of arc} & \text{if arc } (q1, q2) \text{ exists,} \\ -\infty & \text{if arc does not exist.} \end{cases}$$

The triple $\mathcal{G} = (\mathcal{V}, \mathcal{E}, WT)$ is used to indicate a weighted graph with weight matrix WT. For graph $\mathcal{G}$ from Fig. 2.3, the precedence matrix WT is

$$WT = \begin{pmatrix} -\infty & -\infty & -\infty & -\infty & -\infty \\ 2 & -4 & -\infty & 5 & -\infty \\ -\infty & -\infty & -\infty & -\infty & 7 \\ 0 & -\infty & -\infty & -\infty & -6 \\ -\infty & 1 & -\infty & 3 & -\infty \end{pmatrix} .$$

Weighting of arcs can be transferred to weighting of paths and circuits: the weight $|\mathcal{P}|_{wt}$ of a path $\mathcal{P} = (q_1, q_2, \cdots, q_p)$, $p > 1$ is equal to the sum of arc weights along the path:

$$|\mathcal{P}|_{wt} = \sum_{j=1}^{p-1} wt_{q_{j+1},q_j} .$$

An element $wt_{q2,q1} > -\infty$ of the precedence matrix WT defines the weight of an arc in the corresponding graph that leads from node $q1$ to node $q2$. In general, an element $(WT^k)_{q2,q1}$ of the k-th power (in max-plus algebra) of WT reflects the weight of a path $\mathcal{P}$ of length $|\mathcal{P}|_l = k$ leading from node $q1$ to node $q2$. If there is more than one path of length k connecting these two nodes, the maximum weight of all such paths is given.

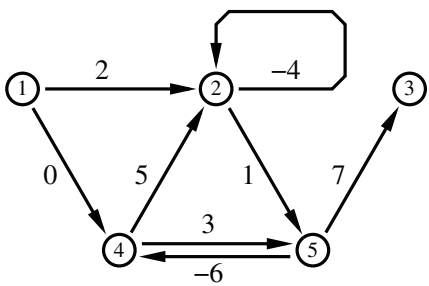

Figure 2.3. Weighted digraph.

Definition 13 (Transitive closure) *A graph $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*)$ is called transitive closure of $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, if $\mathcal{V}^* = \mathcal{V}$ and $\mathcal{E}^* \supseteq \mathcal{E}$ where $(q1, q2) \in \mathcal{E}^*$ if and only if there exists a path in $\mathcal{G}$ from node $q1$ to node $q2$. In the transitive closure $\mathcal{G}^* = (\mathcal{V}^*, \mathcal{E}^*, WT^*)$ of a weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, WT)$, the element $wt_{q2,q1}$ equals the maximum weight of all paths in $\mathcal{G}$ that lead from node $q1$ to node $q2$. By definition, $\mathcal{G}^*$ always incorporates self-loops with weight 0 at all nodes.*

The transitive closure $\mathcal{G}^*$ of a graph $\mathcal{G}$ is a graph with the same set of nodes but additional arcs, such that each pair of nodes that is connected by a path in $\mathcal{G}$ is connected by an arc in $\mathcal{G}^*$. In the case of weighted graphs, the weight matrix WT^* of $\mathcal{G}^*$ is the closure of the weight matrix WT of $\mathcal{G}$ in max-plus algebra, as defined in Equation (2.1). A necessary and sufficient condition for the existence of $\mathcal{G}^*$, is that $\mathcal{G}$ does not contain circuits with negative weight [9].

Definition 14 (Node set contraction) *Let $\mathcal{S} \subset \mathcal{V}$ be a node subset in a digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. By contracting $\mathcal{S}$ to one node qs one obtains a new graph $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$ with $|\mathcal{V}'| = |\mathcal{V}| - |\mathcal{S}| + 1$. The set of arcs $\mathcal{E}'$ is obtained from $\mathcal{E}$ by*

1. *replacing all arcs $(q1, q2)$, $q1 \notin \mathcal{S}, q2 \in \mathcal{S}$ by an arc $(q1, qs)$,*
2. *replacing all arcs $(q1, q2), q1 \in \mathcal{S}, q2 \notin \mathcal{S}$ by an arc $(qs, q2)$, and*
3. *removing all arcs $(q1 \in \mathcal{S}, q2 \in \mathcal{S})$.*

Node qs inherits the input or output attribute of any node $q \in \mathcal{S}$.

In this work, weighted graphs are used for describing subsets in $\mathbb{R}^n$ constrained by linear inequalities. For this, the interpretation of a weighted graph as a *time window precedence network* is introduced:

Definition 15 (Time window precedence network) *A weighted digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, WT)$ with weight matrix WT, where the weights $wt_{q2,q1} \in \mathbb{R}$ represent times is called a* time window precedence network.

In a time window precedence network, nodes represent time instants and arcs represent minimum durations, i. e. time offsets. An arc $(q1, q2) \in \mathcal{E}$ with tail $q1$, head $q2$ and weight $wt_{q2,q1}$ represents the following condition:

$$t_{q2} \geq t_{q1} + wt_{q2,q1} \ . \tag{2.2}$$

In condition (2.2), t_{q1} indicates the time instant for node $q1$ and t_{q2} indicates the time instant for node $q2$. Thus, a time window precedence network $\mathcal{G} = (\mathcal{V}, \mathcal{E}, WT)$ defines a subset of $\mathbb{R}^n$, where n is the number of nodes in the graph: $n = |\mathcal{V}|$.

For example, the subset of $\mathbb{R}^5$ that is defined by Graph $\mathcal{G}$ from Fig. 2.3, is given by $\mathbb{R}^5$ constrained by the following set of linear inequalities:

$$\begin{array}{llll} t_2 \geq t_1 + 2 & t_3 \geq t_5 + 7 & t_4 \geq t_1 & t_5 \geq t_2 + 1 \\ t_2 \geq t_2 - 4 & & t_4 \geq t_5 - 6 & t_5 \geq t_4 + 3 \\ t_2 \geq t_4 + 5 & & & \end{array}$$

Fact 1 (Contraction of two nodes) *two nodes $q1$, $q2$ of a time window precedence network $\mathcal{G}$ with $wt_{q2,q1} = -wt_{q1,q2}$ can be contracted without changing the constraints defined by $\mathcal{G}$.*

Proof In $\mathcal{G}$, arc $(q1, q2)$ represents the condition

$$t_{q2} \geq t_{q1} + wt_{q2,q1} \ . \tag{2.3}$$

Together with the condition represented by arc $(q2, q1)$,

$$t_{q1} \geq t_{q2} + wt_{q1,q2} \tag{2.4}$$

and prerequisite condition $wt_{q2,q1} = -wt_{q1,q2}$, this results in

$$t_{q2} = t_{q1} + wt_{q2,q1} \ . \tag{2.5}$$

Therefore, in all conditions represented by the arcs of $\mathcal{G}$, t_{q2} can be replaced according to (2.5). The resulting set of equations is represented by a time window precedence network $\mathcal{G}'$ that results from $\mathcal{G}$ by contracting nodes $q1$ and $q2$ and setting WT' accordingly. □

Since node $q2$ is eliminated from the graph while contracting it with node $q1$, the relative timing information (2.5) has to be stored for node $q2$.

Fact 2 (Circuits of weight 0) *In a time window precedence network $\mathcal{G}$, all nodes of a circuit of weight 0 can be contracted. This does not affect the constraints defined by $\mathcal{G}$.*

Proof A circuit $\mathcal{C} = (qc, \cdots, qp, qc)$ with weight 0 can be divided into a path $\mathcal{P} = (qc, \cdots, qp)$ and a single arc (qp, qc) whose weight sum is zero:

$$|\mathcal{P}|_{wt} + wt_{qc,qp} = 0 \ . \tag{2.6}$$

Substituting (2.6) into the equation represented by the path,

$$t_{qp} \geq t_{qc} + |\mathcal{P}|_{wt} \tag{2.7}$$

yields

$$t_{qc} \leq t_{qp} + wt_{qc,qp} \ . \tag{2.8}$$

Together with the equation represented by the arc (qp, qc),

$$t_{qc} \geq t_{qp} + wt_{qc,qp} \ , \tag{2.9}$$

this results in

$$t_{qc} = t_{qp} + wt_{qc,qp} \ . \tag{2.10}$$

This is identical to the case of two nodes from Fact 1. □

Fact 2 shows that in a circuit $\mathcal{C}$ with weight $|\mathcal{C}|_{wt} = 0$ the relative timing of all nodes $q \in \mathcal{C}$ is fixed. Contraction of the nodes of a circuit of weight zero can be accomplished by iteratively applying Fact 1.

Graph $\mathcal{G}$ from Fig. 2.3 can be interpreted as a time window precedence network. According to Fact 2, the circuit that is formed by nodes 2, 4, and 5 can be contracted. Fig. 2.4 illustrates the resulting time window precedence network. Table 2.1 provides the relative timing of the eliminated nodes.

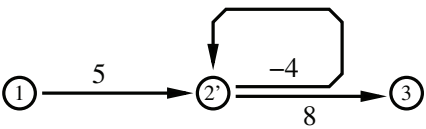

Figure 2.4. Graph $\mathcal{G}$ from Fig. 2.3 with nodes 2, 4, and 5 contracted to node 2'.

Node	connected to node	time difference
2	2'	+0
4	2'	-5
5	2'	+1

Table 2.1. Relative timing of contracted nodes in Fig. 2.4.

Definition 16 (Redundant arc, strongly redundant arc) *An arc* $(q1, q2) \in \mathcal{E}$ *of time window precedence network* $\mathcal{G} = (\mathcal{V}, \mathcal{E}, WT)$ *is called* redundant *if in the graph* $\mathcal{G}' = (\mathcal{V}, \mathcal{E}', WT')$ *with* $\mathcal{E}' = \mathcal{E} \setminus (q1, q2)$*, there exists a path* $\mathcal{P} = (q1, \ldots, q2)$ *with weight* $|\mathcal{P}|_{wt} \geq wt_{q1,q2}$*. It is called* strongly redundant *iff* $|\mathcal{P}|_{wt} > wt_{q1,q2}$*.*

Fact 3 (Redundant arcs) *Arcs in a time window precedence network* $\mathcal{G}$ *that are redundant according to Definition 16 can be deleted. This does not affect the constraints defined by* $\mathcal{G}$*.*

Proof In a time window precedence network, an arc $(q1, q2)$ with weight $wt_{q2,q1}$ represents the following condition:

$$t_{q2} \geq t_{q1} + wt_{q2,q1} \ , \tag{2.11}$$

where t_{qi} is the time instant for the event represented by node qi. Consequently, a path $\mathcal{P} = (q1, \cdots, q2)$ represents the condition

$$t_{q2} \geq t_{q1} + |\mathcal{P}|_{wt} \ . \tag{2.12}$$

If such a path exists and $|\mathcal{P}|_{wt} \geq wt_{q2,q1}$ then (2.12) is a sufficient condition for (2.11).□

Fact 4 (Self-loops) *In a time window precedence network* $\mathcal{G}$*, a self-loop from node* qc *to node* qc *with* $wt_{qc,qc} \leq 0$ *can be deleted. This does not affect the constraints defined by* $\mathcal{G}$*.*

Proof A self-loop represents the equation

$$t_{qc} \geq t_{qc} + wt_{qc,qc} \ , \tag{2.13}$$

which is always true for $wt_{qc,qc} \leq 0$. □

Note that a self-loop from node qc to node qc with $wt_{qc,qc} > 0$ states infeasibility. Such self-loops will not occur if the closure $\mathcal{G}*$ of $\mathcal{G}$ exists.

Petri nets

In addition to the standard graph, we briefly introduce the idea of Petri nets, which form a powerful tool, especially for modeling and visualization of logical and timed discrete event systems. In this thesis, Petri nets will be used only for visualization. We will therefore restrict the following introduction to a basic class of Petri nets. Complete definitions and details on Petri nets can be found, e. g., in [9, 85, 100].

Definition 17 (Petri net) *A Petri net is a pair* $(\mathcal{G}, \mu)$*, where* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *is a bipartite graph,*[4] *where the set of vertices* $\mathcal{V}$ *is split into the disjunct sets* $\mathcal{P}$ *(*places*) and* $\mathcal{Q}$ *(*transitions*); the set of edges* $\mathcal{E}$ *contains pairs* (p_i, q_j) *and* (q_j, p_i)*, where* $p_i \in \mathcal{P}$ *and* $q_j \in \mathcal{Q}$*; the vector* $\mu \in \mathbb{Z}^{|\mathcal{P}|}$ *is called* marking *of the Petri net.*

A place p_i is called *input place* of transition q_j if $\mathcal{E}$ contains an edge (p_i, q_j); it is called *output place* of transition q_j if $\mathcal{E}$ contains an edge (q_j, p_i). A transition q_j is called *input transition* of place p_i if $\mathcal{E}$ contains an edge (q_j, p_i); it is called *output transition* of place p_i if $\mathcal{E}$ contains an edge (p_i, q_j).

A graphical illustration of a Petri net is given in Fig. 2.5. Places are represented by circles and transitions are represented by bars. Places can contain tokens (represented by dots in the places). At any time, the number of tokens in all places is given by the vector μ.

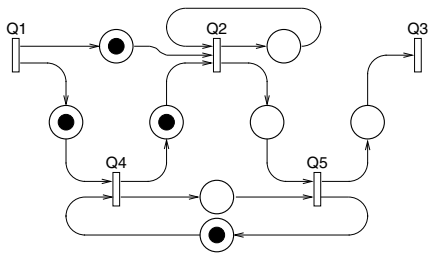

Figure 2.5. A simple Petri net with a marking.

A Petri net not only provides a representation of the dynamic dependencies within a DES model but can also be *executed*, thus simulating the dynamics of the underlying discrete event system. This is based on the *firing rules* for transitions:

- a transition is called *active* or *enabled* when all of its input places contain at least one token,
- an active transition can *fire*: firing removes 1 token from each of the input places of the transition and adds one token to each of its output places.

Definition 18 (Timed Petri net) *A Petri net is called* timed *if times are assigned to places and/or transitions.*

[4] A bipartite graph consists of two different vertex subsets. An edge always connects one vertex from the first subset to one vertex of the second subset.

Times assigned to places are called *holding times*: a token that enters a place will not be available for firing of an output transition before the holding time has passed. Times assigned to transitions are called *firing times*: this is the time it takes the transition to fire. Tokens may stay in the input places or in the output places during that time interval, but they are not available for other transitions.

Definition 19 (Event graph) *A Petri net is called* event graph *(synchronization graph) if all of its places have exactly one input transition and one output transition.*

A max-plus algebra model can be derived from any timed event graph. See [9] for details.

Timed event graphs are related to time window precedence networks as follows: any timed event graph with $\mu = 0$ can furthermore be translated into a time window precedence network,[5] where

1. for each place, the firing time of the input transition is added to its holding time,
2. the set of transitions constitutes the node set of the time window precedence network (inheriting the input/output attribute),
3. the set of places (together with the edge from the input transition and the edge to the output transition) constitutes the arcs of the time window precedence network,
4. the holding times constitute the weights of the respective arcs.

2.2 Mathematical programming

Mathematical programming is concerned with the solution of *optimization problems*. The task of optimization is to find a point within a limited or unlimited space that minimizes or maximizes a certain objective. This section provides an insight into the basics of mathematical programming. As this is a large field of research, the focus will be on those topics that are essential for the problems of this thesis, namely methods to build up and solve mixed integer linear problems. We will not consider dynamic optimization, where the objective has to be calculated by iterative simulation.

[5] An event graph with a *non-zero* marking can as well be translated into a graph model, but would involve an additional *height* attribute for the arcs (see comments in Section 5.2).

A mathematical optimization problem consists of the following main ingredients:

- variables,
- an objective function,
- constraints.

In this thesis, variables will either be real valued or integer valued, the latter also including binary variables. The objective function has to be minimized. Constraints may be of the equality or the inequality type. Both, the objective function and the constraints may involve parameters, which are not considered as variables, since they have a well defined value at the time of computation. In general, a mathematical optimization problem reads as follows:

$$\begin{aligned} \min_{x\in\mathbb{R}^{nc},\, z\in\mathbb{Z}^{nd}} \quad & f(x,z) \\ \text{s.t.} \quad & h(x,z) \;=\; 0 \\ & g(x,z) \;\geq\; 0\,, \end{aligned} \tag{2.14}$$

where x and z are the variables (real valued resp. integer valued), $f(x,z)$ is the objective function, and the vector functions $h : \mathbb{R}^{nc} \times \mathbb{Z}^{nd} \to \mathbb{R}^{ne}$ and $g : \mathbb{R}^{nc} \times \mathbb{Z}^{nd} \to \mathbb{R}^{ni}$ define ne equality constraints and ni inequality constraints.

For $nc > 0$, depending on the value of nd and the structure of the functions f, g, and h, four cases of (2.14) can be distinguished:

f, g, h all linear, $nd = 0$	*linear program*	LP
f, g, h nonlinear, $nd = 0$	*nonlinear program*	NLP
f, g, h all linear, $nd \geq 1$	*mixed integer linear program*	MILP
f, g, h nonlinear, $nd \geq 1$	*mixed integer nonlinear program*	MINLP

The structure of a mathematical programming problem is not 'naturally' connected to its real-world optimization problem: one of the outcomes of this thesis is that the mathematical optimization problem can largely benefit from suitably choosing the way, in which the degrees of freedom are coded by the variables. The way of modeling real-world decision problems as mathematical optimization programs can often decide over success or failure. Therefore, smart modeling techniques are a large field of ongoing research for many application areas [58].

Solution of optimization problems

A *feasible solution* (in mathematical terms also just called *solution*) of the optimization problem (2.14) is any set of admissible values for the variables x and z that meets the constraints given by functions h and g. The set of all feasible solution forms the *feasible region* of the problem. An *optimal solution* is a feasible solution that minimizes the objective function f: no better feasible solution can be found by (local) variations of the values of x and z. During the process of optimization, the term 'optimal solution' may also stand for the best feasible solution found *so far*. A *globally optimal solution* is a feasible solution with the property that there exists no other feasible solution with a better value of f. In this thesis, we are interested in globally optimal solutions. Nevertheless, we do not need to find *all* globally optimal solutions (in fact, many problems exhibit an infinite number of distinct globally optimal solutions). A problem of the form (2.14) is said to be *infeasible* if the set of solutions is empty.

A linear program (LP) forms the basic case of mathematical programming. A well-posed linear program has a unique optimal solution x_{opt}. At least $nc - ne$ of the inequality constraints in g are active in x_{opt}, i. e. $g_i(x_{opt}) = 0$ for at least $nc - ne$ values of i. The most common method for LP solving is the Simplex algorithm resp. more recent primal-dual and interior point methods (see, e. g., [107]), which have the advantage that they can start from intermediate solutions and are often faster. These algorithms return a globally optimal solution or, otherwise, prove infeasibility. Commercial LP solvers are capable of solving large problems with several thousands of variables in a few seconds.

For linear problems, a local search method ('downhill') will always result in a globally optimal solution. This is different for nonlinear programs (NLP), where such methods may end up with local optimal solutions. The quality of the solution will therefore often depend on the choice of good *starting points* for the optimization run. Convergence to globally optimal solutions can be proven only for certain special cases as, e. g., for problems with convex structure. In general, nonlinear problems (NLP, MINLP) are computationally very difficult, even if the problem is unconstrained ($ne = ni = 0$). It turns out that for the cyclic scheduling problems in this thesis a nonlinear formulation of the optimization problem can be avoided. Hence, further details on NLP and MINLPs will not be given in this thesis. The interested reader is referred to appropriate textbooks on nonlinear and global optimization [12, 37].

Optimization problems involving continuous and integer variables are called *mixed integer* programs. An example for an integer linear problem (an MILP with $nc = 0$) is given

in Fig. 2.6. The first idea how to solve a mixed integer linear program (MILP) may be to *relax* the MILP to an LP by skipping the integer condition for the variables z (*LP relaxation*). Of course, the globally optimal LP solution (x'_{opt}, z'_{opt}) does not necessarily meet the integer conditions. This is due to the *integrality gap*, which is the distance between the feasible region of the relaxed problem and the *convex hull* of the set of all integer solutions, i. e. the smallest polyhedron in $\mathbb{R}^{nd}$ containing all integer solutions (see dashed line in Fig. 2.6). Unfortunately, rounding the values of (z'_{opt}) to the nearest integer will *not* yield a globally optimal solution of the underlying mixed integer program and, as the illustration in Fig. 2.6 shows, may not even give a feasible solution.

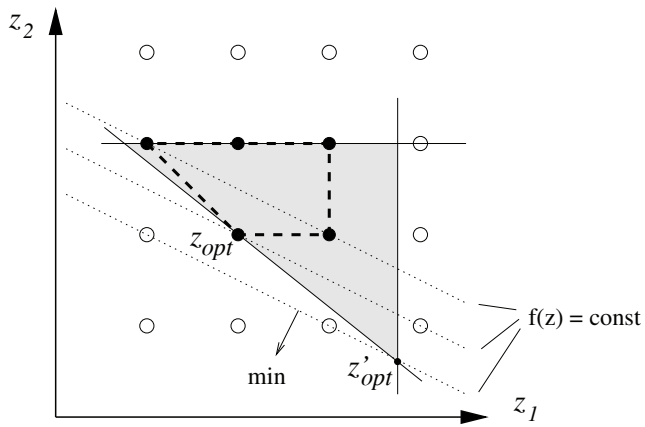

Figure 2.6. Graphical representation of an integer linear program.

Instead, it is inevitable to search the whole decision space given by the variables x and z. Several sophisticated methods have been developed to do this search efficiently. The most popular methods are *branch and bound methods* and *cutting plane methods*. The idea of branch and bound methods is to divide (i. e. to *branch*) the decision space successively into parts ('nodes') for which the relaxed LP problem can provide *bounds*. Based on these bounds, the decision space can be searched in a systematic way: a branch for which the objective function is bounded to a value worse than an already known solution does not need to be taken into account any more. *Cutting plane methods* and *branch and cut* methods prune the search space by adding constraints (*cuts*) that do not violate the convex hull but reduce the size of the feasible region of the relaxed problem. Thus, the integrality gap is reduced without changing the feasible region of the integer problem.

Branching methods are *exact solution methods*: they *implicitly* enumerate the whole decision space. The main advantage against complete-enumeration methods is the fact that (possibly large) parts of the decision space can be ignored after excluding, by use of bounds, the possibility that such parts contain globally optimal solutions. Two fundamental problems have to be addressed during the branch and bound process; namely, where to branch (i. e. for which variables and at which values), and which branch to search first (i. e. deciding about the sequence of nodes). The mathematical optimization process will be more likely to efficiently find globally optimal solutions, if the user provides

- a compact mathematical formulation of the optimization problem,
- strong bounds on the values of variables, and
- additional cuts from the understanding of the underlying process.

For further details on mathematical integer programming, including descriptive explanations and examples, see [107].

Apart from exact solution methods, there exists a large variety of efficient methods based on random search, heuristics and simulation. Examples are *simulated annealing*, *genetic algorithms* or *tabu search*. As these methods are typically not designed to provide proven globally optimal solutions, we did not use them to solve optimization problems for cyclic scheduling. Nevertheless, these methods can be very useful when searching for 'good' solutions in limited computation time. For more ideas on heuristic solutions for real-world problems, see [78].

Modeling tools and solvers

A systematical approach to the process of optimization has to distinguish between two steps: *modeling* the problem as a mathematical program and *solving* the mathematical program. Several software packages are available for both steps. Some of them benefit from the separation of the modeling language and the solution algorithms, which makes it possible to apply several specific solution algorithms to the same problem without the need of recoding the model every time. Modeling languages are used to formulate optimization problems as mathematical models in order to make them available for solvers. There are a variety of modeling languages, often linked to commercially available software, e. g. GAMS [39], AMPL [6], LINGO [71], MPL [82], NUMERICA [53], and OPL

[52, 92]. The presumably most advanced commercially available *solvers* for mixed integer linear programming are CPLEX (Ilog [56]) and Xpress-MP (Dash Optimization [30]). Extensive surveys on software for modeling and optimization can be found in the Internet [93]. For the work presented here, we used GAMS/CPLEX.

The simple model (2.14) provides the interface between arbitrary optimization problems and their solution in mathematical programming. This way, complex problems, e. g. scheduling problems, can be transferred into mathematical programs, which can then be solved efficiently in an application-independent framework. This, of course, does not remove the problem-intrinsic complexity: in principle, the time needed to find a globally optimal solution for an MILP grows exponentially with problem size, or, more precisely, with the number of integer variables [58]. On the other hand, computer technology has developed enormously and processor speed is still rising with extreme growth rates. Therefore, if smart, compact models exist, systematic solutions of relatively large real-world problems can now be derived using mathematical programming, as the next chapters of this thesis will show.

Chapter 3

The cyclic scheduling problem

The general idea of *scheduling* is about allocation of limited resources to activities over time [96]. Sequence and timing decisions have to be taken with respect to certain objectives (optimality constraints). Activities may be, for example, worksteps in a flexible manufacturing system (FMS) or train movements in a railway system or lessons in classroom scheduling. Accordingly, resources may be machines and transportation devices (robots) or track segments or classes, teachers and rooms. The objective function may include overall processing time (makespan) as well as process costs or penalty terms for the violation of due dates. In this work, the terms 'resource' and 'activity' are used in order to reflect the generality of the ideas. Depending on the specific application, the reader may replace 'resource' by 'machine', 'section', or 'worker' and 'activity' by 'operation', 'workstep', or 'task'.

Two basic scheduling frameworks can be distinguished: *static scheduling* and *dynamic scheduling*. A static schedule determines the timing for each activity before the process is started. In contrast, a dynamic scheduling policy describes a set of rules that allow to choose, at each time t, the activities that are to be started next. Static scheduling systems, of course, do not perform well when the timing of an activity depends on effects that are only known at runtime [86]. The main disadvantage of dynamic scheduling is that the solution has to be found within limited computation time. A second disadvantage is that a dynamic scheduling policy does not necessarily result in a regular timing pattern. For high throughput screening and for many other cyclically operated systems, a strictly cyclic schedule with regular pattern is mandatory. Therefore, in this study, we will restrict ourselves to static scheduling. Moreover, we are interested in *exact* solution methods,

i. e. methods to systematically derive schedules that are globally optimal with respect to the scheduling objective rather than just aiming at 'good', but suboptimal schedules obtained in limited calculation time.

3.1 Classification of scheduling problems

A large variety of scheduling problems can be subsumed under the class of *general shop scheduling problems* with three major subclasses: job shop problems, flow shop problems, and open shop problems. All shop problems involve a set of *jobs*, each job consisting of a fixed number of activities that have to be carried out successively. Each activity is assigned to a specific resource (or group of possible resources). The resource is blocked during execution of this activity. A fixed duration is given for each activity. For illustration the reader may assume that for shop problems, a job represents a physical entity (e. g. a workpiece in a production process) passing through the system's resources. In a *flow shop*, each job consists of the same number of activities. Each resource is visited exactly once by each job and the order of activities is the same for all jobs. Only durations may differ between the jobs. In a *job shop*, the order of activities is fixed for each job, but is not necessarily the same for all jobs. If, in a job shop, a job has to visit certain resources more than once, the job is said to *recirculate* [96]. An *open shop* is similar to a flow shop with the relaxation that activities of each job may be processed in any order (no precedence relations between the activities).

Note that for general shops it is always assumed that each job can be handled by only one resource at a time [17], i. e. no two activities of a single job can overlap in time. The precedence structure for the activities of a job in a job shop problem is given in Fig. 3.1. In this thesis, a more generalized notion of the term *job* will be used: A job consists of a set of activities that are linked by a set of timing constraints (e. g. precedence constraints). Depending on the actual structure and values of the timing constraints, activities may follow upon each other, overlap in time (as long as they are not using the same resource) or may even be forced to take place in parallel.[1] Such a structure is called a *precedence network*. Fig. 3.2 shows a simple precedence network, where two different parts pass through their activities, meeting for a common activity ('rendezvous point', e. g. for transfer of substances) and splitting up again. In manufacturing, such a precedence structure may occur,

[1] Note that this enlarged definition of the term 'job' allows that a job may involve more than one physical entity.

e. g., if production of parts involves the assembly of components that are first processed separately. An example for a precedence network with assembling and disassembling of parts is given in Fig. 3.3. Note that, of course, all shop scheduling problems can be described by a precedence network and are therefore still included in our scheme. The idea of precedence network constraints is now further extended to a generalized type of timing constraints.

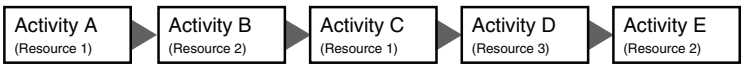

Figure 3.1. Sequence definition for a job in a recirculate job shop problem.

Figure 3.2. Sequence definition for a generalized job type (precedence network).

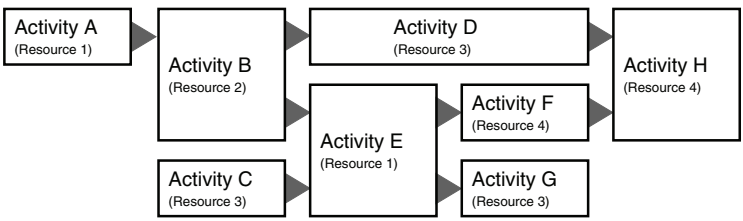

Figure 3.3. A precedence network with assembling (H)/disassembling (B).

3.2 Time window constraints

The most general type of deterministic timing constraints are *time window constraints*, where the duration of a certain time interval is allowed to vary between a lower bound t_l and an upper bound t_u. Upper bounds for time intervals are also referred to as *due dates* resp. *deadlines*.

Time window constraints most frequently arise where chemical or biological processes come into play: A chemical reaction will yield deviant (if not disastrous) results if kept

running for too long. A typical application for scheduling under time window constraints is the production of printed circuit boards, where the boards have to be dipped into chemical baths and the duration of each bath may neither be too short nor too long. Time window constraints are also present in high throughput screening, where, e. g., the duration of incubation phases has to obey strict time bounds.

In classical scheduling theory, activities are assumed to be *monolithic*, i. e. timing constraints only define the sequence and the duration of the activities and therefore involve only start and end times of activities. However, in the most general framework, the term *activity* just defines a time interval, during which the respective resource is blocked. Each activity may internally be composed of multiple worksteps for transfer and processing.[2] We use *discrete event systems* theory to build up system models for this most general case and to describe the constraints that have to be met. This makes it possible to consider not just start and end times of activities but instead to describe the system model in terms of a set of events, together with timing and sequence constraints. An event may represent starting or finishing of activities as well as starting or finishing of transfer processes (synchronization events) or of workstep processes within an activity. Such a DES model allows for time window constraints involving any event that takes place as the system is processing. We are therefore using the term *event-based time window constraints*.

Note that the inclusion of constraints of the time window type in the model formulation still includes simple precedence as the special unbounded case ($t_l = 0$, $t_u = \infty$). Another special case (zero width) is given for so-called *no-wait* constraints, where the start of an activity has to commence immediately at the end of another activity ($t_l = t_u = 0$), or the duration of an activity is fixed to a constant time c ($t_l = t_u = c$). In a no-wait environment, a job must be processed from start to completion without any interruption either on or between machines [45].

In Section 4.1, we will show that simple precedence constraints together with event-based time window constraints can be modeled as a *timed precedence graph* with negative times. We will therefore use the term *time-window precedence network* to characterize a set of activities whose events are connected by time window dependencies. Note that, again, our specification includes the case of simple precedence constraints as a subclass.

[2]Nevertheless, we do not allow for *preemption*, i. e. interrupting of activities for the execution of other activities on the same resource.

3.3 Blocking

In terms of scheduling, a *blocking* environment is a system with zero intermediate buffers between the resources. Blocking occurs when a physical entity that has completed processing on a resource remains on this resource until the next resource becomes available for processing. This, in turn, may prevent an activity of another entity from being started on this resource. In the *DES* terminology, the term 'blocking' refers to the possibility of a *deadlock* situation: in a blocking scheduling environment with dynamic scheduling, a deadlock occurs if two entities a, b allocate resources α, β respectively and each of the two entities waits for the other resource to be freed (i. e. a needs resource β, b needs resource α). The system will then be completely unable to leave this state without manual interaction. An advantage of *static* scheduling is its capability to exclude deadlocks by setting up a *non-blocking specification* in the DES sense: the blocking conditions are included in the mathematical system model together with a specification that only allows for deadlock-free schedules. This specification will be set up in Section 5.2.

Note that blocking may cause idle times within activities. Therefore, the duration of an activity may exceed the duration of the actual worksteps within that activity. To distinguish between the duration of the activity and the duration of the actual worksteps, the term 'minimum processing time' is introduced for the latter: the (deterministic) *minimum processing time* of an activity is the time needed to conduct all worksteps of the activity. If infinite buffers are available between all activities, and no timing constraints apply on the end times of the activities, the actual duration of each activity can be fixed to the value of the minimum processing time.

3.4 Cyclic scheduling

The *cyclic scheduling problem* is characterized by a set of generic activities that have to be executed infinitely many times or during a long period of time. The task of cyclic scheduling is therefore the same as for the standard scheduling problem: to determine the operating sequence and starting times for all activities that have to be carried out. What makes the cyclic scheduling problem different is that the sequence of activities is repeated in a periodical manner. The strictness of periodicity can thereby vary from identical sequences of a subset of activities up to completely identical timing for all periods.

In general, the set of generic activities that has to be executed repetitively represents the

processing of exactly one set of physical entities. The term *batch* is used to identify one instance of the generic set of activities. All activities of a batch belong to the same set of physical entities. The set of generic activities may be grouped into jobs. In cyclic production scheduling, each job represents production of one specific part (some of the jobs, resp. their parts, may be identical), therefore a batch is also referred to as a *minimal part set (MPS)*, which is produced repeatedly.

It is very important to realize that batches may be *nested*: some activities of batch (ρ) take place prior to some activities of the previous batch ($\rho - 1$) or even of batch ($\rho - k$), $1 < k < \infty$. Such cyclic schedules are called *overtaking* [104] (where k is the *overtaking degree*) or *interleaving* [31, 86]. This means that, normally, more than one batch is present in the system at the same time.

In this thesis, *strictly cyclic* schedules are studied. In a strictly cyclic schedule,

- all batches follow exactly the same time scheme (*single batch time scheme*),
- the time distance between the start of consecutive batches is always constant (*cycle time*),
- the last batch does not start before the first batch is finished, i. e. the cyclic schedule could theoretically be repeated infinitely often.

Consequently, in a strictly cyclic schedule, exactly one batch enters the system and one batch leaves the system in each cycle. Strictly cyclic schedules are also called *stable* [66, 67] or *1-periodic*. If the single batch involves only one job or one physical entity, they are called *simple* [21] or *1-unit-cycles*. A generalization of a strictly cyclic (i. e. 1-periodic) schedule would be an *k-periodic* schedule [49]. Such schedules may sometimes be superior to 1-periodic schedules. Note that by considering only strictly cyclic schedules, we are actually not limiting ourselves to 1-periodic schedules, since an k-periodic schedule may always be interpreted as a 1-periodic schedule where the single batch consists of k jobs of the same structure.[3]

For illustration, Fig. 3.4 provides the single batch time scheme, i. e. the generic time scheme for a single batch, for which a strictly cyclic schedule is given in Fig. 3.5. Both figures are given as Gantt charts, which are a convenient tool for the illustration of schedules: activities are represented by bars along the time axis (abscissae). The ordinate

[3] See Chapter 6.4 for details on k-periodic schedules.

distinguishes a discrete set of attributes (normally the resource that is allocated by the activity).

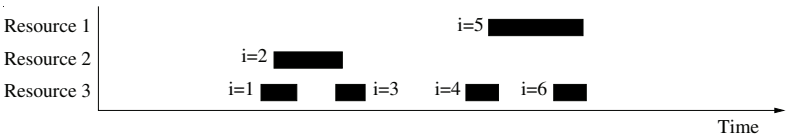

Figure 3.4. Single batch time scheme (Gantt chart).

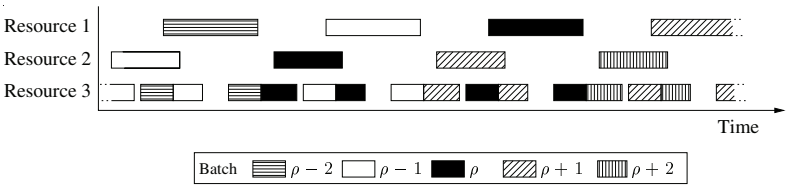

Figure 3.5. Cyclic schedule for the single batch time scheme from Fig. 3.4 (Gantt chart).

The most important characteristic of a cyclic system is its *cycle time* T, i. e. the time distance between the start of consecutive batches.[4] The term *cycle* is naturally defined as a section of the cyclic schedule of length T. The inverse of the cycle time is equivalent to the *throughput* of the system.

3.5 Objective functions for cyclic scheduling

A variety of objective functions can be considered in cyclic scheduling. The most common objective is to increase throughput, i. e. to minimize the cycle time T. If a large, but finite number of batches have to be handled, the objective of throughput maximization is equivalent to the objective of minimization of the *makespan*, i. e. the overall completion time. Another important performance measure for strictly cyclic schedules is the completion time of the single batch, also called the *flow time* of the system. The flow time of a system divided by its cycle time yields the mean number of batches that are present in the system at any time. This quotient is called *work in progress* (WIP). Additionally, a flow time can be assigned to each job: the *job flow time* is the time between the start

[4] For production systems the cycle time is also called *takt time* (e. g. [55]).

and the end of the individual job. A (weighted) sum of job flow times may also be an objective in cyclic scheduling: if, for example, jobs stand for the movement of trains, the job flow times reflect passenger traveling times and have to be minimized. The duration of individual activities may be part of the objective function, if cost is associated to it.

In practice, often a mixture of the abovementioned performance measures has to be considered: the different terms may either enter a common objective function as a weighted sum or the scheduling method may account for them in a hierarchic scheme. For example, the optimal cycle time may be determined in a first step. In a subsequent second step, WIP is minimized while the optimal cycle time is fixed as a hard constraint.

The focus of this thesis is on the solution of the strictly cyclic scheduling problem with non-blocking specification and event-based time window constraints. Generalizations are provided for the case of multi-capacity resources resp. groups of identical resources and for sequence dependent setup times as well as for k-periodic schedules. In this thesis, we will not consider problems with job cancellations or machine breakdowns. All jobs and machines are available at all times. Activities are non-preemptive and each activity uses a well-defined resource or one resource from a group of identical resources. Nevertheless, this model class is fairly general. It covers systems that are blocking or non-blocking, have single-capacity or multi-capacity resources, have precedence network structure or job shop structure, and simple precedence constraints or time window constraints. The proposed methods are therefore capable of modeling and solving instances of many types of cyclic scheduling problems found in the literature.

Chapter 4

Mathematical modeling of cyclic discrete event systems for scheduling

This chapter presents a mathematical modeling framework for cyclically operated timed discrete event systems. As described in Chapter 3, the cyclic process is characterized by two ingredients: the cycle time T and a generic set of activities, of which a large number of instances (batches) are performed in a repetitive way on well-defined resources. For strictly cyclic systems, the timing of each batch is identical. Therefore the timing of the process can be specified by defining the cycle time $T \in \mathbb{R}^+$ and the times $x_e \in \mathbb{R}$ for all events e within the generic batch. We will use the term *single batch time scheme* to characterize the set of generic activities, together with their timing, resp. the timing of their events.[1] Together with the cycle time T, the single batch time scheme defines the timing for the entire cyclic schedule: the times for the events of the ρ-th batch are given by:

$$x_e^{(\rho)} = x_e + \rho \cdot T \ . \tag{4.1}$$

Note that the cyclic model does not account for the overall number of batches: in principle, the cyclic process could be repeated infinitely often, i. e. $\rho \in \mathbb{Z}$.

In Section 4.1, a graph model for the single batch time scheme is derived from the user's problem instance definition. Subsequently, Section 4.2 provides the methods to reduce this graph model and derive a mathematical representation of the single batch time scheme, which will be the basis for the solution methods in Chapter 5.

[1] The single batch time scheme is sometimes also referred to as the *protocol* (see e. g. [86]).

4.1 Single batch time scheme model

The *problem instance definition* given by the user identifies the generic set of activities i, $i = 1, \ldots, n$. It also identifies the set $\mathcal{D}$ of all relevant events within these activities together with their dependencies. The events can be of several types, e. g.:

- starting events $e_o(i)$: marking the start of activity i, i. e. the time instant at which the resource is occupied,
- release events $e_r(i)$: marking the end of activity i, i. e. the time instant at which the resource is released,
- internal events: events within activities that mark the beginning or the end of worksteps or are involved in time window constraints, e. g. the time instant at which a reactant drops into the test substance,
- synchronization events: start or end of transfer of parts, plates or palettes between two resources. A transfer event normally takes place simultaneously to at least one corresponding transfer event associated with an activity on another resource,
- batch start event e_s and batch terminal event e_t: two artificial events defined to simplify notation.

The events are interconnected by temporal interdependencies. These may originate from the following requirements:

- Worksteps (i. e. activities or parts thereof) require a certain minimum time interval for processing. For example, in a drilling machine, a fixed time interval is needed to drill the hole; in a liquid handler, a certain time interval is needed from the moment at which the substance enters the machine and the moment at which the reactant is added. Further examples are the time intervals that are needed for transfer of a workpiece to another resource, or for cleanup between the end of a transfer and the possibility to release the resource for another activity. The durations of these worksteps are normally investigated by test runs on the real plant.
- Starting of one workstep depends on completion of another workstep, e. g. drilling of a workpiece cannot start before clamping is done.

- Two events have to be synchronized, i. e. they are forced to take place at the same instant of time. For example, the start of the transfer of a workpiece between two resources involves a synchronization event on both resources.
- Time window constraints have to hold i. e. the time that is allowed to elapse between two events is restricted by lower and/or upper bounds. Such constraints may, for example, be caused by chemical requirements and will be prescribed by the user. Note that the two events involved may be part of different activities but, of course, have to relate to the same batch.[2]

All durations[3] are assumed to be fixed and deterministic. If a duration is subject to stochastic deviations, a suitable upper bound has to be used for a-priori offline scheduling.

All dependencies can be notated as relations of the form

$$x_{e2} \geq x_{e1} + \Delta t \ , \quad \Delta t \in \mathbb{R} \ .$$

Such a relation can reflect two different constraints depending on the sign of Δt:

$\Delta t \geq 0$: Event $e2$ cannot happen before a time interval of Δt has passed after event $e1$ took place. Time Δt states a *minimum time distance* for the ordered pair of events $(e1, e2)$.

$\Delta t < 0$: The earliest instance of time at which event $e2$ can take place is $-\Delta t$ before event $e1$ takes place. Therefore this constraint states a *maximum time distance* for the ordered pair of events $(e2, e1)$.

We use the term *event based time window constraint* as described in Chapter 3 to reflect this idea.[4]

The events of the generic batch, together with the event based time window constraints, can be described in a structured way using a timed DES graph model. The nodes of the graph represent events in the single batch time scheme; the (directed) arcs of the graph represent temporal interdependencies. A label is assigned to every arc, describing the

[2] An appropriate extension would be possible, but is not part of this thesis since it is only required for some specific applications (see for example [49]) .

[3] Times may be given as real numbers in any time unit as long as the same unit is used for all timing information.

[4] The same idea is reflected by the term 'generalized precedence constraints' (e. g. [18]).

minimum time which has to elapse between the two events.[5] An arc leading from node $q1$ to node $q2$, labeled with time $\Delta t \in \mathbb{R}$ implies that at least time Δt has to pass between occurrence of the event represented by $q1$ and occurrence of the event represented by $q2$. As shown, it is easily possible to model also *upper* bounds for the time interval between two events: if the time interval between the event of node $q1$ and the event of node $q2$ has to be less or equal to Δt, the graph contains an arc from $q2$ to $q1$ labeled by $-\Delta t$. As introduced in Section 2.1, we call such a graph with positive and negative time dependencies a *time window precedence network*.

In this way, a weighted graph representation $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a, WT)$ can be directly created from the problem definition. The set of nodes $\mathcal{V}_a$ contains one node for each event in the single batch time scheme. For notational purposes, the following subsets of $\mathcal{V}_a$ are defined:

$$\mathcal{V}_o \quad \ldots \quad \text{nodes for starting events,} \tag{4.2}$$

$$\mathcal{V}_r \quad \ldots \quad \text{nodes for release events,} \tag{4.3}$$

$$\mathcal{V}_{(i)} \quad \ldots \quad \text{nodes for events in activity } i, \tag{4.4}$$

$$q_s \quad \ldots \quad \text{one batch start node,} \tag{4.5}$$

$$q_t \quad \ldots \quad \text{one batch terminal node.} \tag{4.6}$$

Note that the union of the distinct sets q_s, q_t and $\mathcal{V}_{(i)}$, $i = 1, \ldots, n$ forms the set of all nodes:[6]

$$\sum_{i=1}^{n} \mathcal{V}_{(i)} + q_s + q_t = \mathcal{V}_a \ . \tag{4.7}$$

The set of arcs $\mathcal{E}_a$ represents the temporal interdependencies: the time instant t_{q2} of node $q2$ depends on the time instant t_{q1} of node $q1$ in the following way:

$$t_{q2} \geq t_{q1} + wt_{q2,q1} \ .$$

A number of requirements have to hold for a properly defined model. Firstly, the graph model has to contain exactly one batch start node q_s and one batch terminal node q_t, where

$$\forall q \in \mathcal{V}_o \quad , \quad wt_{q,q_s} = 0 \tag{4.8}$$

$$\forall q \in \mathcal{V}_r \quad , \quad wt_{q_t,q} = 0 \ . \tag{4.9}$$

[5] See Fig. 2.3 on page 12 for an example.

[6] We use the sum notation $\mathcal{X} + \mathcal{Y} = \mathcal{Z}$ for $\mathcal{X} \cup \mathcal{Y} = \mathcal{Z}$ and $\mathcal{X} \cap \mathcal{Y} = \emptyset$.

In addition, for each activity, exactly one node is contained in $\mathcal{V}_o$ and one node is contained in $\mathcal{V}_r$. These nodes are named $q_o(i)$ and $q_r(i)$, respectively:

$$|\mathcal{V}_{(i)} \cap \mathcal{V}_o| = 1\ ,\ \mathcal{V}_{(i)} \cap \mathcal{V}_o =: q_o(i)\ \ \forall i = 1, \ldots, n \tag{4.10}$$

$$|\mathcal{V}_{(i)} \cap \mathcal{V}_r| = 1\ ,\ \mathcal{V}_{(i)} \cap \mathcal{V}_r =: q_r(i)\ \ \forall i = 1, \ldots, n\ . \tag{4.11}$$

No event of an activity can take place before the starting event or after the release event and there has to be a path with strictly positive weight from the starting node to the release node in every activity. A compact notation for these conditions can be given using the closure[7] WT^* of the weight matrix WT:

$$\forall q \in \mathcal{V}_{(i)}, wt^*_{q,q_o(i)} \geq 0 \tag{4.12}$$

$$\forall q \in \mathcal{V}_{(i)}, wt^*_{q_r(i),q} \geq 0 \tag{4.13}$$

$$wt^*_{q_r(i),q_o(i)} > 0\ . \tag{4.14}$$

Equations (4.12) to (4.14) have to hold for all $i = 1, \ldots, n$. They, together with (4.8) and (4.9) ensure that all event times will take place within the interval that is defined by the event times for q_s and q_t and that all activities have a strictly positive duration.

The last condition that has to hold for a properly defined model is the causality constraint that the closure of WT has to exist, i. e.

$$\mathcal{G}_a \text{must not contain circuits of negative weight.} \tag{4.15}$$

The graph model only considers time dependencies between events of the single batch time scheme, i. e. between events that belong to the same batch. These dependencies are called internal constraints [23] or *conjunctive constraints*. In addition to the conjunctive constraints defined by the graph model, there are external constraints resp. *disjunctive constraints*, which will be introduced in Section 5.2. Via disjunctive constraints, event times may not only depend on event times of the same batch but also on event times of other batches. Here, we limit this case to starting events of activities: starting events specify the occupation of a resource; a starting event of an activity cannot occur before the resource is released by the previous activity. Starting events may therefore depend on release events of other activities on the same resource, also those of preceeding or subsequent batches. From the point of view of the single batch time scheme graph model, such events depend on external conditions. This is taken into account by using an open graph. The nodes for starting events, i. e. nodes $q_o(i)$, are *input nodes* of the graph.

[7] See Section 2.1.

Equivalently, nodes $q_r(i)$, are *output nodes*. If the objective function of the scheduling problem contains time instants for events, the nodes of these events have to be marked as input or output nodes, too. This will be specified later in Section 6.1. The attributes 'input' and 'output' are defined via two more subsets of $\mathcal{V}_a$:

$$\mathcal{V}_{ip} \quad \ldots \quad \text{input nodes,} \tag{4.16}$$

$$\mathcal{V}_{op} \quad \ldots \quad \text{output nodes.} \tag{4.17}$$

In the graph illustration, *input nodes* are marked by an incoming arc. Correspondingly, *output nodes* are marked with an outgoing arc.

A vast majority of strictly cyclic scheduling problems can be addressed using this modeling framework. An example for a problem instance definition in form of a time window precedence network can be found in Fig. 4.2 on page 44.

At this point, the time window precedence network could, of course, be directly translated into mathematical inequality constraints arc by arc. In this case, one would need one variable for each event, which would describe the time instant at which this event occurs in the single batch time scheme. The number of variables would therefore equal the number of nodes and each arc would provide one inequality restriction, implying that the mathematical formulation becomes unnecessarily complex or even unmanageable. Mathematical optimization algorithms will often perform much better on compact problem descriptions than on unnecessarily complex, inflated or intricate formulations. Heuristic presolve steps have been developed to deal with inflated problem formulations in practice. However, the only information standardized preprocessing algorithms of solvers have available is the optimization problem formulation in terms of Equations 2.14. Therefore, the engineer's structural knowledge about the process should be used to substantially simplify the mathematical description of the problem before starting mathematical optimization algorithms. This will be done in the following section by

1. performing simplification steps on the DES model (based on the graph representation), and

2. using knowledge about degrees of freedom that are unnecessary (i. e. that can be reduced or even removed without harming the objective value of the globally optimal solution).

4.2 Model reduction

The raw time window precedence network generated from the definition of events and their temporal interdependencies as given in Section 4.1 is the weighted graph $\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a, WT)$. So far, $\mathcal{G}_a$ contains exactly one node for each event e from the set of events $\mathcal{D}$ that is needed to describe the single batch time scheme. We will now simplify the single batch time scheme model by contracting nodes in the graph. To preserve timing information for all events $e \in \mathcal{D}$, we introduce a more general notation: for a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, WT)$, a mapping $\mathcal{M} : \mathcal{D} \mapsto \mathcal{V} \times \mathbb{R}$ assigns a node and a time offset to each event $e \in \mathcal{D}$. To simplify this notation, we assume that all nodes in $\mathcal{V}$ are sorted and numbered in $\mathbb{N}$. The mapping $\mathcal{M}$ therefore consists of a node number $\mathcal{Q}[e] \in \mathbb{N}$ and a time offset $\Delta[e] \in \mathbb{R}$ for each event e. Thus, the time instant x_e at which event e takes place is defined by

$$x_e = t_{\mathcal{Q}[e]} + \Delta[e] \; . \tag{4.18}$$

Time offsets $\Delta[e]$ are necessary to preserve timing information about those events whose nodes will later be removed from the graph by node contraction. Initially, we can set $\Delta[e] = 0 \;\; \forall \, e \in \mathcal{D}$. The initial mapping $\mathcal{Q}[e]$ is a one-to-one mapping $\mathcal{D} \mapsto \{1 \ldots |\mathcal{D}|\}$, since the graph contains one unique node for each event $e \in \mathcal{D}$.

The arcs of the time window precedence network represent the timing constraints that have to hold for the time instants of the events. For nodes that are neither input nor output nodes, the graph represents the complete set of constraints that have to hold for the time instants of their related events. As described in the previous section, only event times for nodes with the input or output attribute may be involved in other constraints of the overall scheduling problem.

The size of the graph model can now be reduced by

- removing redundant arcs according to Fact 3,
- removing self-loops according to Fact 4, and
- contracting nodes according to Facts 1 and 2.

In addition to these steps, the following two Facts can be used to reduce the size of the graph model.

Fact 5 (Nodes with indegree 1) *In a time window precedence network $\mathcal{G}$ representing the single batch time scheme, a node qc with indegree 1 that is not an input node, can be contracted with its predecessor without violating any of the internal or external constraints defined by $\mathcal{G}$.*

Proof There is only one predecessor qp of qc. The arc between qp and qc represents the constraint

$$t_{qc} \geq t_{qp} + wt_{qc,qp} \ . \tag{4.19}$$

As node qc is not an input node, all other constraints that involve variable t_{qc} are of type

$$t_q \geq t_{qc} + wt_{q,qc} \ . \tag{4.20}$$

For any vector t in the original system that satisfies (4.19) and (4.20), the vector t',

$$t'_{qi} = \begin{cases} t_{qp} + wt_{qc,qp} & , i = c \\ t_{qi} & , i \neq c \end{cases} \tag{4.21}$$

also meets (4.19) and (4.20) and therefore still satisfies all internal and external constraints defined by the original time window precedence network. □

Note that this node contraction actually reduces the degrees of freedom in the model but does not interfere with the cycle time of the globally optimal cyclic schedule.

Fact 6 (Nodes with outdegree 1) *In a time window precedence network $\mathcal{G}$ representing the single batch time scheme, a node qc with outdegree* 1 *that is not an output node, can be contracted with its successor without violating any of the internal or external constraints defined by $\mathcal{G}$.*

The application of Fact 6, again, reduces the degrees of freedom in the model without interfering with the globally optimal cycle time. The proof is dual to the proof for Fact 5.

Model reduction starts by initializing the graph model to $\mathcal{G}^{(0)} = \mathcal{G}_a$, $\mathcal{V}^{(0)} = \mathcal{V}_a$, $\mathcal{E}^{(0)} = \mathcal{E}_a$, and $WT^{(0)} = WT$. Each node contraction step reduces the number of nodes, i. e. $|\mathcal{V}^{(k+1)}| < |\mathcal{V}^{(k)}|$. The weight matrix $WT^{(k+1)}$ as well as the mapping $\mathcal{Q}[\mathcal{D}]^{(k+1)}$, $\Delta[\mathcal{D}]^{(k+1)}$ have to be adjusted accordingly.

Model reduction steps from Facts 1 to 3 as well as Facts 5 and 6 may be applied iteratively until no further reduction possibilities remain. The algorithms for such graph reduction steps are standard knowledge in graph theory. For complete algorithmic descriptions, the reader is referred to Appendix B.

The proposed method does not guarantee a minimal single batch time scheme realization, i. e. a minimal number of nodes or arcs and therefore does not guarantee a minimum number of variables and constraints. Nevertheless, the size of the graph is reduced deci-

sively[8] and computation times for the globally optimal solution went down for all problem instances addressed in this work.

Finally, these model reduction steps result in a graph model

$$\mathcal{G}_c = \mathcal{G}^{(final)} = (\mathcal{V}_c, \mathcal{E}_c, WT_c)$$

and a mapping $\mathcal{Q}[\mathcal{D}] = \mathcal{Q}[\mathcal{D}]^{(final)}$, $\Delta[\mathcal{D}] = \Delta[\mathcal{D}]^{(final)}$.

In the following, a mathematical description is generated from this graph model. An equivalent mathematical representation for the constraints defined by $\mathcal{G}_c$ is

$$A't + \hat{\theta} = c' \,, \quad \hat{\theta} \geq 0 \tag{4.22}$$

where $\hat{\theta} \in \mathbb{R}_+^{K+P}$ are non-negative slack variables, $K = |\mathcal{V}_c| - 1$, $K + P = |\mathcal{E}_c|$, A' represents the topology of the graph, and c' contains the weights of the arcs. Every arc $(q1, q2)$ in $\mathcal{G}_c$ with weight $wt_{c,q2,q1}$ represents a condition of the form

$$t_{q2} \geq t_{q1} + wt_{c,q2,q1} \tag{4.23}$$

and therefore results in one row of the linear system (4.22).

The mapping of the $K + P$ arcs to the indices $k = 1, \ldots, K + P$ of $\hat{\theta}$ (i. e. the sorting of indices in $\hat{\theta}$) has to be chosen such that the arcs representing the first K entries form a tree within $\mathcal{G}_c$ according to Definition 11. These variables parametrize the degrees of freedom of the single batch time scheme. The remaining entries of $\hat{\theta}$ are slack variables of P additional constraints, i. e. $\hat{\theta} = [\theta^T \; \breve{\theta}^T]^T$, $\hat{\theta} \in \mathbb{R}^{K+P}$, $\theta \in \mathbb{R}^K$, $\breve{\theta} \in \mathbb{R}^P$. Matrix A' is in $\mathbb{R}^{(K+P)\times(K+1)}$ and $\text{rank}(A') = K$.[9]

It is convenient to fix the absolute position of the single batch time scheme on the time axis, e. g. by setting the time instant for the batch start node to zero. This adds one line to the top of the linear system (4.22), which finally reads

$$At + B_1\theta + B_2\breve{\theta} = c \tag{4.24}$$

with $A = [A_1{}^T \; A_2{}^T]^T$, $A_1 \in \mathbb{R}^{(K+1)\times(K+1)}$, $A_2 \in \mathbb{R}^{P\times(K+1)}$, $B_1 = [\emptyset^K \; I^{K\times K} \; \emptyset^{K\times P}]^T$, $B_2 = [\emptyset^{P\times K+1} \; I^{P\times P}]^T$, and $c \in \mathbb{R}^{K+1+P}$. As the sorting of $\hat{\theta}$ has been chosen such that the arcs representing its first K entries form a tree, A_1 has rank $K + 1$.

[8] For almost all problem instances addressed in this work, the method actually results in a minimal realization in terms of variables, reducing their number up to a factor of 100.

[9] As usual, X^T refers to the transposed of X, $I^{x\times x}$ is the $x \times x$ conventional identity matrix, and $\emptyset^{x\times y}$ is an all-zero $x \times y$ matrix.

Since $[A\ B_2]$ is a quadratic $K+1+P$-matrix of the form

$$\begin{bmatrix} A_1 & 0 \\ A_2 & I \end{bmatrix}$$

where A_1 has full rank, the inverse $[A\ B_2]^{-1}$ exists. Equation (4.24) is therefore equivalent to

$$\begin{bmatrix} t \\ \breve{\theta} \end{bmatrix} = \underbrace{-[A\ B_2]^{-1}B_1}_{\bar{B}}\,\theta + \underbrace{[A\ B_2]^{-1}c}_{\bar{c}} \ . \tag{4.25}$$

In general, the choice of the mapping between the arcs and the indices of $\hat{\theta}$ is not unique, but as long as the arcs that are represented by the first K entries of $\hat{\theta}$ form a tree, any choice guarantees a valid parametrization. A variable θ_k with $\bar{B}_{q,k} \geq 0\ \forall q$ can be intuitively interpreted as an artificial delay that is inserted between two activities of the single batch time scheme or, e. g., between the last workstep of an activity and the end of this activity (release event). This suggests to choose the sorting of $\hat{\theta}$ such that the first $K+1$ rows of $\bar{B}$ form a matrix with nonnegative entries.

In the upper part of Equation (4.25) the time instants for all $K+1$ nodes in $\mathcal{G}_{\mathcal{J}^c}$ are expressed as linear combinations of the time variables θ:

$$t_q = \bar{c}_q + \sum_{k=1}^{K} (\bar{B}_{q,k} \cdot \theta_k)\ ,\ q = 1, \ldots, K+1\ . \tag{4.26}$$

The lower part of Equation (4.25) states P linear inequality constraints on the admissible values of θ:

$$\sum_{k=1}^{K} (\bar{B}_{K+1+p,k} \cdot \theta_k) + \bar{c}_{K+1+p} = \breve{\theta}_p \geq 0\ ,\ p = 1, \ldots, P\ . \tag{4.27}$$

Using (4.26) and the mapping (4.18), the time instant x_e for an event e within the single batch time scheme results in

$$x_e = \bar{c}_{\mathcal{Q}[e]} + \sum_{k=1}^{K} (\bar{B}_{\mathcal{Q}[e],k} \cdot \theta_k) + \Delta[e]\ . \tag{4.28}$$

In particular, the time instant o_i for the starting event $e_o(i)$ and the time instant r_i for the release event $e_r(i)$ of activity i are

$$\begin{aligned} o_i &= \chi_{i,0} + \sum_{k=1}^{K} (\chi_{i,k} \cdot \theta_k) \quad \ldots \quad \text{time, when activity } i \text{ starts ,} \\ r_i &= \psi_{i,0} + \sum_{k=1}^{K} (\psi_{i,k} \cdot \theta_k) \quad \ldots \quad \text{time, when activity } i \text{ ends .} \end{aligned} \tag{4.29}$$

with the abbreviations

$$\chi_{i,0} = \bar{c}_{\mathcal{Q}[e_o(i)]} + \Delta[e_o(i)] \tag{4.30}$$

$$\chi_{i,k} = \bar{B}_{\mathcal{Q}[e_o(i)],k} \ , \quad k = 1, \ldots, K \tag{4.31}$$

$$\psi_{i,0} = \bar{c}_{\mathcal{Q}[e_r(i)]} + \Delta[e_r(i)] \tag{4.32}$$

$$\psi_{i,k} = \bar{B}_{\mathcal{Q}[e_r(i)],k} \ , \quad k = 1, \ldots, K \tag{4.33}$$

for $i = 1, \ldots, n$.

The time instants for the batch start event e_s and the batch terminal event e_t are

$$x_{e_s} = \bar{c}_{\mathcal{Q}[e_s]} + \sum_{k=1}^{K} (\bar{B}_{\mathcal{Q}[e_s],k} \cdot \theta_k) + \Delta[e_s] \ \ldots \text{ batch start time,} \tag{4.34}$$

$$x_{e_t} = \bar{c}_{\mathcal{Q}[e_t]} + \sum_{k=1}^{K} (\bar{B}_{\mathcal{Q}[e_t],k} \cdot \theta_k) + \Delta[e_t] \ \ldots \text{ batch terminal time.} \tag{4.35}$$

The inequality constraints (4.27) finally read

$$\sum_{k=1}^{K} (\kappa_{p,k} \cdot \theta_k) - \vartheta_p + \breve{\theta} = 0 \ , \ \ p = 1, \ldots, P \ . \tag{4.36}$$

with slack variables $\breve{\theta}_p \in \mathbb{R}_+, p = 1, \ldots, P$ and the abbreviations

$$\vartheta_p = \bar{c}_{K+1+p} \tag{4.37}$$

$$\kappa_{p,k} = -\bar{B}_{K+1+p,k} \ , \quad k = 1, \ldots, K \ . \tag{4.38}$$

Equations (4.29) to (4.38) define a parametrization of all time instants within the single batch time scheme that are important for the scheduling problem. This model provides the basis for the solution of the strictly cyclic scheduling problem, which is derived in the next chapter.

4.3 Simple example: modeling

In this section, the proposed method will be illustrated by a simple example. The same example will be continued in Chapter 5 to illustrate the solution method for the cyclic scheduling problem.

A cyclic schedule has to be derived for the single batch time scheme from Section 3, which is repeated for convenience in Fig. 4.1. It consists of $n = 6$ activities. One activity

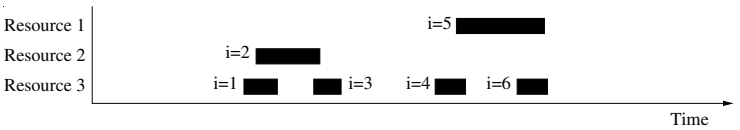

Figure 4.1. Single batch time scheme (Gantt chart).

takes place on both, resource 1 and 2:

$$J_2 = 1,\ J_5 = 2\ . \tag{4.39a}$$

Four activities take place on resource 3:

$$J_1 = J_3 = J_4 = J_6 = 3\ . \tag{4.39b}$$

The corresponding time window precedence network is given in Fig. 4.2. Arcs are labeled with minimum time distances as described in Section 4.1. Input nodes, i. e. nodes that correspond to activity starting events, are labeled with an incoming arc. Output nodes, i. e. nodes that correspond to release events, are labeled with an outgoing arc. In order to keep the graph well readable, the arcs connecting the batch start node q_s and the batch terminal node q_t with the activity start and release nodes are only sketched in the background. Dotted lines with small numbers connecting the graph to the Gantt chart in the lower part of the figure show the mapping $\mathcal{Q}[\mathcal{D}]$ and the time offset $\Delta[\mathcal{D}]$ for the activity start and release events.

The events for those pairs of nodes that are connected with arcs labeled with time 0 in both directions need to happen simultaneously. These events represent the (start of) transfer of a plate between resources. The arcs labeled by * are constraints stated by the user: the transfer event associated with node c occurs at least 47 and at most 82 time units after the event associated with node b. Hence, the relative timing of both events can vary by up to 35 time units.

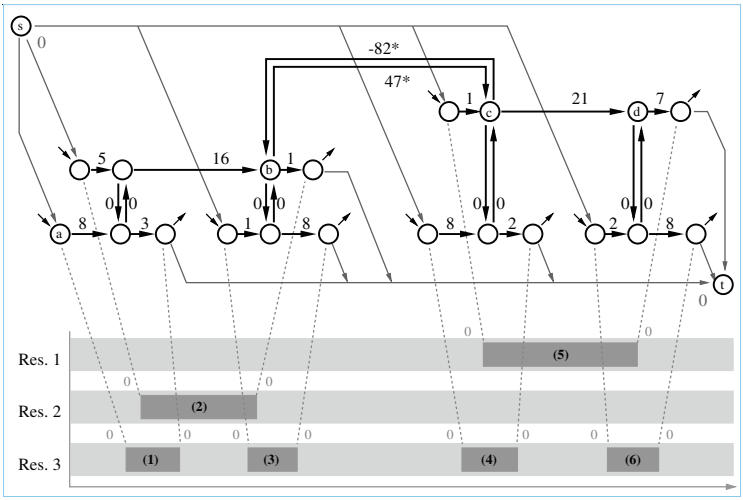

Figure 4.2. Time window precedence network.

Fig. 4.3 shows the graph that is obtained after conducting the model reduction steps from Section 4.2. The number of nodes has been reduced from 22 down to 4, the number of arcs is 4 instead of 36.

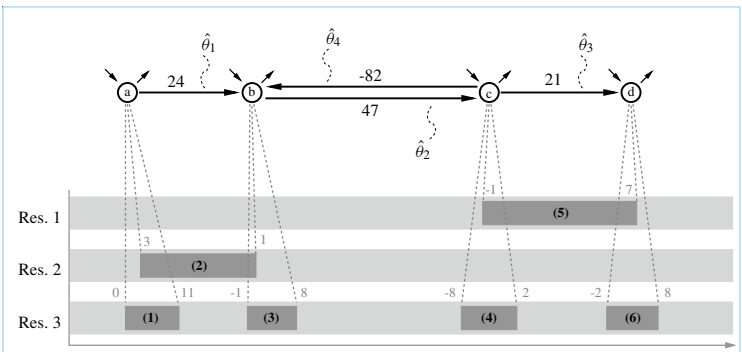

Figure 4.3. Reduced time window precedence network.

Following the modeling framework from Section 4.2, a variable $\hat{\theta}_k$ is assigned to each arc. The assignment is defined in Fig. 4.3. The arcs assigned to $\hat{\theta}_1$ to $\hat{\theta}_3$ build up a tree as

required. Since the graph consists of 4 arcs out of which 3 form a tree, we have $K = 3$ and $P = 1$. The linear equation system (4.24), which defines the event times for the four nodes (a,b,c,d) of the reduced time window precedence network, is

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & -1 & 1 & 0 \end{bmatrix}}_{A} t + \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}}_{B_1} \theta + \underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}}_{B_2} \breve{\theta} = \underbrace{\begin{bmatrix} 0 \\ -24 \\ -47 \\ -21 \\ 82 \end{bmatrix}}_{c} \tag{4.40}$$

which, according to (4.25), is equivalent to

$$\begin{bmatrix} t_1 \\ \vdots \\ t_4 \\ \breve{\theta} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & -1 & 0 \end{bmatrix}}_{\bar{B}} \theta + \underbrace{\begin{bmatrix} 0 \\ 24 \\ 71 \\ 92 \\ 35 \end{bmatrix}}_{\bar{c}} \tag{4.41}$$

The mapping (4.18) between the nodes of the reduced graph and the events of the single batch time scheme is pictured in the lower part of Fig. 4.3. According to definitions (4.30) to (4.33), the parameters $\chi_{i,0}$, $\psi_{i,0}$, $\chi_{i,k}$, and $\psi_{i,k}$ take the following values:

$$\begin{array}{llllllllll}
\chi_{1,0} = 0 & \chi_{1,1} = 0 & \chi_{1,2} = 0 & \chi_{1,3} = 0 & \psi_{1,0} = 11 & \psi_{1,1} = 0 & \psi_{1,2} = 0 & \psi_{1,3} = 0 \\
\chi_{2,0} = 3 & \chi_{2,1} = 0 & \chi_{2,2} = 0 & \chi_{2,3} = 0 & \psi_{2,0} = 25 & \psi_{2,1} = 1 & \psi_{2,2} = 0 & \psi_{2,3} = 0 \\
\chi_{3,0} = 23 & \chi_{3,1} = 1 & \chi_{3,2} = 0 & \chi_{3,3} = 0 & \psi_{3,0} = 32 & \psi_{3,1} = 1 & \psi_{3,2} = 0 & \psi_{3,3} = 0 \\
\chi_{4,0} = 63 & \chi_{4,1} = 1 & \chi_{4,2} = 1 & \chi_{4,3} = 0 & \psi_{4,0} = 73 & \psi_{4,1} = 1 & \psi_{4,2} = 1 & \psi_{4,3} = 0 \\
\chi_{5,0} = 70 & \chi_{5,1} = 1 & \chi_{5,2} = 1 & \chi_{5,3} = 0 & \psi_{5,0} = 99 & \psi_{5,1} = 1 & \psi_{5,2} = 1 & \psi_{5,3} = 1 \\
\chi_{6,0} = 90 & \chi_{6,1} = 1 & \chi_{6,2} = 1 & \chi_{6,3} = 1 & \psi_{6,0} = 100 & \psi_{6,1} = 1 & \psi_{6,2} = 1 & \psi_{6,3} = 1
\end{array} \tag{4.42}$$

It turns out that the parametrization has been chosen in such a way that the time variables $\theta_1, \ldots, \theta_3$ can be interpreted as delays, which are inserted in the sequence of activities at the following positions:

- Variable θ_1 describes a delay during activity $i = 2$, just before the transfer event associated with node b,
- variable θ_2 describes an extension of the time interval between activity $i = 3$ and activity $i = 4$ (No resource is allocated by the plate at that time),

- variable θ_3 describes a delay during activity $i = 5$, just before the transfer event associated with node d.

According to definition (4.37) and (4.38), the parameters ϑ and κ take the following values:

$$\vartheta_1 = 35, \quad \kappa_{1,1} = 0, \ \ \kappa_{1,2} = 1, \ \ \kappa_{1,3} = 0 \, . \tag{4.43}$$

They form the constraint

$$\theta_2 \leq 35 \, , \tag{4.44}$$

which, in this case, simply represents an upper bound for θ_2. This upper bound turns out to define the time window condition represented by the arc labeled by -82^*.

To sum up, the single batch time scheme model consists of parameters

- $J_i, \ i = 1, \ldots, 6$ according to (4.39),
- $\chi_{i,0}, \psi_{i,0}, \chi_{i,k}, \psi_{i,k}, \ i = 1, \ldots, 6, \ k = 1, \ldots, 3$ according to (4.42),
- $\vartheta_p, \kappa_{p,k}, \ p = 1, \ k = 1, \ldots, 3$ according to (4.43),

and the following variables:

- $\theta_k \ , \ k = 1, \ldots, 3.$

Additionally, in order to reconstruct the timing for *all* events of the single batch time scheme after having solved the scheduling problem, we have to preserve the mapping $\mathcal{Q}[\mathcal{D}]$ and $\Delta[\mathcal{D}]$ as shown in the lower part of Fig. 4.3.

Chapter 5

Scheduling problem solution

In this chapter, we present a method to solve the strictly cyclic scheduling problem under the requirement of maximum throughput. We assume that

- the set of generic activities for the single batch is well-determined,
- each activity allocates a well-defined resource.

In order to focus on the basic structure of the problem, we first also claim the following:

- all resources have single capacity,
- setup times (if any) are fixed and are included in the duration of each individual activity.

In Chapter 6, the latter two conditions will be dropped.

After deriving, in Sections 5.1 to 5.4, a mixed integer linear program for the cyclic scheduling problem, the method is illustrated by a simple example in Section 5.5. Section 5.6 presents additional constraints, which allow for an accelerated computation of the globally optimal solution.

5.1 Optimization problem formulation

Based on the single batch time scheme parametrization derived in Sections 4.1 and 4.2, the cyclic scheduling problem can now be formulated as an optimization problem.

The problem of scheduling is to find optimal sequences in which the activities should take place on the resources. Hence, only activity start and release times are relevant for the mathematical formulation of the scheduling problem. Therefore, using parametrization (4.29), the relevant information about the single batch time scheme is defined by the fixed parameters $\chi_{i,0}$, $\psi_{i,0}$, $\chi_{i,k}$, and $\psi_{i,k}$, $i = 1, \ldots, n$, $k = 1, \ldots, K$ and yet unknown variables $\theta_k \geq 0$, $k = 1, \ldots, K$, for which P linear constraints of the form

$$\sum_{k=1}^{K} (\kappa_{p,k} \cdot \theta_k) \leq \vartheta_p \ , \ \ p = 1, \ldots, P \tag{5.1}$$

have to hold.

The modeling procedure of Section 4.1 ensures that the duration of all activities is strictly positive:

$$r_i > o_i, \ \ i = 1, \ldots, n \ . \tag{5.2}$$

Each activity i allocates precisely one resource, denoted by J_i as prescribed in the batch definition by the user:

$$\begin{aligned} J_i \quad &\ldots \quad \text{resource allocated by activity } i, \\ J_i &\in \{1, \ldots, m\}. \end{aligned}$$

Of course, any resource may be allocated subsequently by different activities. Within the single batch time scheme, the number of activities allocating resource j is

$$n_j = \sum_{i=1}^{n} \delta_{J_i j} \ , \ \ j = 1, \ldots, m \ , \tag{5.3}$$

$$\text{where } \delta_{J_i j} = \begin{cases} 0 & \text{for } J_i \neq j \\ 1 & \text{for } J_i = j \ . \end{cases}$$

For the scheduling problem, the aim is to derive a schedule with maximum throughput. For strictly cyclic schedules, exactly one batch is started and one batch is completed per cycle. The objective of maximizing throughput is therefore equivalent to the objective of minimizing cycle time T. Hence, this is the objective function for the mathematical programming problem:

$$\min T \ . \tag{5.4}$$

Obviously, a condition for the cycle time T is that it can never be smaller than the sum of activity durations on any resource during the single batch time scheme:

$$T \geq \sum_{i=1}^{n} (r_i - o_i) \delta_{J_i j} \ , \ \ j = 1, \ldots, m \ . \tag{5.5}$$

$$\text{where } \delta_{J_i j} = \begin{cases} 0 & \text{for } J_i \neq j \\ 1 & \text{for } J_i = j \ . \end{cases}$$

Substituting (4.29) into (5.5) results in

$$\gamma_{j,0} + \sum_{k=1}^{K} (\gamma_{j,k} \cdot \theta_k) - T \leq 0 \ , \quad j = 1, \ldots, m \ , \tag{5.6}$$

where

$$\gamma_{j,0} = \sum_{i=1}^{n} (\psi_{i,0} - \chi_{i,0}) \delta_{J_i j} \tag{5.7}$$

$$\gamma_{j,k} = \sum_{i=1}^{n} (\psi_{i,k} - \chi_{i,k}) \delta_{J_i j} \ . \tag{5.8}$$

Therefore, throughput maximization for a cyclic system is equivalent to the following scheduling problem:

> Find $\theta_k, k = 1, \ldots, K$ such that the cycle time T is minimized subject to constraints (5.1), (5.6) and the additional constraint that no resource allocation conflicts arise.

In the next section, we will derive mathematical conditions for the latter requirement.

5.2 Disjunctive constraints

In our framework, all resources have capacity one. Hence, a resource can never be allocated by two activities simultaneously.[1] This has not only to hold for the single batch time scheme but also with regard to all batches that are present in the plant at the same time. This requirement leads to mathematical conditions, which are often called *disjunctive constraints*. The disjunctive constraints, at the same time, guarantee exclusion of blocking (*non-blocking specification*): a cyclic schedule for which all disjunctive constraints as well as the single batch time scheme constraints hold, is obviously a valid schedule and can be applied with infinite repetitions. Hence, as long as the sequence defined by this schedule is not violated, a deadlock (i. e. blocking in the DES sense) cannot occur.

[1] However, there are no additional switching times, i. e. two activities on the same resource can immediately follow upon each other.

For two activities $(i1, i2)$ of the same resource $J_{i1} = J_{i2}$ belonging to batch $\rho 1$ and $\rho 2$, respectively, the following condition ensures exclusion of overlapping, i. e. ensures that the disjunctive constraints hold:[2]

$$\begin{gathered} o_{i1}^{(\rho 2)} \geq r_{i2}^{(\rho 1)} \;\text{ OR }\; o_{i2}^{(\rho 1)} \geq r_{i1}^{(\rho 2)}, \\ \text{for all } i1, i2 \in \{1, \ldots, n\},\; \rho 1, \rho 2 \in \mathbb{Z},\; i1 \leq i2,\; J_{i1} = J_{i2}, \end{gathered} \tag{5.9}$$

i. e. this condition has to be met for each pair of activities using the same resource, including activities within the same batch and including the same activity within different batches.

Condition (5.6) ensures (5.9) for $i1 = i2$. Note that $r_i > o_i,\; i = 1, \ldots, n$, and

$$o_i^{(\rho)} = o_i + \rho \cdot T \tag{5.10a}$$

$$r_i^{(\rho)} = r_i + \rho \cdot T\,, \tag{5.10b}$$

where $T \in \mathbb{R}^+$.

Theorem 1 (Disjunctive constraints) *For any pair $i1, i2 \in \{1, \ldots, n\}$, $i1 < i2$, $J_{i1} = J_{i2}$, the mutual exclusion condition (5.9) holds if and only if there exists an integer value $z_{(i1,i2)} \in \mathbb{Z}$, such that*

$$z_{(i1,i2)} \cdot T - (o_{i2} - r_{i1}) \leq 0 \tag{5.11a}$$

$$\Big(z_{(i1,i2)} + 1\Big) \cdot T - (r_{i2} - o_{i1}) \geq 0\,. \tag{5.11b}$$

Proof

(5.9) $\Rightarrow$ (5.11)

Consider any pair $i1, i2, i1 < i2,\;\; J_{i1} = J_{i2}$. Substituting (5.10a) and (5.10b) into (5.9) yields

$$\begin{gathered} o_{i1} + \rho 2 \cdot T \geq r_{i2} + \rho 1 \cdot T \;\;\text{ XOR }\;\; o_{i2} + \rho 1 \cdot T \geq r_{i1} + \rho 2 \cdot T \\ \forall \rho 1, \rho 2 \in \mathbb{Z}, \end{gathered} \tag{5.12}$$

which can be reformulated as follows:

$$\begin{gathered} o_{i1} + \underbrace{(\rho 2 - \rho 1)}_{=:\rho} \cdot T \geq r_{i2} \;\text{ XOR }\; o_{i2} \geq r_{i1} + \underbrace{(\rho 2 - \rho 1)}_{=:\rho} \cdot T \\ \forall \rho 1, \rho 2 \in \mathbb{Z}\,, \end{gathered} \tag{5.13}$$

[2] In (5.9), OR and XOR are equivalent as only one part of Condition (5.9) can be satisfied.

or, equivalently,

$$o_{i1} + \rho \cdot T \geq r_{i2} \tag{5.14a}$$

XOR

$$o_{i2} \geq r_{i1} + \rho \cdot T \tag{5.14b}$$

$$\forall\ \rho \in \mathbb{Z}\ .$$

We now show that there always exists an integer value $z_{(i1,i2)} \in \mathbb{Z}$, for which (5.11a) and (5.11b) hold: we choose the integer variable $z_{(i1,i2)}$ to take the value

$$z_{(i1,i2)} = \lfloor \frac{o_{i2} - r_{i1}}{T} \rfloor\ , \tag{5.15}$$

where $\lfloor x \rfloor$ denotes the floor-function, i.e. the largest integer number that is less or equal to x.

From this definition, it immediately follows that

$$z_{(i1,i2)} \leq \frac{o_{i2} - r_{i1}}{T} \tag{5.16}$$

and therefore (5.11a) is met.

Since (5.14) is true for *any* ρ, it has to hold for

$$\rho = z_{(i1,i2)} + 1 \tag{5.17}$$

$$\text{i.e.} \quad \rho = \lfloor \frac{o_{i2} - r_{i1}}{T} \rfloor + 1\ . \tag{5.18}$$

For this value of ρ, condition (5.14b) is not satisfied. Hence, condition (5.14a) holds. Substituting (5.17) into (5.14a) gives

$$o_{i1} + (z_{(i1,i2)} + 1) \cdot T \geq r_{i2} \tag{5.19}$$

and therefore (5.11b).

(5.11) $\Rightarrow$ (5.9)

To show sufficiency, for any pair $i1, i2 \in \{1, \ldots, n\},\ i1 < i2,\quad J_{i1} = J_{i2}$, we consider two cases:

- For the case $\rho \geq z_{(i1,i2)} + 1$, together with (5.11b), we get

$$\rho \geq z_{(i1,i2)} + 1 \geq \frac{r_{i2} - o_{i1}}{T} ,$$

and therefore

$$o_{i1} + \rho \cdot T \geq r_{i2} . \tag{5.20}$$

- For the remaining case $\rho < z_{(i1,i2)} + 1$, since ρ and $z_{(i1,i2)}$ are integer numbers, we get

$$\rho \leq z_{(i1,i2)} ,$$

which, together with (5.11a), is equivalent to

$$o_{i2} \geq r_{i1} + \rho \cdot T . \tag{5.21}$$

Hence, (5.11) is a sufficient condition for

$$o_{i1} + \rho \cdot T \geq r_{i2} \quad \text{OR} \quad o_{i2} \geq r_{i1} + \rho \cdot T \qquad \forall \rho \in \mathbb{Z} \tag{5.22}$$

or, equivalently, for

$$o_{i1} + (\rho 2 - \rho 1) \cdot T \geq r_{i2} \quad \text{OR} \quad o_{i2} \geq r_{i1} + (\rho 2 - \rho 1) \cdot T \qquad \forall \rho 1, \rho 2 \in \mathbb{Z} . \tag{5.23}$$

□

To sum up, a necessary and sufficient condition for the exclusion of allocation conflicts is that

$$z_{(i1,i2)} \cdot T - (o_{i2} - r_{i1}) \leq 0 \tag{5.24a}$$

$$\Big(z_{(i1,i2)} + 1 \Big) \cdot T - (r_{i2} - o_{i1}) \geq 0 \tag{5.24b}$$

holds for each pair of activities $(i1, i2), i1 < i2, J_{i1} = J_{i2}$, where $z_{(i1,i2)}$ is an integer number: $z_{(i1,i2)} \in \mathbb{Z}$.

Note that (5.24) only needs to be considered once for each unordered pair $(i1, i2)$, i. e. $i1 \leq i2$. Nevertheless, it holds for any $(i1, i2)$, including those cases where $i1 > i2$, when defining

$$z_{(i2,i1)} := -1 - z_{(i1,i2)} \quad \forall \ \Big\{ \ (i2, i1), \ i2 \neq i1, \ i2, i1 \in \{1, \ldots, n\} \ \Big\} . \tag{5.25}$$

In the remainder of this section, we give a short physical interpretation for the integer variables $z_{(i1,i2)}$. This is not a prerequisite for the following parts of this thesis.

Consider a resource in a cyclic process that is used by two activities within the single batch time scheme, numbered $i1, i2$. The mathematical model according to (5.24) contains an integer variable $z_{(i1,i2)}$.

Combining (5.24), (5.2) and (5.10) results in

$$o_{i1}^{(\rho 1)} < o_{i1}^{(\rho 2)} < o_{i2}^{(\rho 1)} \; \forall \rho 2, \;\; \rho 1 < \rho 2 \leq \rho 1 + z_{(i1,i2)} \; . \tag{5.26}$$

For $o_{i2} > o_{i1}$ and $z_{(i1,i2)} \geq 0$, this allows for the following physical interpretation for the integer variable $z_{(i1,i2)}$:

Between activity $i1$ for a batch $\rho 1$ and activity $i2$ for the same batch, the resource is used for exactly $z_{(i1,i2)}$ activities of type $i1$ of subsequent batches, i. e. for batches $\rho 2 \in \{\rho 1 + 1, \ldots, \rho 1 + z_{(i1,i2)}\}$.

The integer variable $z_{(i1,i2)}$ could therefore be called the *nesting degree* of the sorted pair of activities $(i1, i2)$.

Fig. 5.1 illustrates this interpretation of $z_{(i1,i2)}$: for a simple constant single batch time scheme with two activities, five examples with different cycle times and different values of $z_{(i1,i2)}$ are given as Gantt charts.

The disjunctive constraints have been derived and formulated in a mathematical way rather than by including them into the graph model. It should be noted that it would nevertheless be possible to extend the graph model from the single batch time scheme such that it also includes the disjunctive constraints. For this, the framework of time-window precedence networks has to be extended towards a bi-valued graph [21], where, in addition to the time distance ('length'), a second weight has to be assigned to each arc ('height'): in this bi-valued graph, the height of the arc leading from the node associated to the end of activity $i1$ to the node associated to the start of activity $i2$ would exactly equal $-z_{(i1,i2)}$.

This bi-valued graph can as well be described as an event graph (i. e. a special Petri net, see Definition 19) with the height of an arc in the bi-valued graph being equivalent to the number of tokens in the initial marking of the Petri net. The optimization problem would then be to do some search for an initial marking that allows for minimum cycle time (remember that max-plus algebra can be used to calculate the minimum possible cycle time for an event graph). The main ideas of this approach have been described in [40]. However, in order to cover the complete class of cyclic systems considered in this thesis,

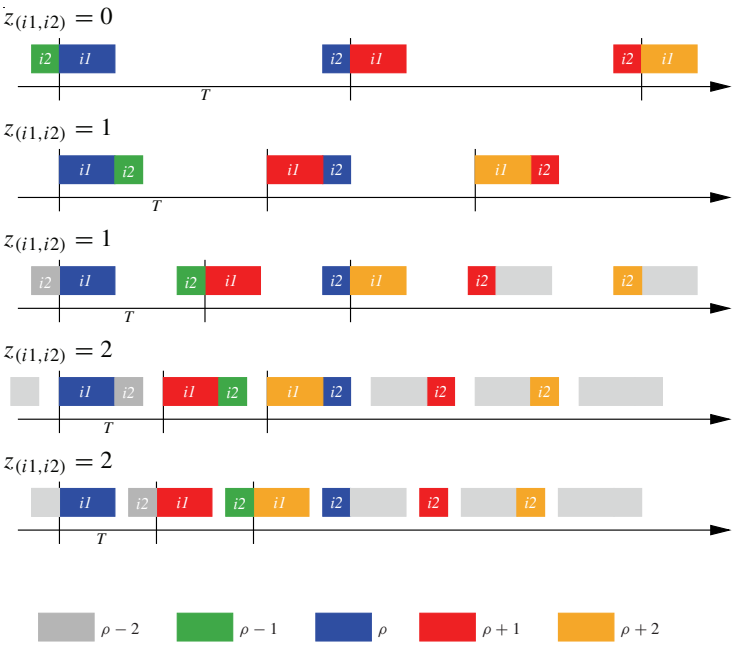

Figure 5.1. Timing examples with different nesting degrees for two activities (Gantt charts).

this approach requires a notion of "negative tokens" and a framework to extend max-plus algebra accordingly. Also, time window constraints cannot be handled. Although max-plus algebra allows for a very efficient calculation of the minimum possible cycle time for a given event graph, the max-plus algebra step only allows to compute the *objective function* for a feasible solution that is already given. In order to solve the optimization problem, we would still need a framework to *search* the space of all feasible solutions in an intelligent way.

For these reasons, it is much more efficient to solve the cyclic scheduling problem by formulating it as a mathematical program. With the model for the single batch time scheme and the disjunctive constraints given as mathematical equations, it is actually just one straightforward step to write down the cyclic scheduling problem as a mixed integer mathematical program.

5.3 Mathematical program

We now formulate the scheduling problem as a mathematical program. For compact notation, some abbreviations are introduced.

Each possible pair of indices $(i1, i2)$, $i2 > i1$, $J_{i1} = J_{i2}$, is denoted by a single number ι,

$$\iota = 1, \ldots, \iota_{max} \ , \quad \iota_{max} = \sum_{j=1}^{m} \frac{n_j(n_j - 1)}{2} \ . \tag{5.27}$$

Hence, each value for ι signifies a pair of activities (within the single batch time scheme) using the same resource.

Substituting (4.29) into (5.24a) and (5.24b) results in

$$z_\iota \cdot T - \ v_{\iota,0} - \sum_{k=1}^{K} (\ v_{\iota,k} \cdot \theta_k) \ \leq \ 0 \tag{5.28}$$

$$\Big(z_\iota + 1 \Big) \cdot T - w_{\iota,0} - \sum_{k=1}^{K} (w_{\iota,k} \cdot \theta_k) \ \geq \ 0 \tag{5.29}$$

with the following abbreviations:

$$v_{\iota,k} \ = \ \chi_{i2,k} - \psi_{i1,k} \tag{5.30}$$

$$w_{\iota,k} \ = \ \psi_{i2,k} - \chi_{i1,k} \tag{5.31}$$

$$v_{\iota,0} \ = \ \chi_{i2,0} - \psi_{i1,0} \tag{5.32}$$

$$w_{\iota,0} \ = \ \psi_{i2,0} - \chi_{i1,0} \ . \tag{5.33}$$

Constraints (5.28) and (5.29) have to hold for all $\iota = 1, \ldots, \iota_{max}$.

Formulating the scheduling problem as a mathematical program, the cycle time T has to be taken as the objective function to be minimized under the constraints given by Equations (5.28) and (5.29) as well as (5.1) and (5.6). The decision space for the optimization problem is defined by the following variables:

- cycle time $T \in \mathbb{R}^+$,
- time variables $\theta_k \in \mathbb{R}_0^+, \ k = 1, \ldots, K$,
- integer variables $z_\iota \in \mathbb{Z}, \ \iota = 1, \ldots, \iota_{max}$.

Hence, the basic cyclic scheduling problem can be written as the following mixed integer nonlinear program:

Min T subject to

$$z_\iota \cdot T - \ v_{\iota,0} - \sum_{k=1}^{K} (\ v_{\iota,k} \cdot \theta_k) \leq 0 \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{5.34a}$$

$$\Big(z_\iota + 1 \Big) \cdot T - w_{\iota,0} - \sum_{k=1}^{K} (w_{\iota,k} \cdot \theta_k) \geq 0 \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{5.34b}$$

$$\sum_{k=1}^{K} (\kappa_{p,k} \cdot \theta_k) \leq \vartheta_p \qquad \text{for } p = 1, \ldots, P \tag{5.34c}$$

$$\gamma_{j,0} + \sum_{k=1}^{K} (\gamma_{j,k} \cdot \theta_k) - T \leq 0 \qquad \text{for } j = 1, \ldots, m \tag{5.34d}$$

Any globally optimal solution to (5.34) constitutes a throughput-optimal cyclic schedule for the underlying problem instance. Several software packages are available to address mixed integer nonlinear problems.[3] However, a straightforward application of such software packages to the above optimization problem can result in long computation times even for small problems. Moreover, in many cases, globally optimal solutions will not be found.[4]

Fortunately, it turns out that (5.34) can be reformulated in mixed integer linear form, as shown in the following section.

5.4 Mixed integer linear program

The nonlinear mixed integer optimization problem (5.34) can be transformed into a linear problem by a reparametrization of the feasible region. Introducing

$$\bar{T} := \frac{1}{T} \quad , \quad \bar{T} \in \mathbb{R}^+ \tag{5.35a}$$

and

$$\bar{\theta}_k := \frac{\theta_k}{T}, \ k = 1, \ldots, K \quad , \quad \bar{\theta}_k \in \mathbb{R}_0^+ \ , \tag{5.35b}$$

[3] For example GAMS/SBB [39].

[4] See also Section 2.2.

the optimization problem reads as follows:

Min $-\bar{T}$ subject to

$$z_\iota - \ v_{\iota,0} \cdot \bar{T} - \sum_{k=1}^{K} (\ v_{\iota,k} \cdot \bar{\theta}_k) \leq 0 \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{5.36a}$$

$$z_\iota + 1 - w_{\iota,0} \cdot \bar{T} - \sum_{k=1}^{K} (w_{\iota,k} \cdot \bar{\theta}_k) \geq 0 \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{5.36b}$$

$$\sum_{k=1}^{K} (\kappa_{p,k} \cdot \bar{\theta}_k) \leq \vartheta_p \cdot \bar{T} \qquad \text{for } p = 1, \ldots, P \tag{5.36c}$$

$$\gamma_{j,0} \cdot \bar{T} + \sum_{k=1}^{K} (\gamma_{j,k} \cdot \bar{\theta}_k) - 1 \leq 0 \qquad \text{for } j = 1, \ldots, m \tag{5.36d}$$

Equations (5.36a) to (5.36d) state a mixed integer linear program (MILP). This optimization problem can be efficiently solved using standard methods of mathematical programming, as described in Section 2.2. Attention has to be paid to numerical aspects due to the fact that now the reciprocal of the original objective function is used: $\bar{T}$ has to be defined as a strictly positive variable and both $\bar{T}$ and $\bar{\theta}$ should be scaled appropriately.

5.5 Simple example: scheduling problem solution

In this section, we carry on with the simple example from Section 4.3. Note that we could immediately find two *suboptimal* solutions to the scheduling problem:

- One (suboptimal) solution is to start each batch at the time when the previous batch is finished. The single batch time scheme is chosen in such a way that the batch is finished as fast as possible. This obviously yields a cycle time of $T_1 = \psi_{6,0} - \chi_{1,0} = 100$.
- To find another suboptimal solution, we remove degrees of freedom by setting $\theta_1 = \theta_2 = \theta_3 = 0$. Hence, the only remaining degree of freedom is T and we determine the smallest T for which activities on the same resource do not overlap. This solution can be found by simple algorithms in polynomial time.[5] It is pictured in Fig. 5.2 (repetition of Fig. 3.5 from page 30) and has a cycle time of $T_2 = 50$.

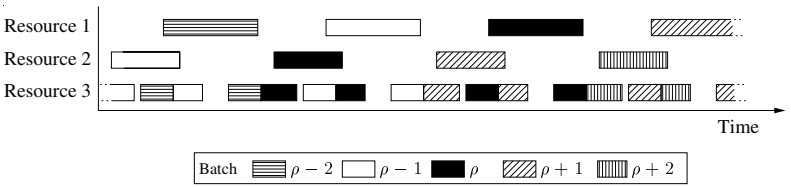

Figure 5.2. Suboptimal cyclic schedule for the single batch time scheme from Fig. 4.1.

In order to find a *globally optimal* solution to the scheduling problem, the search space is now enlarged by allowing variables θ_k, $k = 1, 2, 3$ to be greater than zero.

There are six pairs of activities $(i1, i2)$, $i2 > i1$, $J_{i1} = J_{i2}$ for which constraints (5.24) need to be considered:

$$(i1, i2) \in \{(1, 3), (1, 4), (1, 6), (3, 4), (3, 6), (4, 6)\} \ .$$

This means we need to introduce the integer variables $z_1 = z_{(1,3)}$, $z_2 = z_{(1,4)}$, $z_3 = z_{(1,6)}$, $z_4 = z_{(3,4)}$, $z_5 = z_{(3,6)}$, and $z_6 = z_{(4,6)}$.

The abbreviations (5.30) to (5.33) provide the following notation:

$$\begin{array}{llll}
v_{1,0} = 12 & v_{1,1} = 1 & v_{1,2} = 0 & v_{1,3} = 0 \\
w_{1,0} = 32 & w_{1,1} = 1 & w_{1,2} = 0 & w_{1,3} = 0 \\
v_{2,0} = 52 & v_{2,1} = 1 & v_{2,2} = 1 & v_{2,3} = 0 \\
w_{2,0} = 73 & w_{2,1} = 1 & w_{2,2} = 1 & w_{2,3} = 0 \\
v_{3,0} = 79 & v_{3,1} = 1 & v_{3,2} = 1 & v_{3,3} = 1 \\
w_{3,0} = 100 & w_{3,1} = 1 & w_{3,2} = 1 & w_{3,3} = 1 \\
v_{4,0} = 31 & v_{4,1} = 0 & v_{4,2} = 1 & v_{4,3} = 0 \\
w_{4,0} = 50 & w_{4,1} = 0 & w_{4,2} = 1 & w_{4,3} = 0 \\
v_{5,0} = 58 & v_{5,1} = 0 & v_{5,2} = 1 & v_{5,3} = 1 \\
w_{5,0} = 77 & w_{5,1} = 0 & w_{5,2} = 1 & w_{5,3} = 1 \\
v_{6,0} = 17 & v_{6,1} = 0 & v_{6,2} = 0 & v_{6,3} = 1 \\
w_{6,0} = 37 & w_{6,1} = 0 & w_{6,2} = 0 & w_{6,3} = 1
\end{array} \tag{5.37}$$

[5] We suggest to successively exclude all intervals for the cycle time T, for which two activities on the same resource would overlap in time. Another algorithm which solves this problem can, e. g., be found in [61].

Hence, we get six pairs of constraints of the form (5.34a) resp. (5.34b):

$$\begin{aligned}
z_1 \cdot T - (\ 12 \quad &+\theta_1 \qquad\quad\ \ &) \ &\leq\ 0 \\
(z_1+1) \cdot T - (\ 32 \quad &+\theta_1 \qquad\quad\ \ &) \ &\geq\ 0 \\
z_2 \cdot T - (\ 52 \quad &+\theta_1 \quad +\theta_2 \qquad &) \ &\leq\ 0 \\
(z_2+1) \cdot T - (\ 73 \quad &+\theta_1 \quad +\theta_2 \qquad &) \ &\geq\ 0 \\
z_3 \cdot T - (\ 79 \quad &+\theta_1 \quad +\theta_2 \quad +\theta_3\ &) \ &\leq\ 0 \\
(z_3+1) \cdot T - (\ 100 \quad &+\theta_1 \quad +\theta_2 \quad +\theta_3\ &) \ &\geq\ 0 \\
z_4 \cdot T - (\ 31 \quad &\qquad\ \ +\theta_2 \qquad &) \ &\leq\ 0 \\
(z_4+1) \cdot T - (\ 50 \quad &\qquad\ \ +\theta_2 \qquad &) \ &\geq\ 0 \\
z_5 \cdot T - (\ 58 \quad &\qquad\ \ +\theta_2 \quad +\theta_3\ &) \ &\leq\ 0 \\
(z_5+1) \cdot T - (\ 77 \quad &\qquad\ \ +\theta_2 \quad +\theta_3\ &) \ &\geq\ 0 \\
z_6 \cdot T - (\ 17 \quad &\qquad\qquad\ \ +\theta_3\ &) \ &\leq\ 0 \\
(z_6+1) \cdot T - (\ 37 \quad &\qquad\qquad\ \ +\theta_3\ &) \ &\geq\ 0
\end{aligned} \tag{5.38}$$

With (4.43), Condition (5.34c) gives

$$\theta_2 \leq 35\ . \tag{5.39}$$

With values (4.42) and abbreviations (5.7) and (5.8), Condition (5.34d) results in

$$T \geq 29 + \theta_3, \qquad T \geq 22 + \theta_1, \qquad T \geq 40\ . \tag{5.40}$$

Hence, the mixed integer nonlinear program, which has to be solved in order to obtain a globally optimal solution to the scheduling problem is the following:

> Min T over
> $\big(\ T \in \mathbb{R}^+,\ \ \theta_k \in \mathbb{R}_0^+,\, k \in \{1,2,3\},\ \ z_\iota \in \mathbb{Z},\, \iota \in \{1,\ldots,6\}\ \big)$ such that conditions (5.38), (5.39) and (5.40) hold.

The solution to this problem can be found using a MINLP solver or, of course, by transforming the problem into a mixed integer linear program using (5.35). A globally optimal solution is

$$T = 40,\ \ \theta_1 = 8,\ \theta_2 = 30,\ \theta_3 = 3,\ \ z_1 = 0,\ z_2 = 2,\ z_3 = 3,\ z_4 = 1,\ z_5 = 2,\ z_6 = 0$$

A graphical representation of the resulting cyclic schedule is given in Fig. 5.3. As additional degrees of freedom have been added, it is no surprise that this solution is better than the suboptimal solutions $T_1 = 100$ and $T_2 = 50$.

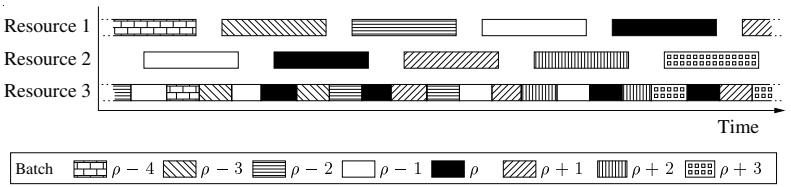

Figure 5.3. Cyclic schedule for the single batch time scheme from Fig. 3.4 with modified timing.

This example illustrated the main solution procedure of the proposed method for a simple problem. Using a standard PC, the related mathematical program can be solved within milliseconds. There is no necessity for additional ideas to further decrease computation time. However, for problems of large size, the time needed to solve the mixed integer program 5.36 may increase exponentially with ι_{max}. Therefore, additional constraints help to keep computing time fast. Such constraints are presented in the following section.

5.6 Strengthening the problem formulation

The set of equations (5.36) completely defines the mixed integer linear program that solves the strictly cyclic scheduling problem. Nevertheless, computation of the globally optimal solution of a mixed integer linear program like (5.36) may be drastically accelerated by providing additional information to the solver. Such information consists of additional constraints that reduce the 'search space'. The additional constraints may even reduce the feasible region of the optimization problem but, of course, must be guaranteed not to cut off all globally optimal solutions. In this section, we derive a number of such additional constraints. The topics of this section are not a prerequisite for the understanding of the other parts of the thesis.

We consider three types of constraints:

- lower and upper *bounding constraints* on the variables of (5.36), and
- *cutting planes* (*cuts*), i. e. constraints that do not change the feasible region of the mixed integer program but reduce the feasible region of its LP relaxation.
- additional bounding constraints that reduce the feasible region of the optimization problem but at the same time guarantee the persistence of at least one globally optimal solution.

We recall from Section 2.2 that the LP relaxation of the MILP (5.36) is the linear mathematical program (LP) that is obtained by allowing the integer variables z to take any value in $\mathbb{R}$. Every feasible solution of the MILP is also a feasible solution of its LP relaxation but not vice versa.

Bounds for variables

At any time, the bounds for a variable in an MILP can be derived by solving the MILP's LP relaxation with the objective of minimizing respectively maximizing this variable. Adding these bounds as additional constraints will not change the feasible region of the LP or its relaxation, but they may have to be known in order to deduce cuts or bounding constraints.

Since the objective function represents minimization of the cycle time T, an upper bound T_{max} for the globally optimal value of T is provided by *any* feasible solution to the mixed integer linear program (5.36). A first bound $T_{max,1}$ can be easily derived by demanding $z_\iota \in \{-1, 0\}\ \forall \iota = 1, \ldots, \iota_{max}$. This corresponds to the restriction that any activity i on resource j must not start before all activities of previous batches on the same resource have been completed. Another bound $T_{max,2}$ can be derived by fixing the values for the time variables θ from the optimal solution found with $T_{max,1}$ and allowing the variables z_ι to take any integer value: $z_\iota \in \mathbb{Z},\ \iota = 1, \ldots, \iota_{max}$. In addition to those two bounds, for many practical cases, a (suboptimal) solution may be known in advance, thus providing a third bound $T_{max,3}$. The resulting upper bound for the optimal value of T is then

$$T_{max} = \min(T_{max,1}, T_{max,2}, T_{max,3})\ .$$

This bound can be added to the mixed integer linear program in terms of an additional bounding constraint

$$T \leq T_{max}\ . \tag{5.41}$$

A lower bound T_{min} with

$$T \geq T_{min} \tag{5.42}$$

is found by solving the LP relaxation of the mixed integer program (5.36).

Cuts

We will now provide a first type of *cuts*: lower and upper bounds for the integer variables z_ι can be found from the LP relaxation of the mathematical program (5.36) including (5.41). For integer variables, lower bounds can be rounded up and upper bounds can be rounded down. The resulting bounding constraints

$$z_{\iota,min} \leq z_\iota \leq z_{\iota,max} \tag{5.43}$$

constitute cuts when added to the mixed integer linear program.

A second type of cuts can be obtained by generalizing this approach to pairs of variables z: for any four disjunct activities $(i3, i4, i5, i6)$ with $J_{i3} = J_{i4}$, $J_{i5} = J_{i6}$ the following relation has to hold:

$$z_{(i3,i4)} - z_{(i5,i6)} \leq - \underline{z}_{(i3,i4,i5,i6)} \ , \tag{5.44}$$

with

$$\underline{z}_{(i3,i4,i5,i6)} := \begin{cases} \lceil \frac{D_{i3,i4,i5,i6}}{T_{min}} \rceil - 1 & \text{for } D_{i3,i4,i5,i6} < 0 \\ \lceil \frac{D_{i3,i4,i5,i6}}{T_{max}} \rceil - 1 & \text{for } D_{i3,i4,i5,i6} \geq 0 \end{cases} \tag{5.45}$$

and

$$D_{i3,i4,i5,i6} = \min_{\theta_1 \ldots \theta_K \text{ s.t. } (5.34c)} \Big(r_{i3} - o_{i5} + r_{i6} - o_{i4} \Big) \tag{5.46}$$

$$= \min_{\theta_1 \ldots \theta_K \text{ s.t. } (5.34c)} \Big(\psi_{i3,0} - \chi_{i5,0} + \psi_{i6,0} - \chi_{i4,0} + \sum_{k=1}^{K} (\psi_{i3,k} - \chi_{i5,k} + \psi_{i6,k} - \chi_{i4,k}) \cdot \theta_k \Big) \ . \tag{5.47}$$

The validity of (5.44) follows from disjunctive constraints (5.24a) with $i1 := i3, i2 := i4$ and (5.24b) with $i1 := i5, i2 := i6$. The proof is given in Appendix A.

The values for $D_{i3,i4,i5,i6}$ for any combination $(i3, i4, i5, i6)$ with $J_{i3} = J_{i4}$, $J_{i5} = J_{i6}$ may be calculated by solving the LP relaxation of the MILP (5.36) $\wedge$ (5.41) minimizing (5.47). Nevertheless, it may be almost as effective but much faster to derive a lower bound for the expression (5.46) directly from the graph model for the single batch time scheme: recall that the entry $wt^*_{q2,q1}$ of the closure of the original weight matrix $WT = WT^{(0)}$ represents the minimum allowable time distance between the events represented by nodes $q1$ and $q2$. A lower bound $\underline{D}_{i3,i4,i5,i6}$ for $D_{i3,i4,i5,i6} = r_{i3} - o_{i5} + r_{i6} - o_{i4}$ is therefore

$$\underline{D}_{i3,i4,i5,i6} = wt^*_{q_r(i3),q_o(i5)} + wt^*_{q_r(i6),q_o(i4)} \ , \tag{5.48}$$

where $q_r(i3)$ and $q_r(i6)$ are the nodes for the release events of activity $i3$ and $i6$ respectively, and $q_o(i4)$ and $q_o(i5)$ are the nodes for the starting events of activity $i4$ and $i5$.

Using $\underline{D}_{i3,i4,i5,i6}$ instead of $D_{i3,i4,i5,i6}$ in (5.45) may result in lower, thus more conservative values for $\underline{z}_{(i3,i4,i5,i6)}$ but hence is on the safe side.

When generalizing (5.44) to non-disjunct activities, the cut for $i3 = i4$ results in

$$z_{(i5,i6)} \geq \underline{z}_{(i3,i4,i5,i6)} - 1 \tag{5.49}$$

and for $i5 = i6$ in

$$z_{(i3,i4)} \leq - \underline{z}_{(i3,i4,i5,i6)} \ . \tag{5.50}$$

Fig. 5.4 provides an illustration of constraint (5.44) for a special case, namely $r_{i3} > o_{i5}$, $r_{i6} > o_{i4}$ and $\underline{z}_{(i3,i4,i5,i6)} = 0$: in this case this constraint reflects the fact that two workpieces can never meet between two resources 1 and 2 if one workpiece has to be transferred from resource 1 to resource 2 and the other workpiece has to be transferred in the opposite direction.[6] Fig. 5.4 shows this situation in a Gantt chart: activity $i6$ may only take place before activity $i5$ if activity $i4$ takes place before activity $i3$.

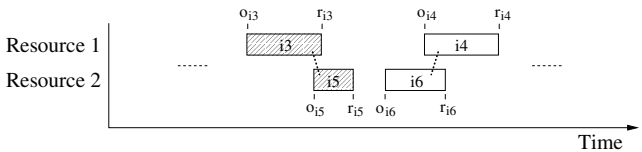

Figure 5.4. An illustration of constraint (5.44).

The cuts (5.44), however, are much more general than illustrated by this special case. The examples in Section 7 will show that these cuts are very efficient: including constraints (5.44), (5.49) and (5.50) in the mathematical program will considerably reduce computation time for the globally optimal solution.

A third type of cuts can be found using graph theory together with max-plus algebra. They read as follows:

$$z_{(i1,i2)} = \sum_{h=i1}^{i2-1} z_{(h,h+1)} + \sum_{h=i1+1}^{i2-1} \hat{z}_{(i1,h,h+1)}, \ \forall i1 < i2, \ i1 = 1, \ldots, n-1, \ i2 = 2, \ldots, n \tag{5.51}$$

[6] If this situation was forced deliberately, it would end up in a deadlock, since any of the two workpieces would be waiting for the other one to release the resource. However, such a situation is already ruled out by the non-blocking specification guaranteed by the disjunctive constraints (see Section 5.2).

with the additional binary variables $\hat{z}_{(i,h,h+1)} \in \{0, 1\}$. For a detailed description of the derivation of these cuts, the reader is referred to [40]. For the example from Section 5.5, the following constraints constitute cuts (5.51):

$$z_{(1,3)} = z_{(1,2)} + z_{(2,3)} + \hat{z}_{(1,2,3)}, \quad \hat{z}_{(1,2,3)} \in \{0, 1\} \tag{5.52}$$

$$z_{(2,4)} = z_{(2,3)} + z_{(3,4)} + \hat{z}_{(2,3,4)}, \quad \hat{z}_{(2,3,4)} \in \{0, 1\} \tag{5.53}$$

$$z_{(1,4)} = z_{(1,2)} + z_{(2,3)} + z_{(3,4)} + \hat{z}_{(1,2,3)} + \hat{z}_{(1,3,4)}, \quad \hat{z}_{(1,3,4)} \in \{0, 1\} \, . \tag{5.54}$$

Using relation (5.25), i. e.

$$z_{(i1,i2)} = -1 - z_{(i2,i1)} \, , \quad i1 \neq i2 \, ,$$

the cuts of type (5.44), (5.49), (5.50) and (5.51) can be included in the mathematical program as inequality constraints of the form

$$\sum_{\iota=1}^{\iota_{max}} bz_{s,\iota} \cdot z_\iota \leq cz_s, \quad s = 1, \ldots, S \, . \tag{5.55}$$

After having added cuts (5.55) to the mixed integer linear program (5.36), a new lower limit T_{min} for the cycle time T can be found from its LP relaxation.[7]

Bounding constraints

In the next step, we will now derive bounding constraints for the time variables θ. These considerations are restricted to the cases in which

$$\kappa_{p,k} \in \{-1, 0, 1\} \, , \quad p = 1, \ldots, P, \quad k = 1, \ldots, K \, . \tag{5.56}$$

The graph modeling algorithm suggested in this chapter will always result in models for which (5.56) holds.

An upper bound for any variable $\hat{\theta}_k$ can be derived from the existing constraints by solving a linear optimization problem (LP) composed of the objective function

$$\text{Min} \; -\hat{\theta}_k$$

and the constraints (5.34c,d) with $T \leq T_{max}$ or alternatively the constraints of the LP relaxation of problem (5.36) $\wedge$ (5.41).

[7] This information may be especially interesting for the user, because it often gives a first idea about the best cycle time that may be possible.

If the problem turns out to be unbounded, i. e. the objective function value goes towards $-\infty$, then there exists no upper bound on $\hat{\theta}_k$ that can be derived from the other constraints. Nevertheless, due to the special properties of the cyclic structure, it is still possible to derive upper bounds on such variables. When doing so, it has to be ensured that these upper bounds do not cut off all globally optimal solutions.

In order to mathematically derive such bounds for the variables θ_k, the mathematical program (5.34) is rewritten skipping the abbreviations (5.27) and (5.30) to (5.33) and using $\hat{\theta} = [\theta^T \; \breve{\theta}^T]^T$, $\hat{\kappa} = [\kappa \; I^{P \times P}]$ and $\hat{K} = K + P$:

Min T over $(T \in \mathbb{R}^+, \hat{\theta}_k \in \mathbb{R}_0^+, z_{(i1,i2)} \in \mathbb{Z})$ subject to

$$z_{(i1,i2)} \cdot T - \left(\chi_{i2,0} - \psi_{i1,0} + \sum_{k=1}^{K} (\chi_{i2,k} - \psi_{i1,k}) \hat{\theta}_k \right) \leq 0 \tag{5.57a}$$

$$\left(z_{(i1,i2)} + 1 \right) \cdot T - \left(\psi_{i2,0} - \chi_{i1,0} + \sum_{k=1}^{K} (\psi_{i2,k} - \chi_{i1,k}) \hat{\theta}_k \right) \geq 0 \tag{5.57b}$$

$$\text{for } \left\{ (i1, i2), \; i2 > i1, \; J_{i1} = J_{i2}, \; i1, i2 \in \{1, \ldots, n\} \right\}$$

$$\sum_{k=1}^{\hat{K}} (\hat{\kappa}_{p,k} \cdot \hat{\theta}_k) = \vartheta_p \quad \text{for } p = 1, \ldots, P \tag{5.57c}$$

$$T \geq \sum_{i=1}^{n} \left(\psi_{i,0} - \chi_{i,0} + \sum_{k=1}^{K} ((\psi_{i,k} - \chi_{i,k}) \hat{\theta}_k) \right) \delta_{J_i j} \quad \text{for } j = 1, \ldots, m \tag{5.57d}$$

Definition 20 (Affine points) *Two points $(\hat{\theta}, T)$ and $(\hat{\theta}', T')$ within the decision space $(\mathbb{R}_+^{\hat{K}}, \mathbb{R}_+)$ of the mixed integer nonlinear program (5.57) are called* affine *if*

$$T' = T \tag{5.58}$$

$$\hat{\theta}' = \hat{\theta} + \xi T \; , \; \xi \in \mathbb{Z}^{\hat{K}} \tag{5.59}$$

$$\sum_{k=1}^{K} (\psi_{i,k} - \chi_{i,k}) \hat{\theta}'_k = \sum_{k=1}^{K} (\psi_{i,k} - \chi_{i,k}) \hat{\theta}_k \; , \quad i = 1, \ldots, n \tag{5.60}$$

Definition 21 (Regular point) *A point $(\hat{\theta}, T)$ within the decision space of the mixed integer nonlinear program (5.57) is called* regular *if it meets constraint (5.57c).*

For two affine points, the single cycles in their cyclic schedules look always identical. The only difference is that activities may belong to different relative batches. Fig. 5.5 illustrates this by showing the Gantt charts for two affine points (in this case both points are also feasible solutions). Clearly, if the first point is an optimal solution with respect to cycle time T, this is also the case for the second point. Therefore, the second point can be ruled out, whereby reducing the size of the feasible region.

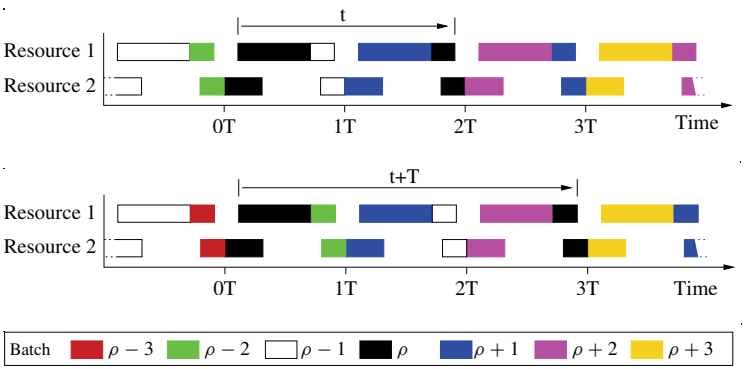

Figure 5.5. Gantt charts for two affine points.

Fact 7 (regular affine point) *If a point $(\hat{\theta}, T)$ is a feasible solution of (5.57), any regular affine point $(\hat{\theta}', T)$ is also a feasible solution.*

Proof Since it is a feasible solution of (5.57), point $(\hat{\theta}, T)$ meets (5.57a,b), i. e. there exist $z_{(i1,i2)} \in \mathbb{Z}$ such that constraints (5.57a,b) hold. An affine point $(\hat{\theta}', T)$ then also meets constraints (5.57a,b): substituting (5.59) in (5.57a) yields

$$z_{(i1,i2)} \cdot T - \left(\chi_{i2,0} - \psi_{i1,0} + \sum_{k=1}^{K} (\chi_{i2,k} - \psi_{i1,k})(\hat{\theta}'_k - \xi_k T) \right) \leq 0$$

or, equivalently,

$$\underbrace{\left(z_{(i1,i2)} - \sum_{k=1}^{K} (\chi_{i2,k} - \psi_{i1,k})\xi_k \right)}_{=:z'_{(i1,i2)}} \cdot T - \left(\chi_{i2,0} - \psi_{i1,0} + \sum_{k=1}^{K} (\chi_{i2,k} - \psi_{i1,k})\hat{\theta}'_k \right) \leq 0 \,.$$

Choosing $z'_{(i1,i2)} = z_{(i1,i2)} + \sum_{k=1}^{K}(\chi_{i2,k} - \psi_{i1,k})\xi_k$, one obtains integer numbers $z'_{(i1,i2)}$ that meet (5.57a). Using (5.59) and Equation (5.60) for $i = \{i1, i2\}$, this yields $z'_{(i1,i2)} = z_{(i1,i2)} + \sum_{k=1}^{K}(\psi_{i2,k} - \chi_{i1,k})\xi_k$ and therefore constraint (5.57b).

Moreover, as $(\hat{\theta}, T)$ is a feasible solution of the entire optimization problem (5.57), Constraint (5.57d) is met for $\hat{\theta}'$ due to Equation (5.60). Since, additionally, $\hat{\theta}'$ is regular, i. e. it meets (5.57c), the affine point $(\hat{\theta}', T)$ meets all constraints (5.57a) to (5.57d) and is therefore a feasible solution of the entire optimization problem (5.57). □

Note that for affine points, their values of the objective function are always identical (as $T = T'$). Therefore, a point $(\hat{\theta}, T)$ can be cut from the feasible region if there exists a regular affine point $(\hat{\theta}', T)$ that is still within the decision space. This will cut off globally optimal solutions but at the same time ensure that at least one globally optimal solution is maintained.

In order to shorten notation, the following abbreviations are used in the following:

$$\phi_{i,k} = \psi_{i,k} - \chi_{i,k},\ k = 0, \ldots, \hat{K}\ . \tag{5.61}$$

A new upper bound $\hat{\theta}_{k^*,max}$ for a variable $\hat{\theta}_{k^*}$ in (5.57) obviously is admissible if for all regular points $(\hat{\theta}, T)$ with $\hat{\theta}_{k^*} > \hat{\theta}_{k^*,max}$ there exists a regular affine point $(\hat{\theta}', T)$ such that $\hat{\theta}'_{k^*} \leq \hat{\theta}_{k^*,max}$:

$$\begin{aligned}
&\forall(\hat{\theta}, T) \text{ with } \hat{\kappa}\hat{\theta} = \vartheta,\ \hat{\theta}_{k^*} > \hat{\theta}_{k^*,max} \\
&\quad \exists\ \xi \in \mathbb{Z}^{\hat{K}} \text{ with } \phi\hat{\theta} = \phi\hat{\theta}',\ \hat{\theta}_k + \xi_k T = \hat{\theta}'_k \geq 0,\ k = 1, \ldots, \hat{K}, \\
&\quad \text{s.th. } \hat{\theta}'_{k^*} \leq \hat{\theta}_{k^*,max} \text{ and } \hat{\kappa}\hat{\theta}' = \vartheta\ .
\end{aligned} \tag{5.62}$$

Using definition (5.59), this is equivalent to

$$\begin{aligned}
&\forall(\hat{\theta}, T) \text{ with } \hat{\kappa}\hat{\theta} = \vartheta,\ \hat{\theta}_{k^*} > \hat{\theta}_{k^*,max} \\
&\quad \exists\ \xi \in \mathbb{Z}^{\hat{K}} \text{ s.th. } \hat{\kappa}\xi = 0, \phi\xi = 0,\ \xi_{k^*} \leq (\hat{\theta}_{k^*,max} - \hat{\theta}_{k^*})/T,\ \hat{\theta} + \xi \cdot T \geq 0.
\end{aligned} \tag{5.63}$$

From this, upper bounding constraints can be derived for almost all variables $\hat{\theta}_{k^*,max}$ that are not yet bounded. Of course, after having decided on an upper bound $\hat{\theta}_{k^*,max}$ for $\hat{\theta}_{k^*}$, this new condition has to be added to the linear equation system $[\hat{\kappa} \cdot \hat{\theta} = \vartheta]$, i. e. (5.57c): the value for $\hat{K}$ has to be incremented by 1, thus introducing a new slack variable $\hat{\theta}_K \geq 0$, and the following line has to be added to (5.57c), thus incrementing P by 1:

$$\hat{\theta}_{k^*} + \hat{\theta}_K = \hat{\theta}_{k^*,max}\ . \tag{5.64}$$

In the following, we present three cases in which such upper bounds can be derived:

1. An upper bound

$$\hat{\theta}_{k^*,max} = T_{max} \tag{5.65}$$

can directly be derived for all k^* with

$$\phi_{i,k^*} = 0, \qquad i = 1, \ldots, n \tag{5.66}$$

$$\hat{\kappa}_{p,k^*} = 0, \qquad p = 1, \ldots, P \ : \tag{5.67}$$

for any regular point $(\hat{\theta}, T)$, we choose

$$\xi_k = \begin{cases} -\lfloor \frac{\hat{\theta}_k}{T} \rfloor & \text{for } k = k^* \\ 0 & \text{for } k \neq k^* \ . \end{cases} \tag{5.68}$$

With $T \leq T_{max}$, it is easy to show that this choice for ξ meets all constraints in (5.63).

2. Additional bounding constraints for variables $\hat{\theta}_k$ can be found, if the linear equation system (5.57c) can be transformed into the following form by linear transformation and/or renumbering of the indices:

$$\left[\begin{array}{c|c|ccc} 1 & 1 & \hat{\kappa}_{1,3} & \cdots & \hat{\kappa}_{1,\hat{K}} \\ \hline 0 & 0 & \hat{\kappa}_{2,3} & \cdots & \hat{\kappa}_{2,\hat{K}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \hat{\kappa}_{P,3} & \cdots & \hat{\kappa}_{P,\hat{K}} \end{array}\right] \hat{\theta} = \left[\begin{array}{c} \vartheta_1 \\ \hline \\ \vdots \\ \vartheta_P \end{array}\right] \tag{5.69}$$

and, additionally, the following holds:

$$\phi_{i,1} = \phi_{i,2} \ , \quad \forall i \in 1, \ldots, n \ .$$

The variable $\hat{\theta}_1$ can then be bounded by adding the additional constraint

$$\hat{\theta}_1 \leq \hat{\theta}_{1,max} = T_{max} \tag{5.70}$$

to the constraints (5.57c) as described in (5.64). It can be shown that with the choice

$$\xi_k = \begin{cases} -\lfloor \frac{\hat{\theta}_1}{T} \rfloor & \text{for } k = 1 \\ +\lfloor \frac{\hat{\theta}_1}{T} \rfloor & \text{for } k = 2 \\ 0 & \text{for } k \geq 3 \end{cases} \tag{5.71}$$

and $T \leq T_{max}$, the constraints in (5.63) are all met.

3. Similarly, a bound can be found, if the linear equation system (5.57c) can be transformed, by linear transformation and/or renumbering of the indices, to

$$\left[\begin{array}{c|c|ccc} -1 & 1 & \hat{\kappa}_{1,3} & \cdots & \hat{\kappa}_{1,\hat{K}} \\ \hline 0 & 0 & \hat{\kappa}_{2,3} & \cdots & \hat{\kappa}_{2,\hat{K}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \hat{\kappa}_{P,3} & \cdots & \hat{\kappa}_{P,\hat{K}} \end{array}\right] \hat{\theta} = \left[\begin{array}{c} \vartheta_1 \\ \hline \\ \vdots \\ \vartheta_P \end{array}\right] \tag{5.72}$$

where

$$\phi_{i,1} = -\phi_{i,2} \ , \quad \forall i \in 1, \ldots, n$$

and the following upper bound is known:

$$\sum_{k=3}^{\hat{K}} \hat{\kappa}_{1,k} \cdot \hat{\theta}_k \leq ub \ . \tag{5.73}$$

The variable $\hat{\theta}_1$ can then be bounded by adding the additional constraint

$$\hat{\theta}_1 \leq \hat{\theta}_{1,max} = \max(T_{max}, -\vartheta_1 + ub + T_{max}) \tag{5.74}$$

as described in (5.64). For (5.72), it can be shown that the choice

$$\xi_k = \begin{cases} -\lfloor \frac{\hat{\theta}_1 - \hat{\theta}_{1,max}}{T} \rfloor - 1 & \text{for } k = 1, 2 \\ 0 & \text{for } k \geq 3 \end{cases} \tag{5.75}$$

together with $T \leq T_{max}$ satisfies the constraints in (5.63).

The proofs for all three cases can be found in Appendix A.

After having derived bounding constraints for all time variables $\theta_k = \hat{\theta}_k$, $k = 1, \ldots, K$, the resulting upper bounds $\theta_{k,max}$, $k = 1, \ldots, K$, can be included in the MILP (5.36) directly by demanding

$$\bar{\theta}_k \leq \theta_{k,max} \cdot \bar{T} \quad \text{for} \quad k = 1, \ldots, K \ . \tag{5.76}$$

Conclusion

Note that the constraint derivation steps in this section can be repeated alternately and iteratively. In this way, stricter constraints may be derived successively. For some scheduling problems, the approach of *constraint propagation*, which is based on this idea, allows to find (optimal) solutions, even without having to solve a mixed integer mathematical

program (e. g. [17, 109]). Nevertheless, we only use the bounds and cuts in order to enhance the mathematical representation of the scheduling problem, which is finally solved by mathematical programming methods.

To sum up, adding the bounding constraints and cuts derived in this section results in the following mixed integer linear program:

Min $-\bar{T}$ subject to

$$z_\iota - v_{\iota,0} \cdot \bar{T} - \sum_{k=1}^{K} (v_{\iota,k} \cdot \bar{\theta}_k) \leq 0 \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{5.77a}$$

$$z_\iota + 1 - w_{\iota,0} \cdot \bar{T} - \sum_{k=1}^{K} (w_{\iota,k} \cdot \bar{\theta}_k) \geq 0 \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{5.77b}$$

$$z_{\iota,min} \leq z_\iota \leq z_{\iota,max} \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{5.77c}$$

$$\sum_{\iota=1}^{\iota_{max}} bz_{s,\iota} \cdot z_\iota \leq cz_s \qquad \text{for } s = 1, \ldots, S \tag{5.77d}$$

$$\bar{\theta}_k \leq \theta_{k,max} \cdot \bar{T} \qquad \text{for } k = 1, \ldots, K \tag{5.77e}$$

$$\sum_{k=1}^{K} (\kappa_{p,k} \cdot \bar{\theta}_k) \leq \vartheta_p \cdot \bar{T} \qquad \text{for } p = 1, \ldots, P \tag{5.77f}$$

$$\frac{1}{T_{max}} \leq \bar{T} \leq \frac{1}{T_{min}} \tag{5.77g}$$

$$\gamma_{j,0} \cdot \bar{T} + \sum_{k=1}^{K} (\gamma_{j,k} \cdot \bar{\theta}_k) - 1 \leq 0 \qquad \text{for } j = 1, \ldots, m \tag{5.77h}$$

Adding the bounding constraints and cuts simplifies the 'search space' for the mathematical program and reduces computation time of the globally optimal solution considerably, paying off for the extra computation time to calculate the bounds and cuts. Of course both, the original mathematical program (5.36) and the mathematical program with additional constraints (5.77) will provide globally optimal solutions with the same objective function value.

Chapter 6

Extensions

In this chapter, several generalizations of the strictly cyclic scheduling problem from Chapter 5 are considered.

6.1 Generalized objective function

The aim of the basic cyclic scheduling problem is to maximize throughput. The objective therefore is to minimize the cycle time T. In practice, however, the aim may be to minimize the makespan, i. e. the time $T_{makespan}$ that is needed to process a fixed number l_b of batches. In general, the makespan depends on the cycle time T, the number of batches l_b and the batch duration T_b (flow time) and can easily be given as

$$T_{makespan} = T \cdot (l_b - 1) + T_b \tag{6.1}$$

with $T_b = x_{e_t} - x_{e_s}$ i. e. T_b equaling the time distance between the event times of the batch start event and the batch terminal event in the single batch time scheme. Using parametrization (4.34) and (4.35) for x_{e_t} and x_{e_s}, the objective of minimizing $T_{makespan}$ results in an objective function of the following form:

$$\text{Min } f_0 T + \sum_{k=1}^{K} f_k \theta_k \ . \tag{6.2}$$

More generally, an objective function of the form (6.2) may be a weighted sum of any event times or time differences. In order to ensure validity of the results in Section 4.2, the corresponding node $\mathcal{Q}[e]$ of any event e whose event time x_e enters the objective

function with negative sign has to be marked as an input node. Correspondingly, the node $\mathcal{Q}[e]$ of any event e whose event time x_e enters the objective function with positive sign has to be marked as an output node.

Substituting the objective function of the optimization problem (5.34) by the new objective function (6.2) results in the following mixed integer nonlinear program:

Min $f_0 T + \sum_{k=1}^{K} f_k \theta_k$ subject to

$$z_\iota \cdot T - v_{\iota,0} - \sum_{k=1}^{K} (v_{\iota,k} \cdot \theta_k) \leq 0 \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{6.3a}$$

$$\Big(z_\iota + 1\Big) \cdot T - w_{\iota,0} - \sum_{k=1}^{K} (w_{\iota,k} \cdot \theta_k) \geq 0 \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{6.3b}$$

$$\sum_{k=1}^{K} (\kappa_{p,k} \cdot \theta_k) \leq \vartheta_p \qquad \text{for } p = 1, \ldots, P \tag{6.3c}$$

$$\gamma_{j,0} + \sum_{k=1}^{K} (\gamma_{j,k} \cdot \theta_k) - T \leq 0 \qquad \text{for } j = 1, \ldots, m \tag{6.3d}$$

Problem (6.3a) to (6.3d) includes two special cases:

1. $f_0 = 1$, $f_k = 0,\ k \geq 1$: This is the basic case solved in Chapter 5.
2. $f_0 = 0,\ T = const$: This special case corresponds to the problem of finding an optimal single batch time scheme if the cycle time is given a-priori (e.g. from the solution of the basic case 1.). Note that the remaining variables θ_k, $k = 1, \ldots, K$ and z_ι, $1, \ldots, \iota_{max}$, together with constraints (6.3a) to (6.3d), form a mixed integer *linear* optimization problem.

For the general case of the optimization problem (6.3a) to (6.3d) a transformation to linear form is not known. Nevertheless, if l_b is sufficiently large, the batch duration becomes less relevant in (6.1) In most cases, a globally optimal solution for the makespan will automatically be found by first solving special case 1 (maximizing throughput) and then solving special case 2 with $T = const$ taken from the solution of special case 1 (minimizing batch duration and therefore finishing last batch "as fast as possible").

6.2 Multi-capacity resources

In some cases, scheduling problems contain resources with a capacity greater than one: either a resource is able to handle more than one activity simultaneously *(multiple servers)* or the system contains several resources of the same type that can be used alternatively. Fig. 6.1 shows an extract from a cyclic schedule for a plant with a multi-capacity resource. The single batch time scheme in this example involves three activities on two single capacity resources Res. 1 and Res. 2. Additionally, there is one resource Res. 3 with multiple capacity $cap_3 = 2$, which is used by two activities of the single batch time scheme. The Gantt chart in Fig. 6.1 clearly shows that the two instances of Res. 3 are used alternatively by the successive batches. Note that, as the optimal allocation of multi-capacity resource instances to batches is not known in advance, the problem cannot simply be solved by introducing two identical single capacity resources. Fig. 6.1 additionally shows that, different to single capacity resources, activities on multi-capacity resources may be *reentrant* i. e. two activities of the same generic type are allowed to overlap.

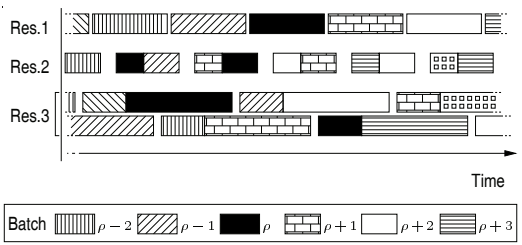

Figure 6.1. Cyclic schedule with a multi-capacity resource.

For such resources, instead of the disjunctive constraints (5.24), different constraints need to be introduced. A set of constraints is needed for each multi-capacity resource to make sure that the upper limit of simultaneous activities ('capacity') is never exceeded. The problem formulation of Sections 4 and 5 can be extended in such a way that also resources with multiple capacity are taken into account [73].

The notation used here is the following: the overall number of multi-capacity resources is denoted by $\tilde{m}$. For these resources, the indices $j = m+1, \ldots, m+\tilde{m}$ are used. For each multi-capacity resource j, the multiplicity, i. e. the maximum number of activities

allowed to allocate the resource simultaneously, is denoted by

$$cap_j \in \{2, 3, 4, \ldots\} \ , \quad j = m + 1 \ , \ldots , \ m + \tilde{m} \ . \tag{6.4}$$

Within the single batch time scheme, there is an overall number of $\tilde{n}$ activities i taking place on resources $J_i \geq m + 1$. For these activities, the indices $i = n + 1 \ , \ldots , \ n + \tilde{n}$ are used. For the example in Fig. 6.1, consequently the following numbers apply: $n = 3$, $m = 2, \tilde{m} = 1, \tilde{n} = 2, cap_3 = 2$.

Each activity i, $i > n$, allocates one entry of the multi-capacity resource J_i. Within the single batch time scheme, it starts at time instant o_i (occupation of one resource entry) and finishes at time instant r_i (release of one resource entry). The duration $r_i - o_i$ of the activity may exceed the cycle time T. For an exact mathematical formulation of the disjunctive constraints, it is necessary to assume that the allocation *includes* time instant o_i and *excludes* time instant r_i.

The load of multi-capacity resource j at time instant t is denoted by

$$load_j(t) \in \mathbb{Z}_0^+ \ . \tag{6.5}$$

The load represents the number of entries of the resource that are allocated at time t. As a basic requirement for the cyclic schedule, at any time, the load must not exceed the capacity of the resource (capacity constraint):

$$load_j(t) \leq cap_j \ \ \forall t \in \mathbb{R}, \ \forall j, \ m + 1 \leq j \leq m + \tilde{m} \ . \tag{6.6}$$

In order to derive an expression for $load_j(t)$, we first determine how much a single activity of the single batch time scheme contributes to the load at time t. Afterwards we just need to sum up these numbers.

At a time instant t during cyclic operation, activity i has *been started* for all batches ρ with

$$o_i^{(\rho)} \leq t. \tag{6.7}$$

Using (5.10a) and (5.10b), as ρ is an integer number, this is true for all

$$\rho < \hat{\rho} = \lfloor \frac{t - o_i}{T} \rfloor + 1 \ . \tag{6.8}$$

On the other hand, at time t, activity i has *not yet been finished* for all ρ with

$$r_i^{(\rho)} > t, \tag{6.9}$$

i. e. for all

$$\rho \geq \check{\rho} = \lfloor \frac{t - r_i}{T} \rfloor + 1. \tag{6.10}$$

From (6.8) and (6.10), it follows that the number of batches, for which activity i is running (i. e. has been started and has not yet been finished) at time instant t is

$$\begin{aligned} act_i(t) &= \hat{\rho} - \check{\rho} && (6.11)\\ &= \lfloor \frac{t - o_i}{T} \rfloor - \lfloor \frac{t - r_i}{T} \rfloor . && (6.12) \end{aligned}$$

This is the number of entries of multi-capacity resource J_i that is allocated by activity i at time instant t.

Consequently, the number of entries of resource j allocated by any activity i with $J_i = j$ at time instant t is

$$\begin{aligned} load_j(t) &= \sum_{i, J_i = j} act_i(t) && (6.13)\\ &= \sum_{i, J_i = j} (\lfloor \frac{t - o_i}{T} \rfloor - \lfloor \frac{t - r_i}{T} \rfloor) . && (6.14) \end{aligned}$$

Since no increase of the load can occur at other times, it is sufficient to ensure that (6.6) holds at the time of all activity starts $o_{i2}^{(\rho)},\ i2 > n,\ J_{i2} = j$. Because of the cyclic mode of operation, this can be furthermore narrowed down to one single batch, e. g. the single batch time scheme, i. e. for

$$t = o_{i2}, \quad n + 1 \leq i2 \leq n + \tilde{n}. \tag{6.15}$$

Substituting (6.14) and (6.15) into (6.6) and using $i = i1$, we get the capacity constraints for multi-capacity resources:

$$\sum_{i1, J_{i1} = J_{i2} = j} (\hat{z}_{(i1,i2)} - \check{z}_{(i1,i2)}) \leq cap_j$$

$$\forall i2, \quad n + 1 \leq i2 \leq n + \tilde{n}. \tag{6.16}$$

with

$$\begin{aligned} \hat{z}_{(i1,i2)} &= \lfloor \frac{o_{i2} - o_{i1}}{T} \rfloor + 1, && (6.17)\\ \check{z}_{(i1,i2)} &= \lfloor \frac{o_{i2} - r_{i1}}{T} \rfloor + 1. && (6.18) \end{aligned}$$

In order to incorporate Equation (6.16) and definitions (6.17), (6.18) into the MILP problem formulation, we need to reformulate (6.17) and (6.18) as the set of inequality constraints

$$\hat{z}_{(i1,i2)} \cdot T - (o_{i2} - o_{i1}) > 0 \tag{6.19}$$

$$(\hat{z}_{(i1,i2)} - 1) \cdot T - (o_{i2} - o_{i1}) \leq 0 \tag{6.20}$$

$$\check{z}_{(i1,i2)} \cdot T - (o_{i2} - r_{i1}) > 0 \tag{6.21}$$

$$(\check{z}_{(i1,i2)} - 1) \cdot T - (o_{i2} - r_{i1}) \leq 0 \tag{6.22}$$

$$\hat{z}_{(i1,i2)},\ \check{z}_{(i1,i2)} \in \mathbb{Z} \tag{6.23}$$

that have to hold for all pairs $(i1, i2)$, with indices $i1, i2 \in \{n+1 \ldots n+\tilde{n}\}$.

Constraint (6.20) can be omitted without any impact on the globally optimal value for the objective function, i. e. the cycle time T: for any value $\hat{z}_{(i1,i2)}$ that violates Constraint (6.20), we can always find a value $\hat{z}'_{(i1,i2)} = \hat{z}_{(i1,i2)} - \xi, 0 < \xi \in \mathbb{Z}$, that still meets Constraint (6.19). This value will also meet (6.16), which is the only other constraint imposed on $\hat{z}_{(i1,i2)}$. Equivalently, Constraint (6.21) can be omitted.

For compact formulation, again, some abbreviations are introduced: Each possible pair of indices $(i1, i2)$, $J_{i1} = J_{i2}, i1, i2 \in \{n+1 \ldots n+\tilde{n}\}$, is denoted by a single number ζ,

$$\zeta = 1, \ldots, \zeta_{max}\ ,\quad \zeta_{max} = \sum_{j=m+1}^{m+\tilde{m}} n_j^2\ . \tag{6.24}$$

Similarly to the definition of ι for single capacity resources (Section 5.3), each value for ζ signifies a pair of activities $(i1, i2)$ within the single batch time scheme using the same resource $J_{i1} = J_{i2} > m$. Note that, whereas each value of ι signifies a pair $(i1, i2)$ with $i1 < i2$, $i1, i2 \in \{1 \ldots n\}$, each value of ζ has to signify a pair $(i1, i2)$, $i1, i2 \in \{n+1 \ldots n+\tilde{n}\}$, including those pairs with $i1 = i2$ and $i1 > i2$. The relationship is stored in $\Upsilon_{i,\zeta}$:

$$\Upsilon_{i,\zeta} = \begin{cases} 1 & \text{if } \zeta \text{ signifies a pair } (i1, i2) \text{ with } i = i2 \\ 0 & \text{otherwise.} \end{cases} \tag{6.25}$$

With the abbreviations $\tilde{v}$ defined equivalently to Equations (5.30), (5.32),

$$\tilde{v}_{\zeta,k} = \chi_{i2,k} - \psi_{i1,k} \tag{6.26}$$

$$\tilde{v}_{\zeta,0} = \chi_{i2,0} - \psi_{i1,0}\ , \tag{6.27}$$

and

$$\tilde{u}_{\zeta,k} = \chi_{i2,k} - \chi_{i1,k} \tag{6.28}$$

$$\tilde{u}_{\zeta,0} = \chi_{i2,0} - \chi_{i1,0} \tag{6.29}$$

for $\zeta = 1, \ldots, \zeta_{max}$, the disjunctive constraints for multi-capacity resources (6.19) and (6.22) finally read:

$$\hat{z}_\zeta \cdot T - \tilde{u}_{\zeta,0} - \sum_{k=1}^{K} \tilde{u}_{\zeta,k} \cdot \theta_k > 0 \tag{6.30}$$

$$(\check{z}_\zeta - 1) \cdot T - \tilde{v}_{\zeta,0} - \sum_{k=1}^{K} \tilde{v}_{\zeta,k} \cdot \theta_k \leq 0 \tag{6.31}$$

$$\hat{z}_\zeta,\ \check{z}_\zeta \in \mathbb{Z}$$

for $\zeta = 1, \ldots, \zeta_{max}$.

These constraints, together with Constraint (6.16), are added to the optimization problem.

Defining $cap_j = 1$, $j = 1, \ldots, m$ (i. e. for the single capacity resources), Constraint (5.6) can be generalized to

$$\gamma_{j,0} + \sum_{k=1}^{K} (\gamma_{j,k} \cdot \theta_k) - T \cdot cap_j \leq 0\ , \quad j = 1, \ldots, m + \tilde{m}\ , \tag{6.32}$$

with $\gamma_{j,0}$ and $\gamma_{j,k}$ as defined in (5.8) and (5.7).

Constraints (6.16) and (6.30) to (6.32) can be transformed into linear form in the same way as the standard problem formulation using transformation (5.35).

The resulting mixed integer linear program that solves the cyclic scheduling problem with multi-capacity resources includes the following variables:

- inverse cycle time $\bar{T} \in \mathbb{R}^+$,
- normalized time variables $\bar{\theta}_k \in \mathbb{R}_0^+,\ k = 1, \ldots, K$,
- integer variables $z_\iota \in \mathbb{Z},\ \iota = 1, \ldots, \iota_{max}$,
- integer variables $\hat{z}_\zeta, \check{z}_\zeta \in \mathbb{Z},\ \zeta = 1, \ldots, \zeta_{max}$.

In summary, the MILP looks as follows:

Min $-\bar{T}$ subject to

$$z_\iota - \ v_{\iota,0} \cdot \bar{T} - \sum_{k=1}^{K} (\ v_{\iota,k} \cdot \bar{\theta}_k) \leq 0 \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{6.33a}$$

$$z_\iota + 1 - w_{\iota,0} \cdot \bar{T} - \sum_{k=1}^{K} (w_{\iota,k} \cdot \bar{\theta}_k) \geq 0 \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{6.33b}$$

$$\sum_{\zeta=1}^{\zeta_{max}} (\hat{z}_\zeta - \check{z}_\zeta) \cdot \Upsilon_{i,\zeta} \leq cap_{J_i} \qquad \text{for } i = n+1, \ldots, n+\tilde{n} \tag{6.33c}$$

$$\hat{z}_\zeta - \tilde{u}_{\zeta,0} \cdot \bar{T} - \sum_{k=1}^{K} \tilde{u}_{\zeta,k} \cdot \bar{\theta}_k > 0 \qquad \text{for } \zeta = 1, \ldots, \zeta_{max} \tag{6.33d}$$

$$\check{z}_\zeta - 1 - \tilde{v}_{\zeta,0} \cdot \bar{T} - \sum_{k=1}^{K} \tilde{v}_{\zeta,k} \cdot \bar{\theta}_k \leq 0 \qquad \text{for } \zeta = 1, \ldots, \zeta_{max} \tag{6.33e}$$

$$\sum_{k=1}^{K} (\kappa_{p,k} \cdot \bar{\theta}_k) \leq \vartheta_p \cdot \bar{T} \qquad \text{for } p = 1, \ldots, P \tag{6.33f}$$

$$\gamma_{j,0} \cdot \bar{T} + \sum_{k=1}^{K} (\gamma_{j,k} \cdot \bar{\theta}_k) - cap_j \leq 0 \qquad \text{for } j = 1, \ldots, m+\tilde{m} \tag{6.33g}$$

Other than finding the optimal cycle time, this method can be used to find out whether (and to which extend) capacities can be reduced without affecting the optimal cycle time (see, e. g. [41, 42, 98]). This problem can be solved in the same framework: in the mixed integer linear program (6.33), the capacities $cap_j \in \mathbb{Z}^+$, $j = m+1 \ldots m+\tilde{m}$ just have to be defined as variables instead of constants. Of course, the capacities may have to be bounded by additional constraints

$$cap_j \leq cap_{max,j} \ , \ \ j = m+1 \ldots m+\tilde{m} \tag{6.34}$$

or, more general, by sum constraints of the form

$$\sum_{j=m+1}^{m+\tilde{m}} bc_j \cdot cap_j \leq cc \ . \tag{6.35}$$

The optimal cycle time under these additional constraints can be found by solving the MILP (6.33) extended by (6.34) and (6.35), where cap_j are defined as variables instead

of constants. In a second step, the cycle time can then be fixed to the optimal value found in the first step. The objective now is to minimize the weighted sum of capacities:

$$\min \sum_{j=m+1}^{m+\tilde{m}} fc_j \cdot cap_j \ , \tag{6.36}$$

where entry fc_j may, e. g., reflect the cost associated to incrementing capacity cap_j by one. The mixed integer linear program obtained from (6.33) $\wedge$ (6.34) $\wedge$ (6.35) after replacing its objective function by (6.36) finally provides the cyclic schedule with globally optimal cycle time and the minimum weighted sum of capacities needed.

6.3 Sequence dependent setup times

A special characteristic exhibited by various scheduling problems are preprocessing times that depend on the sequence in which the activities take place on a resource. An example process that shows this property is the production of paint of different colors in a chemical production plant: if a light color is to be produced after a dark color, an additional cleaning step has to be conducted before adding the light-color ingredients. However, the cleaning step is not necessary, if a dark color follows upon a light one (see, e. g. [81]). A similar effect occurs for a robot moving along a rail: before starting the next workstep, the (idle) robot has to move to the respective position. The time that is needed for this additional operation depends on the position of the robot at the end of the previous activity. Such properties exhibit an identical mathematical problem structure and can be subsumed under the term of *sequence dependent setup times* or *switching times*.

The proposed problem formulation already includes *constant* setup times implicitly: batches allocate two resources simultaneously during transfer. We will now introduce additional pre-setup times that depend on the sequence of events and extend the problem formulation in order to account for this additional requirement.

Consider the following requirement [73]: If two activities $i1$ and $i2$ ($J_{i1} = J_{i2}$) of any batches follow upon each other on the same resource, there is the special demand that the time distance between the end time r_{i1} of activity $i1$ and the start time o_{i2} of activity $i2$ has to be not only non-negative but large enough to allow for the switching time needed for this sorted pair of activities. The minimum time which has to pass between the end of any activity $i1$ and the start of any later activity $i2$ is denoted by

$$sw_{(i2,i1)} \in \mathbb{R}_0^+ \ . \tag{6.37}$$

A specific switching time can be assigned to any pair of activities $(i1, i2)$ that use the same resource $J_{i1} = J_{i2}$. Of course, this time distance has to hold even if other activities take place on the resource between the end of activity $i1$ and the start of activity $i2$. Switching times $sw_{(i2,i1)}$ and $sw_{(i1,i2)}$ may be different. If no switching time is required between two activities $(i1, i2)$, the switching time value is set to zero: $sw_{(i2,i1)} = 0$. Physically meaningful switching times will normally meet the triangular inequality constraint

$$sw_{(i2,i1)} \leq sw_{(i3,i1)} + sw_{(i2,i3)}, \quad i1, i2, i3 \in \{1, \ldots, n\}, \; J_{i1} = J_{i2} = J_{i3} \; . \tag{6.38}$$

The additional requirement of switching times can be easily included into the problem formulation of Section 5. If there were only one pair of activities $(i1, i2)$, these activities would always take place in turns, the switching times could therefore be incorporated into the model just by prolonging the duration of activity $i1$ by the switching time $sw_{(i2,i1)}$ and the duration of activity $i2$ by the switching time $sw_{(i1,i2)}$. Usually, there are more than two activities taking place on the same resource. Hence, this modification must not be done globally but should only come into effect as the disjunctive constraints for the pair of activities $(i1, i2)$ are considered. Consequently, instead of Equations (5.24a) and (5.24b), the following constraints have to be met:

$$z_{(i1,i2)} \cdot T - (o_{i2} - (r_{i1} + sw_{(i2,i1)})) \; \leq \; 0 \tag{6.39}$$

$$\Big(z_{(i1,i2)} + 1\Big) \cdot T - (r_{i2} + sw_{(i1,i2)} - o_{i1}) \; \geq \; 0 \; . \tag{6.40}$$

Hence, the abbreviations (5.32) and (5.33) have to be replaced by

$$v_{\iota,0} \;\; = \;\; \chi_{i2,0} - \psi_{i1,0} - sw_{(i2,i1)} \tag{6.41}$$

$$w_{\iota,0} \;\; = \;\; \psi_{i2,0} - \chi_{i1,0} + sw_{(i1,i2)} \; . \tag{6.42}$$

The inclusion of switching times into the problem formulation therefore *only* requires a change of values for the parameters $v_{\iota,0}$ and $w_{\iota,0}$.

The solution to the optimization problem then yields a globally optimal schedule that meets the constraints for the standard problem as well as the additional switching time requirements.

Additionally, it is possible to find an enhanced lower bound on the cycle time T: with switching times, the definition of $\gamma_{j,0}$ in constraint (5.5) can be replaced by

$$\gamma_{j,0} = \sum_{i=1}^{n} (\psi_{i,0} - \chi_{i,0}) \delta_{J_i j} + swms(j) \; , \tag{6.43}$$

with

$$swms(j) = \min_{\mathcal{W} \in \mathcal{P}(\mathcal{J}_j)} sw_{(\mathcal{W}_1, \mathcal{W}_{|\mathcal{W}|})} + \sum_{k=2}^{|\mathcal{W}|} sw_{(\mathcal{W}_k, \mathcal{W}_{k-1})} \tag{6.44}$$

where $\mathcal{P}(\mathcal{J}_j)$ is the set of all permutations of the set $\mathcal{J}_j$ of indices $\{i,\ J_i = j\}$, $\mathcal{W}$ is one element of this set, and $\mathcal{W}_k$ is the k-th element of $\mathcal{W}$.[1]

6.4 k-periodic schedules

The scheduling problem for strictly cyclic[2] operation can be solved in a globally optimal way by solving the mixed integer linear optimization problem presented in Section 5. However, a strictly cyclic timetable with a constant time offset between all consecutive batches is not always optimal with respect to throughput. For a specific class of cyclic scheduling problems, a two-level hierarchical nesting of cycles allows for a throughput rate higher than in the strictly cyclic case.

Again, the task is to process a (theoretically infinite) number of entities, which all need the same set of worksteps (activities). The process for all entities follows an identical timing. In the hierarchical cyclic framework, we use the term *job* for such an entity. Together with their timing, the corresponding set of activities is called the 'job time scheme'.

For a hierarchical cyclic structure, a finite number of such jobs are grouped together to form one batch. Within a batch, these individual jobs are also processed in a cyclic scheme ('inner cycles'). The batches itself are, again, periodically repeated in a strictly cyclic scheme ('outer cycles'). For the outer cyclic scheme the rules are the same as for the strictly cyclic case described in Section 4:

- The time distance between the start of consecutive batches is always constant.
- The processing time scheme is identical for all batches.
- The cyclic time scheme does allow for infinite repetition.

For the inner cyclic scheme the following rules apply:

[1] Equation (6.44) represents the famous 'traveling salesperson' problem. For large problem instances, it will therefore be advisable to skip this step and to use (5.5) instead, which is a more conservative lower bound on T.

[2] See Section 3.4 for the definition of a strictly cyclic schedule.

- The time distance between the start of consecutive jobs is constant.
- All individual jobs have to be processed in the same time scheme (job time scheme).

A hierarchical cyclic schedule, where k jobs form one batch, is called k-periodic. For a k-periodic schedule, the time offset between an activity for entity x and its counterpart for entity $x + k$ is constant for all entities and all activities. Fig. 6.2 shows a 4-periodic schedule with a cycle time $T = 108$. Since four jobs are finished within each batch, this schedule has an average throughput rate of one job in $108/4 = 27$ time units. The best 1-periodic, i. e. strictly cyclic schedule for the same job time scheme is shown in Fig. 6.3. The cycle time of the strictly cyclic schedule is $T = 36$, i. e. one job is finished every 36 time units. Therefore, in terms of throughput, the 4-periodic schedule is superior to the 1-periodic schedule.

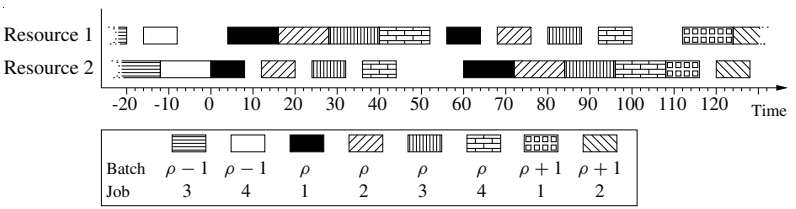

Figure 6.2. A 4-periodic schedule.

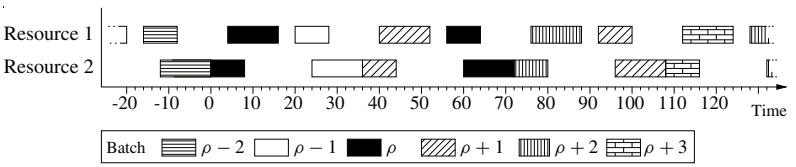

Figure 6.3. Best 1-periodic schedule for the example from Fig. 6.2.

If the number of jobs in the inner cycle, i.e. the number of jobs for one batch is fixed a-priori, the problem is, again, reduced to the strictly cyclic scheduling problem described in Sections 4 and 5: the jobs in the inner cycle can be grouped into one batch, where the parametrization (4.29), together with the linear constraints (4.36) ensure identical time distance between the start of the jobs and identical processing time schemes for all jobs.

However, the optimal number of jobs per batch is usually not known a-priori, but has to be determined simultaneously with the optimal values for the other variables (i.e. the single batch time scheme and the cycle time T). This is done by formulating the scheduling problem for the hierarchical cyclic structure as an optimization problem which, in addition to the timing of the single batch time scheme and the cycle time for the outer cycle, also includes the number of jobs within the single batch time scheme as an integer variable $Y \in \mathbb{N}$.

Batch time scheme

In our hierarchical setting, the set of activities in the single batch time scheme is not fixed but depends on the value of the variable Y.

The time scheme for the individual jobs within the single batch is identical for all jobs. It consists of n^* activities, which are described by their starting times and end times, parametrized by K^* time variables $\theta_k \in \mathbb{R}_0^+$:

$$\begin{aligned} o_i^* &= \chi_{i,0}^* + \sum_{k=1}^{K^*} (\chi_{i,k}^* \cdot \theta_k) \quad \ldots \quad \text{time, when activity } i \text{ starts ,} \\ r_i^* &= \psi_{i,0}^* + \sum_{k=1}^{K^*} (\psi_{i,k}^* \cdot \theta_k) \quad \ldots \quad \text{time, when activity } i \text{ ends .} \end{aligned} \tag{6.45}$$

$$\begin{aligned} r_i^* &> o_i^* \\ i &= 1, \ldots, n^* \, . \end{aligned} \tag{6.46}$$

Each activity i allocates one resource $J_i^* \in \{1, \ldots, m\}$.

Again, user-defined constraints ensure that the activity durations allow for finishing the required operations and that the sequence and time window constraints for the individual jobs within the job time scheme hold. These constraints are represented by P linear constraints of the form

$$\sum_{k=1}^{K^*} (\kappa_{p,k}^* \cdot \theta_k) \leq \vartheta_p^* \, , \quad p = 1, \ldots, P \, . \tag{6.47}$$

The parametrization of the job time scheme $\chi_{i,0}^*$, $\chi_{i,k}^*$, $\psi_{i,0}^*$, $\psi_{i,k}^*$, as well as the additional constraints defined by $\kappa_{p,k}^*$ and ϑ_p^* can be adopted from the definition of the problem instance in the same way as described for strictly cyclic schedules in Chapter 4. It is

furthermore assumed that an upper bound Y_{max} for the variable Y is prescribed by the user.

Thus, the single batch time scheme consists of at most Y_{max} jobs and n activities, where $n = Y_{max} \cdot n^*$. Within the batch, jobs are started cyclically with cycle time T^*. As all jobs follow the time scheme given in (6.45), the activity start and end times for job $h \in \{1, \ldots, Y_{max}\}$ are

$$\begin{aligned} o_i^{*(h)} &= \chi_{i,0}^* + \sum_{k=1}^{K^*} (\chi_{i,k}^* \cdot \theta_k) + (h-1) \cdot T^* \quad \text{for } i = 1, \ldots, n^* \\ r_i^{*(h)} &= \psi_{i,0}^* + \sum_{k=1}^{K^*} (\psi_{i,k}^* \cdot \theta_k) + (h-1) \cdot T^* \quad \text{for } i = 1, \ldots, n^*. \end{aligned} \tag{6.48}$$

As the cycle time of the inner cycle T^* is a variable that is part of the single batch time scheme, notation $\theta_K = \theta_{K^*+1} := T^*$ will be used. Therefore the constraints for the variables $\theta_k, k = 1, \ldots, K$, are

$$\sum_{k=1}^{K} (\kappa_{p,k} \cdot \theta_k) \leq \vartheta_p \ , \ \ p = 1, \ldots, P \ , \tag{6.49}$$

where

$$\kappa_{p,k} = \kappa_{p,k}^* \ , \quad p = 1, \ldots, P \ , \ k = 1, \ldots, K^* \tag{6.50}$$

$$\kappa_{p,k} = 0 \ , \quad p = 1, \ldots, P \ , \ k = K \tag{6.51}$$

$$\vartheta_p = \vartheta_p^* \ , \quad p = 1, \ldots, P \ . \tag{6.52}$$

To simplify notation, the activities of all Y_{max} jobs are subsumed in one set and are serially numbered by a common index i. Thus, the single batch time scheme for the hierarchical cyclic structure is defined as follows:

$$\begin{aligned} o_i &= \chi_{i,0} + \sum_{k=1}^{K} (\chi_{i,k} \cdot \theta_k) \ , \quad i = 1, \ldots, n \\ r_i &= \psi_{i,0} + \sum_{k=1}^{K} (\psi_{i,k} \cdot \theta_k) \ , \quad i = 1, \ldots, n \ , \end{aligned} \tag{6.53}$$

$$
\begin{aligned}
\psi_{i,0} &= \psi^*_{i^*,0}\ , \quad i = 1, \ldots, n \\
\chi_{i,0} &= \chi^*_{i^*,0}\ , \quad i = 1, \ldots, n \\
\psi_{i,k} &= \psi^*_{i^*,k}\ , \quad i = 1, \ldots, n;\ , \quad k = 1, \ldots, K^* \\
\psi_{i,k} &= \lfloor (i-1) \,/\, n^* \rfloor\ , \quad i = 1, \ldots, n\ , \quad k = K = K^* + 1 \\
\chi_{i,k} &= \chi^*_{i^*,k}\ , \quad i = 1, \ldots, n;\ , \quad k = 1, \ldots, K^* \\
\chi_{i,k} &= \lfloor (i-1) \,/\, n^* \rfloor\ , \quad i = 1, \ldots, n\ , \quad k = K = K^* + 1
\end{aligned}
$$

where $n = n^* \cdot Y_{max}$ and $i^* = (i-1) \bmod n^* + 1$.

The resources allocated by the activities are

$$J_i = J^*_i\ , \quad i = 1, \ldots, n\ . \tag{6.54}$$

As only the first Y jobs are part of the single batch time scheme, only activities i, $y_i = \lfloor (i-1) \,/\, n^* \rfloor + 1 \leq Y$ take place:

$Y \geq y_i$ ... activity i is part of the single batch time scheme (='effective')

$Y < y_i$ ... activity i is not part of the single batch time scheme.

Constraints

Since the outer cyclic structure is identical to the standard strictly cyclic case described in Section 4.1, the constraints for the optimization problem are derived in the same way. However, the set of activities in the single batch time scheme is not fixed but depends on the number of jobs per batch (Y). The set of constraints for the optimization problem therefore depends on the value of the variable Y.

First, a lower bound for the cycle time T can be set up equivalently to Constraint (5.5). For the hierarchical cyclic case, this constraint needs to account for the number of effective activities in the single batch time scheme: for a certain value of Y, only activities i that are effective (i. e. $y_i \leq Y$) contribute to the sum of activity durations in the single batch time scheme. Hence, a lower bound for the cycle time T is

$$T \geq \sum_{i=1}^{n} (r_i - o_i) \delta_{J_i j} \Delta_{y_i Y}\ , \quad j = 1, \ldots, m \tag{6.55}$$

$$\text{where } \Delta_{y_i Y} = \begin{cases} 0 & \text{for } y_i > Y \\ 1 & \text{for } y_i \leq Y\ . \end{cases} \tag{6.56}$$

Substituting (4.29) into (6.55) results in

$$\gamma_{j,0,Y} + \sum_{k=1}^{K} (\gamma_{j,k,Y} \cdot \theta_k) - T \leq 0 \ , \quad j = 1, \ldots, m \ , \tag{6.57}$$

where

$$\begin{aligned} \gamma_{j,0,Y} &= \sum_{i=1}^{n} (\psi_{i,0} - \chi_{i,0}) \delta_{J_i j} \Delta_{y_i Y} \\ \gamma_{j,k,Y} &= \sum_{i=1}^{n} (\psi_{i,k} - \chi_{i,k}) \delta_{J_i j} \Delta_{y_i Y} \ . \end{aligned} \tag{6.58}$$

This set of constraints depends on the value of the variable Y. In order to have a fixed number of constraints, (6.57) can be transformed into the following set of 'OR' statements:

$$\gamma_{j,0,Y'} + \sum_{k=1}^{K} (\gamma_{j,k,Y'} \cdot \theta_k) - T \leq 0 \quad \text{OR} \quad Y \neq Y' \tag{6.59}$$
$$\text{for } j = 1, \ldots, m \ , Y' = 1, \ldots, Y_{max} \ .$$

Due to the definition of $\gamma_{j,0,Y}$ and $\gamma_{j,k,Y}$, (6.58), (6.53) together with (6.46), the following always holds for $Y1 \geq Y2$:

$$\gamma_{j,0,Y1} + \sum_{k=1}^{K} (\gamma_{j,k,Y1} \cdot \theta_k) \geq \gamma_{j,0,Y2} + \sum_{k=1}^{K} (\gamma_{j,k,Y2} \cdot \theta_k) \ . \tag{6.60}$$

Therefore, Equation (6.59) can be finally given as follows:

$$\gamma_{j,0,Y'} + \sum_{k=1}^{K} (\gamma_{j,k,Y'} \cdot \theta_k) - T \leq 0 \quad \text{OR} \quad Y \leq Y' - 1 \tag{6.61}$$
$$\text{for } j = 1, \ldots, m \ , Y' = 1, \ldots, Y_{max} \ .$$

The second set of constraints for the optimization problem is provided by the linear constraints on the variables θ_k, $k = 1, \ldots, K^*$, given in Equation (6.49).

The third type of constraints are the disjunctive constraints (5.9) that have to be met for all pairs of effective activities belonging to the single batch time scheme. Therefore, Equations (5.24a) and (5.24b) have to hold for all pairs $(i1, i2)$ for which

$$y_{i1} \leq Y \ \text{ and } \ y_{i2} \leq Y \ . \tag{6.62}$$

Accordingly, Constraints (5.28), (5.29) have to come into effect if (6.62) holds for the indices $i1$, $i2$ associated to the index ι. A variable g_ι is used to formulate this condition: constraints (5.28), (5.29) have to hold if

$$Y > g_\iota \,, \tag{6.63}$$

$$\text{with } g_\iota = \max(y_{i1}, y_{i2}) - 1 \,. \tag{6.64}$$

In other words,

$$z_\iota \cdot T - v_{\iota,0} - \sum_{k=1}^{K} (\, v_{\iota,k} \cdot \theta_k) \leq 0 \quad \text{OR} \quad Y \leq g_\iota \,,\ \iota = 1, \ldots, \iota_{max} \tag{6.65}$$

$$\Big(z_\iota + 1 \Big) \cdot T - w_{\iota,0} - \sum_{k=1}^{K} (w_{\iota,k} \cdot \theta_k) \geq 0 \quad \text{OR} \quad Y \leq g_\iota \,,\ \iota = 1, \ldots, \iota_{max} \,, \tag{6.66}$$

where ι_{max} has to be determined for the case $Y = Y_{max}$.

Optimization problem

The objective is to process a maximum number of jobs per time. As Y jobs are grouped into one batch, the objective function for throughput maximization is to minimize the cycle time of the outer cycles divided by the number of jobs per batch:

$$\text{minimize } \frac{T}{Y} \,.$$

This term represents the inverse of the job throughput, i. e. the mean time offset between consecutive jobs.

This function has to be minimized subject to constraints (6.49), (6.61), (6.65), and (6.66).

Some lower and upper bounds for the cycle time T as well as for the integer variables z_ι can be included in the optimization problem according to Section 5.6.

To sum up, the optimization problem that describes the cyclic scheduling problem with hierarchical cyclic structure incorporates the variables

$$(T \in \mathbb{R}^+,\ \theta_k \in \mathbb{R}_0^+,\ z_\iota \in \mathbb{Z},\ Y \in \mathbb{N}, Y > 0)$$

and reads as follows:

Min $\frac{T}{Y}$ subject to

$$z_\iota \cdot T - \ v_{\iota,0} - \sum_{k=1}^{K}(\ v_{\iota,k} \cdot \theta_k) \leq 0 \quad \text{OR} \quad Y \leq g_\iota \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{6.67a}$$

$$\Big(z_\iota + 1\Big) \cdot T - w_{\iota,0} - \sum_{k=1}^{K}(w_{\iota,k} \cdot \theta_k) \geq 0 \quad \text{OR} \quad Y \leq g_\iota \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{6.67b}$$

$$\sum_{k=1}^{K}(\kappa_{p,k} \cdot \theta_k) \leq \vartheta_p \qquad \text{for } p = 1, \ldots, P \tag{6.67c}$$

$$T_{min} \leq T \leq T_{max} \tag{6.67d}$$

$$\gamma_{j,0,Y'} + \sum_{k=1}^{K}(\gamma_{j,k,Y'} \cdot \theta_k) - T \leq 0 \quad \text{OR} \quad Y \leq Y' - 1 \tag{6.67e}$$

$$\text{for } j = 1, \ldots, m \quad , \quad Y' = 1, \ldots, Y_{max}$$

$$Y \leq Y_{max} \tag{6.67f}$$

The mixed integer nonlinear program (6.67) can again be translated into linear form using transformation (5.35) and the following ideas from mathematical programming:

1. An objective function of the form

   $$\min \ -a \cdot Y \ ,$$

   where $Y \leq Y_{max}$ is a positive integer, can be transformed into the objective function

   $$\min \ -\sum_{h=1}^{Y_{max}} a_h$$

   and the following set of constraints for all $h = 1, \ldots, Y_{max}$:

   $$a_h \leq a$$

   $$a_h \leq 0 \quad \text{OR} \quad h \leq Y \ .$$

2. a constraint of the form

   $$a \leq b \quad \text{OR} \quad c \leq d$$

   can be translated into

   $$-Mx + a \leq b \quad \text{AND} \quad -M(1 - x) + (c - d) \leq 0$$

   with the binary variable $x \in \{0, 1\}$ and a sufficiently large number M [2, 43].

The transformation is done as follows: first, we apply transformation (5.35) on the mixed integer nonlinear program (6.67). This results in the objective function

$$\text{Min } \frac{1}{\bar{T} \cdot Y} \ ,$$

which can be replaced by

$$\text{Min } \ -\bar{T} \cdot Y \ .$$

According to idea 1, this is equivalent to

$$\text{min } \ -\sum_{h=1}^{Y_{max}} a_h$$

and

$$a_h \leq 0 \quad \text{OR} \quad h \leq y \ , \ a_h \leq a \ , \ h = 1 \dots Y_{max} \ .$$

Using idea 2, the latter can be transformed into

$$-M\tilde{x}_h + a_h \leq 0 \quad \text{AND} \quad -M(1 - \tilde{x}_h) + (h - Y) \leq 0 \ , \ h = 1 \dots Y_{max}$$

with the large number M and the additional binary variables $\tilde{x}_h \in \{0, 1\}$, $h = 1 \dots Y_{max}$. After equivalently transforming Constraints (6.67a), (6.67b) and (6.67e) using transformation (5.35) and idea 2, we get a linear form of (6.67), which incorporates the variables

$$\begin{aligned}
\bar{T} &\in \mathbb{R}^+, \\
\bar{\theta}_k &\in \mathbb{R}_0^+, && k = 1, \dots, K, \\
z_\iota &\in \mathbb{Z}, && \iota = 1, \dots, \iota_{max}, \\
x_\iota &\in \{0, 1\}, && \iota = 1, \dots, \iota_{max}, \\
\tilde{x}_h &\in \{0, 1\}, && h = 1, \dots, Y_{max}, \\
\hat{x}_{Y'} &\in \{0, 1\}, && Y' = 1, \dots, Y_{max}, \\
Y &\in \mathbb{N}, \\
a_h &\in \mathbb{R}_0^+, && h = 1, \dots, Y_{max} \ .
\end{aligned}$$

and reads as follows:

$$\text{Min} - \sum_{h=1}^{Y_{max}} a_h \text{ subject to}$$

$$-M\tilde{x}_h + a_h \leq 0 \qquad \text{for } h = 1, \ldots, Y_{max} \tag{6.68a}$$

$$a_h \leq \bar{T} \qquad \text{for } h = 1, \ldots, Y_{max} \tag{6.68b}$$

$$-M(1 - \tilde{x}_h) + (h - Y) \leq 0 \qquad \text{for } h = 1, \ldots, Y_{max} \tag{6.68c}$$

$$-M \cdot x_\iota + z_\iota - v_{\iota,0} \cdot \bar{T} - \sum_{k=1}^{K} v_{\iota,k} \cdot \bar{\theta}_k \leq 0 \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{6.68d}$$

$$M \cdot x_\iota + z_\iota + 1 - w_{\iota,0} \cdot \bar{T} - \sum_{k=1}^{K} w_{\iota,k} \cdot \bar{\theta}_k \geq 0 \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{6.68e}$$

$$-M \cdot (1 - x_\iota) + Y - g_\iota \leq 0 \qquad \text{for } \iota = 1, \ldots, \iota_{max} \tag{6.68f}$$

$$\sum_{k=1}^{K} (\kappa_{p,k} \cdot \bar{\theta}_k) \leq \vartheta_p \cdot \bar{T} \qquad \text{for } p = 1, \ldots, P \tag{6.68g}$$

$$\frac{1}{T_{max}} \leq \bar{T} \leq \frac{1}{T_{min}} \tag{6.68h}$$

$$Y \leq Y_{max} \tag{6.68i}$$

$$-M \cdot \hat{x}_{Y'} + \gamma_{j,0,Y'} \cdot \bar{T} + \sum_{k=1}^{K} (\gamma_{j,k,Y'} \cdot \bar{\theta}_k) - 1 \leq 0 \tag{6.68j}$$

$$\text{for } j = 1, \ldots, m, \qquad Y' = 1, \ldots, Y_{max}$$

$$-M \cdot \hat{x}_{Y'} + Y - (Y' - 1) \leq 0 \tag{6.68k}$$

$$\text{for } j = 1, \ldots, m, \qquad Y' = 1, \ldots, Y_{max}$$

The solution to the mixed integer linear program (6.68) provides the hierarchical cyclic schedule with optimal throughput and accordingly chosen periodicity. An example for the solution of a k-periodic scheduling problem can be found in [76].

Of course, this method is not restricted to a cyclic batch time scheme. The following further generalizations can be made:

- the time distance between the start of consecutive jobs within the single batch time scheme may vary, and/or
- the jobs in the single batch time scheme may have different time schemes. This also handles problems that include *pooling conditions*: in pooling, a number of jobs undertake a specific activity conjointly[3] [54].

In order to incorporate these generalizations into the MINLP (6.67), the parametrization (6.53) has to be changed accordingly.

[3] Consider, e. g., the oven in a bakery.

Chapter 7

Application

The framework for modeling and solution of the cyclic scheduling problem developed in this thesis can be applied to almost all kinds of cyclically operated systems that can be described by discrete event models. Some examples are

- various types of cyclically operated plants, e. g. robotic work-cells, flexible manufacturing systems or automated laboratories as high throughput screening plants, as well as cyclically operated machines, e. g. high throughput duplex printers and copy machines.
- cyclic systems in traffic and transportation, e. g. railway systems, synchronized rail timetables, phased traffic lights,
- distributed control systems, where sensors, actors and controllers act repetitively with a common cycle time and communicate via bus systems.
- cyclically operated batch plants in chemical processing, where throughput improvements often result in substantial savings [109].

In this chapter, we present three different fields of application. For each application area, the scheduling problem is illustrated by examples. Data about model sizes and computation times is given for a number of problem instances.

In order to solve problem instances of all types, extensive matlab software code [28] has been developed that allows to conduct the following steps:

- data import

- model reduction (Section 4.2)
- establishing disjunctive constraints (Section 5.2)
- establishing bounding constraints and additional cuts (Section 5.6)
- MILP solution (establishing minimum cycle time T, Sections 5.4 and 5.6)
- MILP solution (minimizing batch duration, Section 6.1)
- data export and visualization.

The software also takes into account all extensions from Chapter 6.

The mixed integer linear programs are solved on a standard PC with the CPLEX solver [56], which is the most widely used software for mixed integer linear programming. CPLEX is used under the modeling environment GAMS [39]. The LP relaxations and LPs are computed using CPLEX's LP solver.

7.1 High throughput screening

High throughput screening (HTS) has become a standard technology for the discovery of drugs in pharmaceutical industries. A high throughput screening plant allows to analyze several hundreds of thousands of substance samples in a fully automated way [32]. The HTS scheduling problem is a relatively new field of research [11, 31, 47]. Several specialized methods are available for laboratory automation [7, 33, 36, 86], but no specific research has been done on structural approaches to find schedules for strictly cyclic operation.

In high throughput screening, a batch subsumes all worksteps that are necessary to analyze one set of substances. Such a set consists of up to 1536 substances, which are aggregated on one *microplate*. Additional microplates may be included in the batch to convey reagents or waste material. A problem instance in high throughput screening is called *assay*. It consists of a limited or unlimited number of batches and the definition of the worksteps resp. activities that have to be performed on all batches in the same way. The latter is the single batch time scheme definition, which, in high throughput screening, is also called *protocol*. The microplates constitute the workpieces of HTS problems.

An HTS plant involves a fixed set of resources, which can be distinguished into components (stations) and transportation devices. Components perform liquid handling, storage,

reading, plate handling and incubation steps. Transportation devices may be longitudinal or rotational movers or robot arms.

The complexity of the assay, i. e. the size of the single batch time scheme in terms of activities reaches from simple AIR (Add, Incubate, Read) assays, which will look similar to the example problem in Sections 4.3 and 5.5, up to arbitrary complex assays where a large number of activities have to be performed for each single batch. The size of the underlying automation system may reach from specialized workstations to large, integrated robotic systems [13, 47]. Generally, the trend in laboratory automation goes towards flexible systems that combine components for different tasks and link them by several transportation devices. Such machines are fully automated and allow to conduct activities for several microplates resp. batches at the same time.

The HTS scheduling problem is to find an optimal schedule for a given problem instance definition, i. e. a given assay. The problem is solved offline before the plant is started. The objective of HTS scheduling normally is to minimize the makespan by maximizing the sample throughput [31]. The degrees of freedom in the timing of the single batch have to be used in such a way that a schedule with minimum cycle time T becomes possible. This task is often conducted manually using expert knowledge and experience. As new technologies get available, the number of potential assay permutations increases [77]. This means that there is not one 'HTS single batch time scheme', but a new instance of the cyclic scheduling problem has to be solved for each single batch time scheme the user wants to run. Of course, a *manual solution* will normally not result in the global optimum and no information will be available about the distance between the solution found so far and the potential global optimum.

In contrast, investigating the scheduling problem by systematic mathematical algorithms allows to

- generate schedules that are proven to be globally optimal with respect to throughput.

In addition, it allows to

- find optimal arrangements of components and transportation devices,
- find an optimal allocation of worksteps to machines,
- determine bottlenecks in the system.

The latter three steps can be done by comparing different options with respect to their optimal throughput and can be performed in a fraction of the time that would be needed for manual try-and-error.

The HTS scheduling problem imposes a unique set of constraints:

- all batches have to follow exactly the same time scheme (to ensure comparability of the measurement results),
- the same resource may be revisited several times by the same batch ($\rightsquigarrow$ no job shop),
- apart from the incubation, there is no in-process storage, i. e. there are no unlimited buffers between the resources of the plant ($\rightsquigarrow$ non-blocking specification needed),
- time window constraints are imposed by the user (e. g., some reagents will need to be added 'just in time'),
- A batch may involve more than one plate in parallel (*multi-plate assay* $\rightsquigarrow$ precedence network needed).
- sometimes a 'good' solution (single batch time scheme) is known a-priori from experience.

Tightening the first constraint, we consider strictly cyclic schedules, where the time distance between the start of consecutive batches is always constant. This is an additional constraint that is imposed deliberately. Nevertheless, such a regular timing may be favored by the user because it results in constant rest periods for the substances that are added to each sample.

Example 1: ELISA

We will now consider a number of real problem instances for high throughput screening. A typical example for an HTS process is given by the ELISA test. An ELISA assay typically involves two or more microplates for each sample, i. e. for each batch. We use a standard ELISA example process from [86]. The general precedence structure for this example process is sketched in Fig. 7.1. It clearly shows that the process involves two microplates for each batch: an assay plate and a compound plate. Both microplates have to be synchronized at a specific dispense activity. During the process, the assay plate is

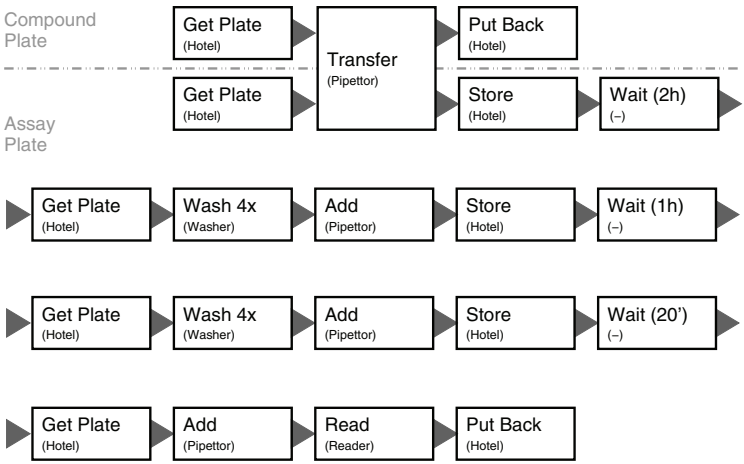

Figure 7.1. Precedence structure for the ELISA assay.

put into the 'Hotel' resource for incubation resp. reaction. It can be assumed that this resource has a sufficient number of places to store microplates, as all microplates are situated in one of the places of the hotel when the process starts. Therefore, only the handling mechanics of the hotel machine are considered as the 'Hotel' resource, which has capacity one. The complete time window precedence network for all events of the single batch time scheme of the ELISA assay is given in Appendix C. The weights of the timed dependencies between events are normally investigated by test runs on the real plant (*teach-in*), the bounds for the time window constraints are defined by the user. A compact illustration of the ELISA assay is provided by the Gantt chart for the single batch time scheme in Fig. 7.2. This example involves only one transport unit: a robot is used to move the microplates between all activities on the machine resources. The time the empty robot needs to move to the next resource after having transferred a plate to its destination is neglected.

The single batch time scheme consists of four distinct segments. It is assumed that there are no upper bounding constraints on the time distances between these segments. A first step to simplify the problem is therefore to divide the single batch time scheme into four distinct jobs: each segment is assigned to a separate job. This is possible, because, for infinite cyclic repetition, the four segments of the single batch time scheme can later be reassigned to physical microplates in such a way that the minimum time distances

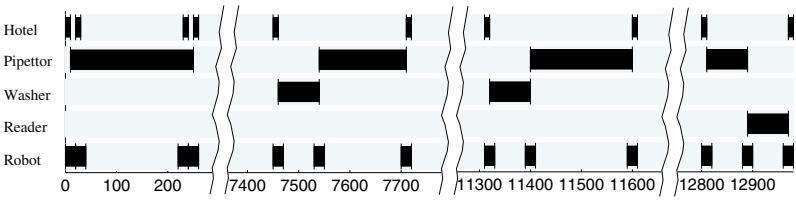

Figure 7.2. Single batch time scheme for the ELISA assay.

demanded by the user are kept.[1]

Apart from choosing relative timing of the segments, i. e. the jobs, all activities may be arbitrarily prolonged by inserting an idle time before the time instant at which plate leaves the resource. Altogether, these degrees of freedom are used to generate a single batch time scheme that allows for a minimum cycle time.

Following the modeling process of Chapters 4 and 5, a mathematical program can be established. The basic MILP consists of 28 continuous and 130 integer variables and includes 321 constraints. The solution of the mathematical program provides a strictly cyclic schedule with globally optimal throughput. During the modeling process, an upper bound $T_{max,3} = 895$ for T can be found by solving the problem for the fixed single batch time scheme from Fig. 7.2, i. e. without allowing for any change in the timing. A lower bound $T_{min} = 670$ can be established by solving the relaxed MILP.

After adding bounding constraints for the variables and cuts as described in Section 5.6, a globally optimal solution to the mathematical program is found within 0.12 seconds. The optimal cycle time results to be $T_{opt} = 770$ and therefore allows for an increase in throughput of $16,2\%$ compared to the solution known from the upper bound $T_{max,3} = 895$. Information about the size of the mathematical program model as well as computation times are given in the tables below.

Fig. 7.3 shows a part (a time period) from the optimal strictly cyclic schedule for the ELISA assay. Four jobs are singled out by different colors. Of course, the colored jobs do not belong to the same batch. From the cyclic process, four jobs (one of each type) are assigned to the first batch in such a way that the minimum incubation times are obeyed. From that, the other job-to-batch assignments follow from strict cyclicity. Fig. 7.4 shows the cyclic process for the first 20 batches by combining all resources into one line and distributing batches along the ordinate instead. The figure shows that one batch after

[1]The division step would even be possible if the segments were connected by time window constraints that allow for an interval larger than the maximum expected cycle time.

the other starts with the first job and passes through the other three jobs, interrupted by incubation periods.

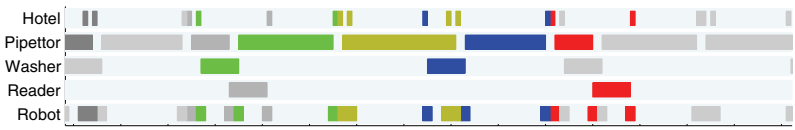

Figure 7.3. Part (time period) from the cyclic schedule for the ELISA assay.

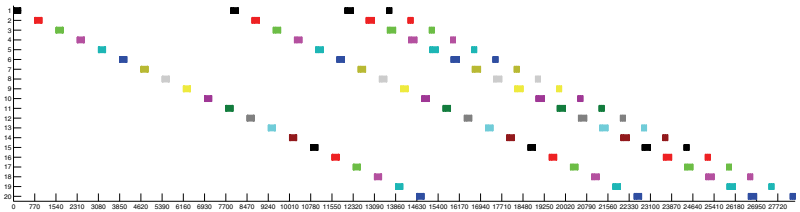

Figure 7.4. Graphical representation of a cyclic schedule for 20 batches.

In a non-cyclic solution to this assay from the literature [86], the makespan for 20 batches is 10:10 hours. For the globally optimal strictly cyclic solution, the makespan is less than 8 hours. Of course, the artificial restriction on *strictly cyclic schedules* will never result in a makespan better than the *globally optimal* makespan for the unrestricted case. The non-cyclic solution is therefore obviously not the globally optimal solution but a suboptimal one. At the same time, this illustrates the following: determining globally optimal schedules with the restriction on strictly cyclic solutions may often be superior to the search of a 'good' solution in the non-restricted search space.

Example 2: HTS f

As a second example, we consider an HTS assay for which transportation tasks are carried out by different transportation devices. This assay is inspired from practical application problems at CyBio, a manufacturer of large HTS plants. The time scheme for the single batch consists of 20 activities on an overall number of 9 resources. It is illustrated as a Gantt chart in Fig. 7.5. This assay involves two incubation phases. The first incubation

phase is defined by the time interval t_{incub1} between the two activities marked with an asterisk in Fig. 7.5. The following time window constraints have to hold for this time interval:

$$710 \leq t_{incub1} \leq 800 \ .$$

This incubation phase takes place on Component 1, which is assumed to have a limited number of storage places for microplates. Component 1 is therefore modeled as a multi-capacity resource as described in Section 6.2 with capacity yet to be determined. Different to the ELISA example, where microplates do not allocate any resources during incubation periods, the incubation process is now explicitely modeled as an activity that takes place on this multi-capacity resource.

In a first step, the resource is assumed to have unlimited (i. e. sufficiently high) capacity.

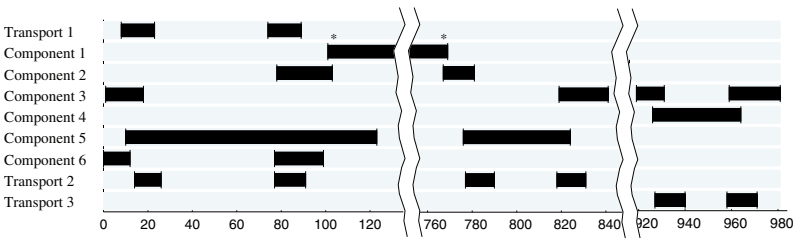

Figure 7.5. The single batch time scheme of the 'HTS f' assay (Gantt chart).

The corresponding cyclic scheduling problem results in a mixed integer linear program that contains 19 real variables and 19 integer variables. The optimal cycle time is $T_{opt} = 210$.

A subsequent step allows to determine the minimum capacity needed for Component 1: this number is found by solving the MILP again, this time fixing the cycle time to the optimal value of 210 and minimizing the value of the capacity instead (see Section 6.2). It turns out that the optimal cycle time $T_{opt} = 210$ can be achieved by providing a capacity of 4 for Component 1.

Computational Case studies

In order to prove applicability of strictly cyclic scheduling to high throughput screening, globally optimal schedules have been computed for a large number of HTS problem in-

stances. In the following, we provide some data about model sizes and computation times for the following problem instances.

- 'EXAMPLE1': the small example problem from Section 5.5, transformed into MILP form (5.34) and including 14 cuts according to Section 5.6.
- 'ELISA': the assay discussed in this section (Example 1)
- 'HTS f': HTS assay discussed in this section (Example 2)
- 'HTS 1' to 'HTS 9': real assays from the HTS industries[2]
- 'HTS XL', 'HTS XXL': two fictive HTS assays which are composed of two assays (HTS XL), resp. three assays (HTS XXL) (simultaneous conduction of multiple assays with the same cycle time) and show the applicability of the method to larger problem instances.

The following numbers about problem sizes are given in Table 7.1:

- the number of resources m,
- the number of activities in the single batch time scheme n,
- the distribution of the activities on the individual resources n_j,
- the number of arcs $|\mathcal{E}_a|$ in the original time window precedence network $\mathcal{G}_a$ (see page 35),
- the number of arcs $|\mathcal{E}_c|$ in the reduced time window precedence network (see page 40),
- the number K of variables θ needed to parametrize the degrees of freedom in the single batch time scheme,
- the number ι_{max} of integer variables needed to state the disjunctive constraints for single-capacity resources, and
- the number ζ_{max} of integer variable pairs needed to state the disjunctive constraints for multi-capacity resources.

[2] CyBio AG.

Table 7.1. HTS examples: problem size.

Name	m	n	n_j	$\lvert\mathcal{E}_a\rvert$	$\lvert\mathcal{E}_c\rvert$	K	ι_{max}	ζ_{max}
EXAMPLE1	3	6	[1 1 4]	36	4	3	6	-
ELISA	5	30	[10 4 2 1 13]	229	29	27	130	-
HTS f	9	20	[2 1 2 4 1 2 2 4 2]	158	22	18	17	1
HTS 1	10	25	(up to 7)	4493	30	22	36	-
HTS 2	11	27	(up to 5)	4883	35	24	29	-
HTS 3	5	18	(up to 8)	2174	22	17	40	-
HTS 4	6	22	(up to 9)	2989	31	21	50	-
HTS 5	10	21	(up to 5)	8025	23	20	20	-
HTS 6	8	23	(up to 7)	5044	24	21	41	-
HTS 7	8	23	(up to 7)	5044	24	21	41	-
HTS 8	7	23	(up to 6)	13360	22	20	46	-
HTS 9	12	37	(up to 8)	7546	40	34	79	-
HTS XL	18	57	[12 3 4 2 2 8 2 2 2 ... ... 2 1 4 8 1 1 1 1 1]	416	77	51	143	-
HTS XXL	18	87	[18 5 6 2 4 12 4 3 3 ... ... 3 1 6 12 1 1 2 2 2]	634	118	77	350	-

Table 7.2 provides the globally optimal values for the objective function, i. e. the cycle time, and compares them to the best solutions that could be found manually or from bounds.[3]

Table 7.2. HTS examples: optimization results.

Name	known solution	T_{opt}	throughput improved by
EXAMPLE1	50	40	25%
ELISA	895	770	16.2%
HTS f	840	210	300%

Continued on next page

[3]Time units may, e. g., be seconds, minutes, hours, or parts thereof. Although this is the case for all examples in this section, the globally optimal cycle time does not need to be integer even if all weights in the time window precedence network are integer numbers.

Continued from previous page

Name	known solution	T_{opt}	throughput improved by
HTS 1	163	113	44.2%
HTS 2	598	490	22%
HTS 3	127	127	0%
HTS 4	1479	855	73%
HTS 5	400	308	29.9%
HTS 6	228	99	130%
HTS 7	1263	469	169%
HTS 8	743	743	0%
HTS 9	683	531	28.6%
HTS XL	5548	600	825%
HTS XXL	5548	878	532%

In the case of examples HTS3 and HTS8, globally optimal solutions had already been found by experts through manual search. However, other than in manual search, the method from this thesis not only provides a solution but also proves global optimality.

Finally, Table 7.3 provides the size of the resulting mixed integer linear program (5.77) in terms of integer variables (#int), real variables (#real) and constraints (#constr)[4,5]. Additionally, the lower bound T_{min} and the upper bound T_{max} is given. All mixed integer linear programs incorporate additional bounds and cuts as derived in Section 5.6. Along with the computation times that are needed by the CPLEX solver to find the proven globally optimal solution (t_{solve}), Table 7.3 shows the computation time after which any first feasible solution (t_{1st}) has been found and the time at which the globally optimal solution has been shown (t_{fin}), yet not necessarily proven.[6] Along the computation process, an MILP solver can provide information on the remaining maximum gap between the best solution found so far and the best solution that might still be possible. Table 7.3 provides this value together with t_{1st} and t_{fin}. All Computation times are given in CPU seconds on a 2GHz Linux PC.

[4] Pure bounds for variables are not counted as constraints.

[5] In addition, integer variables z_t that occur only in redundant constraints, are removed from the MILP in advance.

[6] A gap of 0% for t_{fin} signifies that at the time of first showing the solution, the solver simultaneously proved global optimality.

Table 7.3. HTS examples: mixed integer linear programs.

Name	#real	#int	#con	T_{min}	T_{max}	t_{solve}	t_{1st} (gap)	t_{fin} (gap)
EXAMPLE1	4	6	30	40	50	< 0.01	< 0.01 (15.00%)	< 0.01 (0%)
ELISA	28	85	1165	670	895	0.12	< 0.1 (5.48%)	0.12 (0%)
HTS f	19	19	233	187	840	0.01	0.01 (0%)	0.01 (0%)
HTS 1	23	33	322	108	163	0.04	0.04 (0%)	0.04 (0%)
HTS 2	25	29	291	480	598	0.08	< 0.01 (2.07%)	< 0.01 (2.07%)
HTS 3	18	40	507	127	127	< 0.01	< 0.01 (0%)	< 0.01 (0%)
HTS 4	22	50	518	816	1479	0.03	< 0.01 (15.75%)	0.03 (0%)
HTS 5	21	18	145	287	308	0.02	0.02 (0%)	0.02 (0%)
HTS 6	22	41	447	96	202	0.06	< 0.01 (3.12%)	< 0.01 (3.12%)
HTS 7	22	41	447	457	1041	0.04	< 0.01 (2.63%)	0.04 (0%)
HTS 8	21	46	321	743	743	0.02	0.02 (0%)	0.02 (0%)
HTS 9	35	79	507	531	531	0.05	0.05 (0%)	0.05 (0%)
HTS XL	52	131	1051	600	5548	0.65	0.4 (3.50%)	0.65 (0%)
HTS XXL	78	321	2402	878	5548	122	75 (0.68%)	122 (0%)

The following conclusions can be drawn from Table 7.3:

- The mixed integer linear programs for all real HTS problems can be solved in less than one second.[7] The large problem 'HTS XXL' can be solved in highly satisfying time.
- In many cases, a first solution is found early.
- In most cases, the user may already be satisfied with the first solution, since the gap, i. e. the maximally possible improvement, is shown to be small.[8]

The data given in Table 7.3 refers to the MILP (5.77), i. e. the mathematical problem formulation that includes model reduction from Section 4.2 as well as bounds and cuts from Section 5.6.

In the following, we pick four of the HTS examples to compare the MILP computation times that are needed if some of the steps towards the mixed integer linear program (5.77) are omitted. Data is given in Table 7.4 for the following variants:

(all) MILP according to (5.77), i. e. including the model reduction steps, bounds and cuts (repetition of the respective line from Table 7.3).

(-BC) MILP according to (5.36), i. e. including the model reduction steps from Section 4.2 but excluding bounds and cuts from Section 5.6.

(-C) MILP (5.77a-c,e-h), i. e. including the model reduction steps and bounds but excluding the cuts (5.77d).

(raw) MILP derived directly from the original time window precedence network and the disjunctive constraints (5.24), i. e. omitting model reduction steps as well as bounds and cuts.

(-mr) MILP (5.77) including the bounds and cuts, but omitting the model reduction steps from Section 4.2.

A dash (-) in column t_{solve} marks MILP's that could not be solved by GAMS/CPLEX within reasonable time (24hrs).

[7] Of course, preprocessing (import of the problem instance definition, model reduction, generation of the disjunctive constraints as well as the additional cuts and bounds) also takes a few seconds.

[8] This is, of course, only relevant for the large examples, since all other solutions are returned in an instant.

Table 7.4. HTS examples: comparison of MILP variants.

Name	#real	#int	#con	T_{min}	T_{max}	t_{solve}	t_{1st} (gap)	t_{fin} (gap)
ELISA (-BC)	28	130	321	n/a	n/a	–	2 (26.09%)	– (–)
ELISA (-C)	28	130	321	670	895	3	< 3 (1.99%)	< 3 (1.99%)
ELISA (all)	28	85	1165	670	895	0.12	< 0.1 (5.48%)	0.12 (0%)
ELISA (-mr)	144	130	5660	670	12980	88	< 0.1 (17.39%)	5 (11.59%)
HTS 9 (-BC)	35	79	244	n/a	n/a	–	0.1 (61.22%)	– (–)
HTS 9 (-C)	35	79	244	392	531	9	0.2 (35.46%)	0.2 (35.46%)
HTS 9 (all)	35	79	507	531	531	0.05	0.05 (0%)	0.05 (0%)
HTS 9 (raw)	270	79	1075	n/a	n/a	–	79 (77.30%)	– (–)
HTS 9 (-mr)	270	79	1338	531	531	0.28	0.28 (0%)	0.28 (0%)
HTS XL (-BC)	52	143	432	n/a	n/a	51	51 (0%)	51 (0%)
HTS XL (-C)	52	143	432	600	5548	17	17 (0%)	17 (0%)
HTS XL (all)	52	131	1051	600	5548	0.65	0.4 (3.50%)	0.65 (0%)
HTS XL (-mr)	261	143	1943	600	5548	8	2 (1.65%)	8 (0%)
HTS XL (raw)	261	143	1181	n/a	n/a	11715	7304 (1.33%)	11715 (0%)
HTS XXL (-BC)	78	350	913	n/a	n/a	–	– (–)	– (–)
HTS XXL (-C)	78	350	913	878	5548	–	– (–)	– (–)

Continued on next page

Continued from previous page

Name	#real	#int	#con	T_{min}	T_{max}	t_{solve}	t_{1st} (gap)	t_{fin} (gap)
HTS XXL (all)	78	321	2402	878	5548	122	75 (0.68%)	122 (0%)
HTS XXL (-mr)	397	350	2055	878	5548	–	– (–)	– (–)
HTS XXL (raw)	397	350	$> 1e4$	n/a	n/a	–	– (–)	– (–)

From Table 7.4, three main conclusions can be drawn:

- Omitting cuts (5.77d) from Section 5.6 resulted in an increase in computation time for the globally optimal solution by at least a factor of 25 – compare (-C) to (all).
- Omitting both, cuts and bounds from Section 5.6, a globally optimal solution could only be found for one of the examples – see (-BC).
- Omitting the model reduction steps from Section 4.2 resulted in a significant increase of computation times for the globally optimal solution – compare (-mr) to (all). A globally optimal solution could only be found in one case (and this took more than 3 hrs) for those cases where the MILP was directly derived from the original time window precedence network and the disjunctive constraints – compare (raw) to (-BC).

Since the branch and bound methods that are implemented in an MILP solver involve discrete decisions, a relatively small variation of the problem formulation or even a change in the solver's parameters may significantly influence computation times for the globally optimal solution. Nevertheless, the results from Table 7.4 suggest to apply all steps of model reduction and strengthening of the problem formulation that have been presented in this thesis.

The method for mathematical modeling and solution of the cyclic scheduling problem can be applied to all HTS assays that evolve from practical applications. As described, it also includes the following special cases:

- Multi-capacity resources: the system involves resources that have a capacity greater than one but small enough to potentially limit throughput. The appropriate extensions are given in Section 6.2.

- Sequence dependent setup times: the time that is needed for cleanup of a resource may depend on the sequence of activities on this resource as described in Section 6.3. The same effect occurs if the times a transportation device needs for empty travels, i. e. to reach the starting position for the next transport activity, cannot be neglected.

- Schedules for *simultaneous conduction of multiple assays*: it can sometimes be very effective to run multiple different assays in parallel [48]. Resources that are not heavily used by the activities of one assay may be used by activities that belong to another assay, i. e. a different set of substances and worksteps. Of course, all assays will be run with the same cycle time T. The standard method can be used to do this just by composing a 'merged single batch time scheme' from the single batch time schemes of the individual assays.

- k-periodic schedules: for the special case in which incubation times are very long and the number of samples, i. e. the number of batches, is small, the last batch may start already before the time at which the first batch is completely finished. In this case, as well as some others, a strictly cyclic schedule for the whole process does normally not provide a good solution. Section 6.4 provides a method to cope with such assays.

7.2 Flexible manufacturing systems

A *flexible manufacturing system* (FMS) consists of manufacturing cells (machines) that are linked by an automatic material handling system (transportation devices). An FMS can be used to produce different types of products without the need of manually changing the system configuration in between. Several entities, which may be of different types, can be processed simultaneously.

There are some aspects in which flexible manufacturing systems differ from high throughput screening plants, as they have been discussed in the previous section.

- Time window constraints only appear for certain special products, e. g. for chemical process steps.

- There is normally no need to process all entities in an identical time scheme. Nevertheless, an artificial restriction to cyclic schedules may be desirable in order to find optimal schedules within reasonable computation time.

- Some flexible manufacturing systems involve only one robot, which conducts all transport activities (*robotic work-cell*).
- Some flexible manufacturing systems involve pallets or *AGVs* ('autonomous' or 'automatic' guided vehicles), on which the entities pass through the plant.

Following the generalized framework of this thesis, the cyclic process for flexible manufacturing systems can be modeled in terms of events and resources. No distinction is made between transportation units and machines. In flexible manufacturing systems, resources may be, e. g., machines, robots, pallets/AGVs or track segments. Activities may reflect worksteps on machines, material handling or transports.

If transport is performed by one robot, a production process may simply be modeled by a sequence of worksteps on machines. Loading times, i. e. the times needed to transfer parts between the robot and a machine, may be resource dependent or job dependent or (the most general case) activity dependent. Robot traveling times $t_{travel}(j2, j1)$ have to be known for all pairs of machines and may additionally be load-dependent (empty robot moves faster). They will normally meet the triangular inequality constraints

$$t_{travel}(j2, j1) \leq t_{travel}(j3, j1) + t_{travel}(j2, j3), \quad j1, j2, j3 \in \{1 \ldots m\} . \tag{7.1}$$

The minimum processing time for any transport activity is made up of the loading time, the traveling time to the next machine, and the unloading time.[9] After having unloaded a part, the empty robot moves to another machine, where another part is waiting for transport. A minimum time distance is needed between the two transport activities, which has to allow for the empty robot to move to the next machine. This time depends on the distance between the machines and therefore on the sequence of transport activities in the cyclic schedule. Hence, the empty robot traveling times involve sequence dependent setup times as described in Section 6.3. Special cases of robotic work-cells demand for *no-wait* constraints or *loaded robot no-wait* constraints [45]. Such constraints can be easily included in the model by using the idea of time window constraints.

[9]The *duration* of a transport activity may exceed its *minimum processing time* in the presence of idle times of the loaded robot. Nevertheless, it can be shown that such idle times are needed only in the presence of time window constraints, pre- or post-processing times, or for systems with more than one robot.

Example

An example for the solution of an FMS scheduling problem with the methods described in this thesis can be found in [40]. In the remainder of this section, we focus on an example, which illustrates the applicability of the methods to FMS scheduling along with the extension on *multi-capacity* resources. We consider a standard example from [41, 42] (a slightly modified version of an original example taken from [98]): a production plant is composed of four machine types M1, ..., M4, which can manufacture three products P1, P2, P3. The production ratio has to be P1:P2:P3 = 1:1:2, i. e. in each cycle, one part of type P1 and P2 each and two parts of type P3 have to be delivered. To simplify notation, we introduce two product types P31 and P32 that follow the same sequence of worksteps. Production of one product P1 resp. P2, P31, P32 involves the following worksteps in fixed sequence (processing times given in brackets):

- P1: M1 (1) , M2 (1) , M3 (3) , M4 (3)
- P2: M1 (1) , M4 (1) , M3 (2)
- P31: M1 (1) , M2 (2) , M4 (1)
- P32: M1 (1) , M2 (2) , M4 (1).

During production, the parts are carried by pallets: at the beginning of the first activity, each part of type Pi allocates one pallet of type PALi, $i \in \{1, 2, 31, 32\}$. The pallet is released at the time of finishing the last activity for the product.

There are 5 identical servers for machine type M1:

$$cap_{M1} = 5 \, .$$

For the other machines, only the sum of their capacities is limited to 20:

$$cap_{M2} + cap_{M3} + cap_{M4} \leq 20 \, . \tag{7.2}$$

The sum of pallets is assumed to be limited to 100, i. e.

$$cap_{PAL1} + cap_{PAL2} + cap_{PAL31} + cap_{PAL32} \leq 100 \, . \tag{7.3}$$

All transport times are neglected in this example. There is sufficient space for the loaded pallets to wait if the next machine is not yet free. In this, the example differs from the other examples that are considered in this thesis.

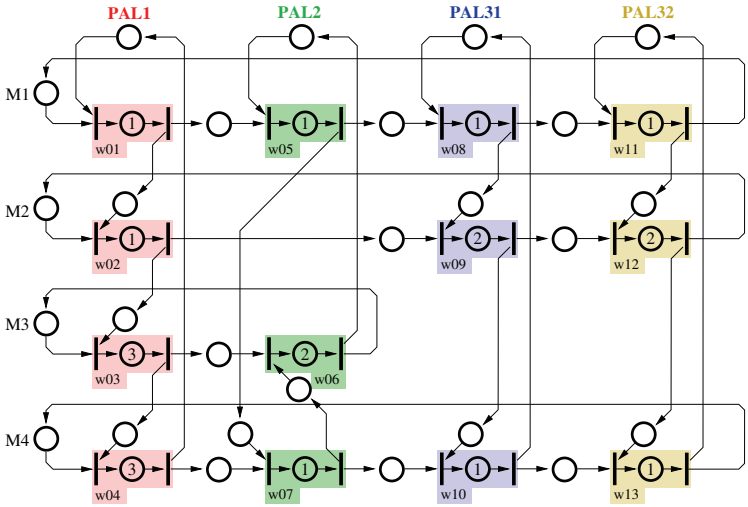

Figure 7.6. Petri net structure for FMS example with predefined sequences for machines.

Fig. 7.6 shows a Petri net system model, where the worksteps are represented by places with holding times and the workstep starting and finishing events are represented by transitions. Numbers in the places define their holding times, which are equivalent to the processing times of the worksteps. The number of tokens in places labeled with 'PAL1', 'PAL2', 'PAL31', 'PAL32' represents the number of pallets of the respective type that are not allocated at the moment. The number of tokens in places labeled with 'M1' to 'M4' represent the number of machine servers of the respective type that are not allocated at the moment. At the beginning of the process, all tokens are put into these places ('not allocated'). The pallet tokens come back to their 'not allocated' place when the complete sequence of worksteps for their product is finished. The machine server tokens come back to their 'not allocated' place when the complete sequence of worksteps defined for each machine type is finished. It is important to note that in the example pictured in Fig. 7.6, not only the sequence of worksteps for the products, but also the sequence of worksteps for the machines is fixed a-priori (For example, each machine server of type M4 is forced to perform worksteps w04, w07, w10 and w13 for one set of products and in this predefined sequence before becoming available for another set of products). This simplifies the

cyclic scheduling problem, since the disjunctive constraints reduce to simple sequencing constraints.

The minimum possible cycle time for this system is $T_{fixed} = 1$, as given in [42]. According to [42], this cycle time can be achieved with the following number of servers, i. e. the following capacities for the machines:

$$cap_{M1} = 5,\ cap_{M2} = 5,\ cap_{M3} = 9,\ cap_{M4} = 6\ . \tag{7.4}$$

Identical results are obtained when using the method presented in this thesis. The following problem-specific details have to be obeyed when building the mixed integer linear program.

- All resources are modeled with multiple capacity as described in Section 6.2.
- Resources M1 to M4 represent the four machine types. All four worksteps in the single batch time scheme that use machine M1 are combined into one singular activity (activity 1), which uses resource M1: the starting event of activity 1 is the start of workstep w01, the release event of activity 1 is the finishing event of workstep w11. The same is done for worksteps on machines M2 to M4, resulting in activities 2 to 4. The capacities of the resources M1 to M4 reflect the number of servers needed of each machine type.
- Resources PAL1 to PAL32 represent the four pallet types. In the single batch time scheme, similar to Resources M1 to M4, these resources are used by one activity each: for example, resource PAL2 is used by activity 6, which starts with the beginning of workstep w05, includes worksteps w05, w07 and w06, and is finished with the end of workstep w06.
- Capacities cap_j have to be defined as *variables* in the mathematical program. The sum constraints (7.2) and (7.3) have to be included.

The graph model reflecting the single batch time scheme is given in Fig. C.1 in the appendix. In a first step, the mixed integer linear program is set up and the optimum cycle time is computed. In a subsequent step, the cycle time is fixed to the optimum value and the sum of capacities

$$cap_{\sum} = cap_{M1} + cap_{M2} + cap_{M3} + cap_{M4} + cap_{PAL1} + cap_{PAL2} + cap_{PAL31} + cap_{PAL32} \tag{7.5}$$

is minimized. The Gantt chart for the *single batch time scheme* of the resulting optimal solution is given in Fig. 7.7. Worksteps for the four products P1, P2, P31, and P32 are represented by different colors. A new batch starts every 1 time unit, i. e. the single batch time scheme from Fig. 7.7 is repeated with a cycle time of 1. Note that there is only one activity per single batch for each of the resources. Within each machine activity, several worksteps for the different product types take place (according to the sequence prescribed in Fig. 7.6). For illustration, these worksteps are marked by small lines of the respective colors.

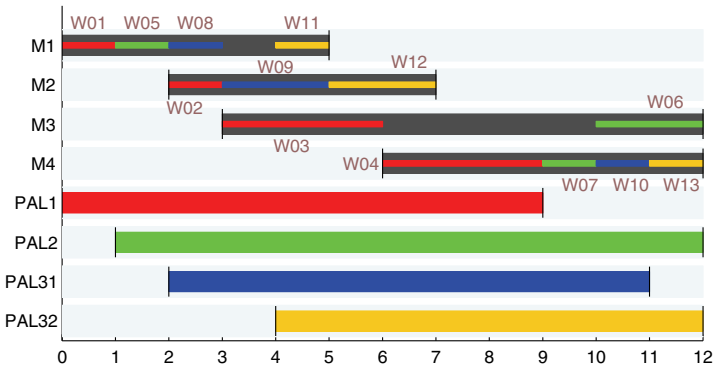

Figure 7.7. Optimal solution for predefined sequence (single batch time scheme).

From Fig. 7.7, it can be seen that servers of machine type M1 and M3 are sometimes allocated by activities without any work taking place. Actually, at any instant of time, at least 4 servers of type M3 and one server of type M1 are *idle*, i. e. no work is performed there, but nevertheless, they are allocated and therefore not available for any other batches. This is a direct consequence of pre-defining the sequence of worksteps for the machines.

Our framework additionally allows to compute an optimal cyclic schedule with minimum capacities if the sequence at which the machines conduct the worksteps is free for some or for all of the machines. In order to show this, we will now generalize the problem by allowing servers of machine M4 to accomplish worksteps for any batches and in any sequence. Of course, there may be production processes, where this is not possible. This is the case if byproducts result from a workstep that remain on the machine and block the machine such that it can only be used next to perform one specific follow-up workstep for

another product of the same batch. Nevertheless, in this generalized example, we assume for machine M4 that the worksteps for different products do not depend on each other.

Fig. 7.8 shows a Petri net for this configuration. The sequence of places for servers of type M4 is replaced by a non-deterministic place, which represents the new situation.

The cyclic scheduling model has to be changed accordingly: the four worksteps on machine M4 are modeled as separate activities instead of one activity. The modified graph model for the single batch time scheme can be found in the appendix in Fig. C.1. Again, in a first step, the mixed integer linear program is set up and the optimal cycle time is computed. The sum constraints (7.2) and (7.3) are included in the MILP formulation. In a subsequent step, the cycle time is fixed to the optimum value and the sum of capacities (7.5) is minimized. The optimum cycle time turns out to be

$$T_{opt} = 5/6 = 0.8333 \ ,$$

i. e. throughput can be improved by 20% if the sequence of worksteps for machine M4 is free. The capacities needed for machines M1 to M4 are now

$$cap_{M1} = 5, \ cap_{M2} = 6, \ cap_{M3} = 6, \ cap_{M4} = 8 \ .$$

The capacities for the pallets are

$$cap_{PAL1} = 11, \ cap_{PAL2} = 9, \ cap_{PAL31} = 5, \ cap_{PAL32} = 6 \ .$$

The capacity numbers, of course, still meet constraints (7.2) and (7.3). Fig. 7.9 shows the optimal single batch time scheme for this example. The single batch time scheme is cyclically repeated with a cycle time of $5/6$.

Finally, as a third example problem, the optimum cycle time can be further reduced to $T_{opt} = 1/1.25 = 0.8$ when allowing a free sequence of worksteps for *all* machines. This is a global lower bound to the cycle time, which results from the limitation $capM1 \leq 5$ and is also mentioned in [42].

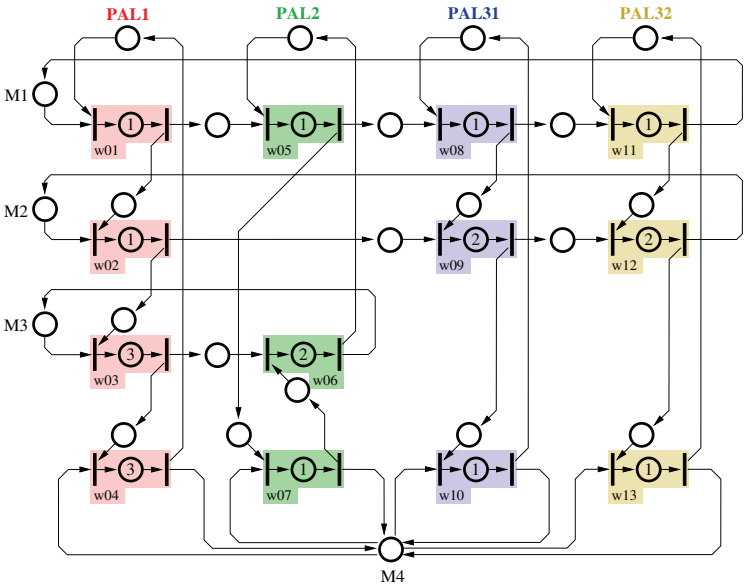

Figure 7.8. Petri net structure for FMS example with free sequence for machine M4.

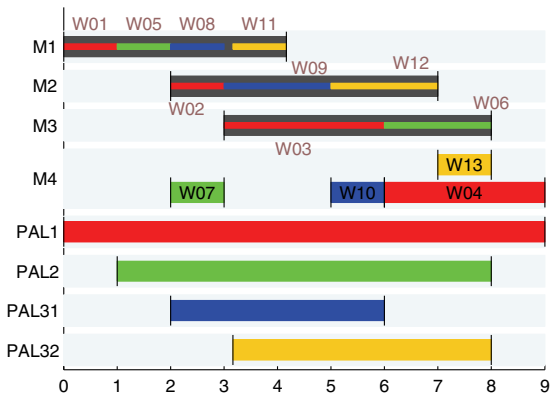

Figure 7.9. Optimal solution for free sequence on M4 (single batch time scheme).

7.3 Traffic and transportation

Traffic and transportation systems, such as road, ship and especially rail traffic, can be handled particularly well by methods from discrete event systems (DES) theory, as for most high-level control tasks in traffic and transportation systems, only discrete phenomena are important. The most efficient DES tool for the analysis and simulation of cyclically operated transportation systems is max-plus algebra. It can be used to

- analytically determine the minimal possible cycle time,
- identify critical paths,
- build efficient models for simulation.

This has been done intensively for rail traffic systems, where a cyclic timetable is often desirable [16, 108]. However, one of the most important degrees of freedom in rail scheduling cannot be addressed in max-plus algebra: if several trains use common track segments concurrently, the scheduling algorithm has to determine the sequence in which the trains are allowed to pass these track segments. As already mentioned, max-plus algebra, as such, does not allow for open sequence decisions in the model. This drawback is removed by the methods described in this thesis.

We will conclude this chapter by providing a small example that illustrates the application of strictly cyclic scheduling to rail traffic systems.

The considered system has originally been published in [69, 70]. It involves the track network pictured in Fig. 7.10. The network consists of three branches and a central switch. It is partitioned into 9 distinct segments, each of which can only be used by one train at a time. Trains may therefore exclusively meet at two double line segments in the middle of branch A-O and C-O.

It is assumed that trains start at point A and travel along the following route:

- from point A via point O to point C,
- from point C via point O to point B,
- from point B via point O to point A.

These three routes are pictured as arrows in Fig. 7.10. Traveling times are deterministic and do not depend on the train. Numbers next to the track segments in Fig. 7.10 show

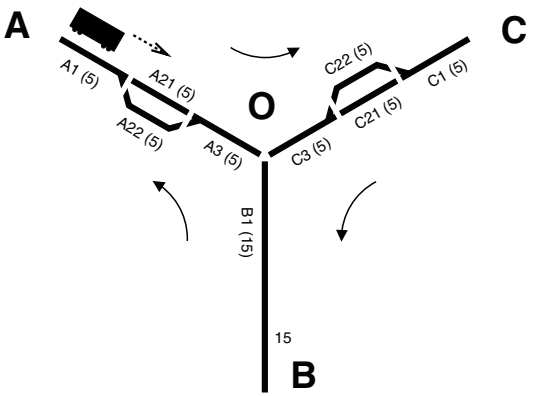

Figure 7.10. Simple track network for cyclic operation.

the time needed to cross the track segments in one direction. A train arriving at points A, B or C stops for unloading resp. loading. While loading or unloading, the respective track segment remains blocked by the train. For simplicity of representation, loading and unloading times as well as turnaround times are neglected in this example. After loading, the train continues its travel along the next route. It is assumed that this process is repeated infinitely.

If only one train exists, the cycle time obviously is $t_{opt,1} = 90$. As more trains come into play, a cyclic scheduling problem has to be solved in order to find the minimum possible cycle time and to determine the meeting places that allow for this cycle time. The cyclic scheduling problem consists of 9 track segment resources and a train resource, which is a multi-capacity resource and may be used to limit the number of available physical trains. The movement of a train is modeled by a sequence of 12 activities: activities 1 to 3 represent the train's travel along the track from point A to point O, activities 4 to 8 represent the travel from point O to point C as well as loading and returning to point O, activity 9 represents travel from point O to point B and back, and finally, activities 10 to 12 represent returning to point A. Subsequent traveling activities for the individual train overlap in time due to signaling safety: a track segment resource is not only occupied by the train during its travel from one end of the segment to the other end of the segment, but also for 1 time unit before the train enters it and for 1 time unit after the train has

left. Fig. 7.11 shows the Gantt chart for the single batch time scheme. A 'batch', in this case, subsumes the traveling of a train from point A back to point A as described above. One entry of multi-capacity resource 'train' is allocated at the beginning (before loading at point A) and released at the end (after unloading at point A).

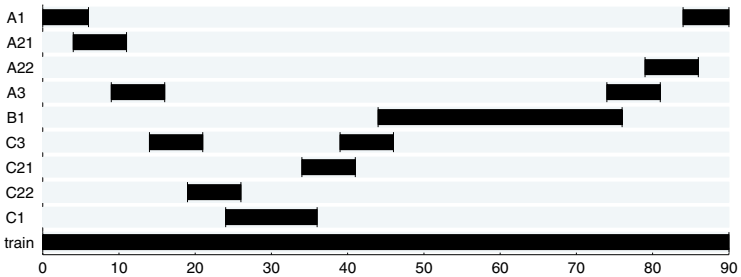

Figure 7.11. Resource allocations of one train (single batch time scheme).

The graph model for this example can be found in Fig. C.7 in the appendix. Based on this model, the minimum cycle time for different numbers of available trains can be easily determined by successively incrementing the capacity of resource 'train' and solving the resulting mixed integer linear program. The optimum cycle times are:

- two trains: $t_{opt,2} = 45$,
- three trains: $t_{opt,3} = 36$,
- four trains: $t_{opt,4} = 32$.

No better cycle time can be found for more than four trains: throughput is limited by the fact that trains may only meet at two places. Moreover, a fifth train would involve two trains allocating the resource at point A simultaneously, which is not allowed. Of course, the maximum reasonable number of trains could alternatively be found by first allowing for an infinite number of trains and solving the cyclic scheduling problem with respect to throughput and subsequently minimizing the capacity of resource 'train' as described in Section 6.2.

The solutions to the cyclic scheduling problems also provide information about the places where trains have to meet in order to achieve the optimal cycle times. For example, the optimal cyclic schedule for three trains is achieved if trains meet at the double line track segment in A - O. This scheme is pictured in Fig. 7.12.

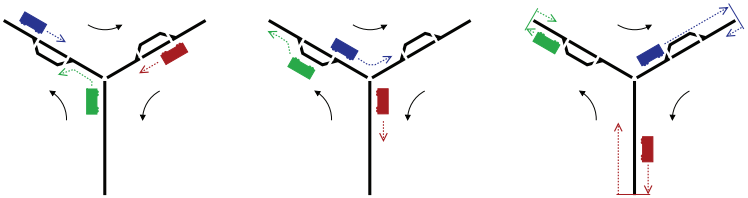

Figure 7.12. Optimal cyclic operation scheme for three trains.

Of course, the same method can easily be applied to larger networks, i. e. to determine the maximum possible throughput for a single track line with several stations, where oncoming trains may meet. Note that such a problem not only involves questions of scheduling (*Where should trains meet? Which train should go first?*) but also a deadlock avoidance problem (*Will the trains inevitably be blocked if another train enters the line now ?*). A deadlock occurs if some trains block each other and can only continue their journey after a manual intervention which includes the need of sending trains backwards. The rail traffic deadlock problem can be addressed by several DES theory methods (e. g. supervisory control theory [99]) and has been studied intensively. A deadlock avoidance approach inhibits train movements that would result in a deadlock. In such, it removes some possible actions from the system, but still leaves a number of different options. Not all of these options allow for a globally optimal solution concerning throughput. The scheduling approach presented in this thesis determines the globally optimal sequence of train movements and, automatically, ensures the avoidance of deadlocks (*non-blocking specification*).

Passenger rail systems are often run under a cyclic timetable with fixed cycle time (i. e. one hour or integer parts thereof). The optimization problem would then be not to minimize the cycle time, but to find timetables that minimize passenger traveling times and changeover times. For this problem, a different objective function would be needed. Nevertheless, the mathematical modeling framework for cyclically operated systems, as described in this thesis, will still hold.

Chapter 8

Conclusion

8.1 Summary

This work proposes a new solution method for a general class of cyclic scheduling problems. We presented a systematic approach that defines the task of scheduling as a mathematical optimization problem and results in globally optimal solutions.

The most essential part of optimization is the formulation of a suitable model for the underlying process. In this thesis, a systematic DES modeling approach has been presented, which is partitioned into two major steps:

1. setting up a discrete event model for the cyclic process, consisting of the single batch time scheme and the cycle time (Chapter 4),
2. transforming the DES model into a set of mathematical equations which, together with the disjunctive constraints allow to formulate the scheduling problem as a mathematical program (Chapter 5).

Since this approach starts from a general graph model for the single batch time scheme, i. e. the set of cyclically repeated events, it covers a very broad class of scheduling problems. The method allows to solve cyclic scheduling problems that combine general precedence network structure with time window constraints as well as blocking capability. Several extensions, of which the most important is probably the inclusion of resources with

multiple capacity, enable the method to deal with almost all systems that are based on strictly cyclic repetition of activities on a set of limited resources.

The focus of this work is on systematic analysis of the nature of the underlying timing and sequencing problems. Compared to other approaches, this allows to

- find proven globally optimal solutions, and to
- compute, along the search, a strict upper bound for the remaining distance to the globally optimal solution.

It turned out that a compact formulation of the problem makes the suggested method perfectly suitable to solve large real-world problems in a globally optimal sense within very short computation times. In a cyclic process with hundreds or thousands of cycles, even a small reduction of the cycle time pays off multiply. Therefore, putting effort into a systematic scheduling approach that guarantees a global optimum is obviously a rewarding investment.

The main results of this thesis are:

- Cyclic scheduling problems with precedence network structure, time window constraints, and blocking capability can be formulated as mixed integer linear mathematical programs. They can be solved in a globally optimal sense using a DES model for the cyclic process.
- Compact problem formulation is important to achieve short computation times. Additional bounding constraints as well as cuts significantly accelerate the computation of a globally optimal solution. Real problems from the HTS industries, for which no globally optimal solution had been known before, were solved with computation times of less than one second.
- Cyclic operation has the natural advantage of a regular and thus very compact structure. It may even be worthwhile to deliberately restrict a system to cyclic operation, as the globally optimal cyclic schedule may well be superior to the best solution that can be found by a limited search in the high-dimensional set of acyclic schedules.

8.2 Perspectives

The strictly cyclic scheduling approach constitutes a fundamental framework on which further generalizations can be based. Such generalizations should include k-periodic schedules with different job time schemes, as well as pooling (see Section 6.4) and scheduling for cyclic processes with a relatively small number of batches, where the minimization of the cycle time will not result in good values for the makespan, i. e. the overall time interval between the starting event of the first batch and the terminal event of the last batch.

We also suggest to investigate hybrid system models, which involve continuous dynamics that take place within the activities and thus offer additional degrees of freedom, which enter a generalized objective function in terms of energy or cost. Scheduling based on hybrid system models is especially important for chemical processes, but may e. g. as well be used for rail transportation systems, where spare time can be exploited by reducing velocity and thus energy consumption.

Although the requirement of regular timing in strictly cyclic operation makes static scheduling a mandatory requirement, deviations from the predetermined schedule may inevitably occur during runtime. If this happens, dynamic scheduling strategies are needed, which change the timing of events as well as the order of activities within a limited time horizon. The objective of online rescheduling is to ensure that throughput is kept at a maximum possible rate while as many time window constraints as possible are kept. It is an advantage of cyclic operation that a rescheduling strategy may abort processing for a single batch (e. g. if compulsory time window constraints failed to hold), thus getting back to regular operation, and simply add an additional batch at the end of the overall process. If activity sequences are changed, there is a possibility of the non-blocking specification being violated. In this case, the algorithm also has to make sure that deadlocks are excluded. Such rescheduling strategies may also be advantageous if requirements on the timing of certain batches change during runtime (incremental scheduling).

Finally, the a-priory offline scheduling may be refined towards non-strictly cyclic schedules. This can be done in several ways. For example, the start and end phase of a cyclic process with a fixed number of batches may allow for reduced values for the cycle time T. The start and end phase of the cyclic process can additionally be accelerated if the time schemes of consecutive cycles are allowed to deviate ('jitter'). The latter idea may also be exploited for the entire cyclic schedule in an enhanced cyclic scheduling strategy. A similar problem is discussed in [15].

High throughput screening systems marked the start in the application of the strictly cyclic scheduling method. Throughout the work for this thesis, a large number of application areas for cyclic processes have been identified, but left untouched due to lack of time. Some of them will produce scheduling problems that can instantly be solved with the methods presented in this work. Some of them, however, will raise new challenges and will therefore be the motivation for the further development of systematic approaches on the very interesting problem of cyclic scheduling.

Appendix A

Proofs

Cuts from Section 5.6

Proof We will consider separately the two cases $D_{i3,i4,i5,i6} \geq 0$ and $D_{i3,i4,i5,i6} < 0$. From the definition of $\underline{z}_{(i3,i4,i5,i6)}$ for $D_{i3,i4,i5,i6} \geq 0$, together with $T \leq T_{max}$, we get

$$\underline{z}_{(i3,i4,i5,i6)} = \lceil \frac{D_{i3,i4,i5,i6}}{T_{max}} \rceil - 1 < \frac{D_{i3,i4,i5,i6}}{T_{max}} \leq \frac{D_{i3,i4,i5,i6}}{T}$$

and therefore

$$D_{i3,i4,i5,i6} > \underline{z}_{(i3,i4,i5,i6)} \cdot T \ . \tag{A.1}$$

Since

$$\begin{aligned} D_{i3,i4,i5,i6} &= \min_{\theta_1^* \ldots \theta_K^* \text{ s.t. (5.34c)}} \Big(r_{i3} - o_{i5} + r_{i6} - o_{i4} \Big) \Big|_{\theta_1^* \ldots \theta_K^*} \\ &\leq \Big(r_{i3} - o_{i5} + r_{i6} - o_{i4} \Big) \Big|_{\theta_1 \ldots \theta_K} \text{ for any } \theta_1 \ldots \theta_K \ , \end{aligned}$$

Equation (A.1) results in

$$r_{i3} - o_{i5} + r_{i6} - o_{i4} > \underline{z}_{(i3,i4,i5,i6)} \cdot T \ . \tag{A.2}$$

Combining (5.24a) with $i1 := i3$, $i2 := i4$ and (5.24b) with $i1 := i5$, $i2 := i6$, together with (A.2) yields

$$(z_{i3,i4} - z_{i5,i6} - 1) \cdot T < - \underline{z}_{(i3,i4,i5,i6)} \cdot T \ .$$

With $T > 0$, we get

$$z_{i3,i4} - z_{i5,i6} - 1 < - \underline{z}_{(i3,i4,i5,i6)} \ .$$

Since $z_{i3,i4}$ as well as $z_{i5,i6}$ and $\underline{z}_{(i3,i4,i5,i6)}$ are integer numbers, this finally results in

$$z_{i3,i4} - z_{i5,i6} \leq - \underline{z}_{(i3,i4,i5,i6)} \ .$$

For $D_{i3,i4,i5,i6} < 0$, since $T \geq T_{min}$, constraint (A.1) still holds:

$$\underline{z}_{(i3,i4,i5,i6)} = \lceil \frac{D_{i3,i4,i5,i6}}{T_{min}} \rceil - 1 < \frac{D_{i3,i4,i5,i6}}{T_{min}} \leq \frac{D_{i3,i4,i5,i6}}{T}$$

The rest of the proof is identical to the first case $D_{i3,i4,i5,i6} \geq 0$.

The cuts for the special cases $i3 = i4$ and $i5 = i6$ result in an equivalent way using the fact that

$$r_i - o_i \leq T \quad \forall i = 1, \ldots, n \ .$$

□

Bounding constraints from Section 5.6

In the following we proof that the bounding constraints (5.65) (5.70), and (5.74) meet the admissibility constraints (5.63), which we repeat here for convenience:
According to (5.63), a new upper bound $\hat{\theta}_{k^*,max}$ for a variable $\hat{\theta}_{k^*}$ in the mixed integer linear program (5.57) is admissible if

$$\begin{aligned}
\forall(\hat{\theta}, T) \text{ with} \quad & \hat{\kappa}\hat{\theta} = \vartheta && \text{(A.3)} \\
& \hat{\theta}_{k^*} > \hat{\theta}_{k^*,max} && \text{(A.4)} \\
\exists\, \xi \in \mathbb{Z}^{\hat{K}} \text{ s.th.} \quad & \hat{\kappa}\xi = 0 && \text{(A.5)} \\
& \phi\xi = 0 && \text{(A.6)} \\
& \xi_{k^*} \leq (\hat{\theta}_{k^*,max} - \hat{\theta}_{k^*})/T && \text{(A.7)} \\
& \hat{\theta} + \xi T \geq 0 && \text{(A.8)}
\end{aligned}$$

Note that all $\hat{\theta}_k \geq 0$, $k = 1, \ldots, \hat{K}$, as well as $T > 0$.

1. Proof (5.65)
In (5.65), an upper bound $\hat{\theta}_{k^*,max} = T_{max}$ is established for all k^* with

$$\begin{aligned}
\hat{\kappa}_{p,k^*} &= 0, \qquad p = 1, \ldots, P && \text{(A.9)} \\
\phi_{i,k^*} &= 0, \qquad i = 1, \ldots, n \ . && \text{(A.10)}
\end{aligned}$$

This upper bound meets the admissibility conditions (A.5) to (A.8) by choosing

$$\xi_k = \begin{cases} -\lfloor \frac{\hat{\theta}_k}{T} \rfloor & \text{for } k = k^* \\ 0 & \text{for } k \neq k^* \ : \end{cases} \tag{A.11}$$

conditions (A.5) and (A.6) are met trivially. From (A.11) and $T \leq T_{max}$, it follows that[1]

$$\xi_{k^*} \leq 1 - \frac{\hat{\theta}_{k^*}}{T} \leq \frac{T_{max}}{T} - \frac{\hat{\theta}_{k^*}}{T} \ ,$$

and thus (A.7). Condition (A.8) is trivially met for $k \neq k^*$. It also holds for $k = k^*$, since

$$\hat{\theta}_{k^*} + \xi_{k^*} \cdot T = \hat{\theta}_{k^*} - \lfloor \frac{\hat{\theta}_{k^*}}{T} \rfloor \cdot T \geq \hat{\theta}_{k^*} - \frac{\hat{\theta}_{k^*}}{T} \cdot T = 0 \ .$$

□

2. Proof (5.70)
For this case, the following prerequisites hold:

$$\begin{aligned} \hat{\kappa}_{1,1} = \hat{\kappa}_{1,2} &= 1 \\ \hat{\kappa}_{p,1} = \hat{\kappa}_{p,2} &= 0 \qquad \forall p \in 2, \ldots, P \ , \\ \phi_{i,1} = \phi_{i,2} &\qquad \forall i \in 1, \ldots, n \ . \end{aligned} \tag{A.12}$$

For these prerequisites and the upper bound $\hat{\theta}_{1,max} = T_{max}$, all admissibility conditions (A.5) to (A.8) are met:
for any $\hat{\theta}_1 > T_{max}$, we choose

$$\xi_k = \begin{cases} -\lfloor \frac{\hat{\theta}_1}{T} \rfloor & \text{for } k = 1 \\ +\lfloor \frac{\hat{\theta}_1}{T} \rfloor & \text{for } k = 2 \\ 0 & \text{for } k \geq 3 \ , \end{cases} \tag{A.13}$$

for which (A.5) obviously holds. Condition (A.6) is ensured by the prerequisite (A.12). From (A.13) and $T \leq T_{max}$, it again follows that

$$\xi_1 \leq 1 - \frac{\hat{\theta}_1}{T} \leq \frac{T_{max}}{T} - \frac{\hat{\theta}_1}{T} \ ,$$

and thus (A.7). Finally, (A.8) holds for $k = 1$, since

$$\hat{\theta}_1 + \xi_1 \cdot T = \hat{\theta}_1 - \lfloor \frac{\hat{\theta}_1}{T} \rfloor \cdot T \geq \hat{\theta}_1 - \frac{\hat{\theta}_1}{T} \cdot T = 0 \ .$$

Equivalently, it holds for $k = 2$ since $\xi_2 \geq 0$ and obviously for $k = 3, \ldots, \hat{K}$. □

[1] Note that $\lfloor x \rfloor \geq x - 1$.

3. Proof (5.74)
In the third case, the following prerequisites hold:

$$\begin{aligned} \hat{\kappa}_{1,1} = -1 \ , \quad \hat{\kappa}_{1,2} = 1 & \\ \hat{\kappa}_{p,1} = \hat{\kappa}_{p,2} = 0 \qquad & \forall p \in 2, \ldots, P \ , \\ \phi_{i,1} = -\phi_{i,2} \qquad & \forall i \in 1, \ldots, n \ . \end{aligned} \tag{A.14}$$

Therefore the first line of the linear equation system (A.3) reads

$$-\hat{\theta}_1 + \hat{\theta}_2 + \sum_{k=3}^{\hat{K}} \hat{\kappa}_{1,k} \cdot \hat{\theta}_k = \vartheta_1 \ . \tag{A.15}$$

For the sum term in (A.15), the following upper bound has to be known:

$$\sum_{k=3}^{\hat{K}} \hat{\kappa}_{1,k} \cdot \hat{\theta}_k \leq ub \ . \tag{A.16}$$

For these prerequisites, the bounding constraint

$$\hat{\theta}_1 \leq \hat{\theta}_{1,max} = \max(T_{max}, T_{max} - \vartheta_1 + ub) \tag{A.17}$$

meets all admissibility conditions (A.5) to (A.8):
for any $\hat{\theta}_1 > \hat{\theta}_{1,max}$, we choose

$$\xi_k = \begin{cases} -\lfloor \frac{\hat{\theta}_1 - \hat{\theta}_{1,max}}{T} \rfloor - 1 & \text{for } k = 1, 2 \\ 0 & \text{for } k \geq 3 \end{cases} \tag{A.18}$$

for which (A.5) obviously holds and condition (A.6) is ensured by the prerequisite (A.14).
From (A.18), it follows that

$$\xi_1 \leq -\frac{\hat{\theta}_1 - \hat{\theta}_{1,max}}{T} \ ,$$

and thus (A.7) is met. Condition (A.8) holds for $k = 1$, since

$$\xi_1 \geq -\frac{\hat{\theta}_1 - \hat{\theta}_{1,max}}{T} - 1$$

and therefore

$$\hat{\theta}_1 + \xi_1 \cdot T \geq \hat{\theta}_{1,max} - T \geq T_{max} - T \geq 0 \ .$$

Equivalently, condition (A.8) holds for $k = 2$ since

$$\xi_2 = \xi 1 \geq +\frac{\hat{\theta}_{1,max} - \hat{\theta}_1}{T} - 1 \geq \frac{T_{max} - \vartheta_1 - ub - \hat{\theta}_1}{T} - 1 \geq \frac{-\vartheta_1 - ub - \hat{\theta}_1}{T}$$

and therefore

$$\hat{\theta}_2 + \xi_2 \cdot T \geq \hat{\theta}_2 - \vartheta_1 - ub - \hat{\theta}_1 \ ,$$

where $\hat{\theta}_2 - \vartheta_1 - ub - \hat{\theta}_1 \geq 0$ because of (A.15) and the upper bound (A.16). Finally, condition (A.8) obviously holds for $k = 3, \ldots, \hat{K}$. □

Appendix B

Algorithms

This appendix gives short listings of the algorithms needed to perform model reduction as developed in Section 4.2.

Algorithm to find redundant and strongly redundant arcs according to Fact 3:

All strongly redundant arcs of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, WT)$ can be found by simply checking the following condition for WT:

$$wt_{q2,q1} < wt^*_{q2,q1} \Longrightarrow \text{arc } (q2, q1) \text{ is strongly redundant.} \tag{B.1}$$

In (B.1), $wt^*_{q2,q1}$ refers to the $(q2, q1)$-element of the closure WT^* of WT. Prerequisite (4.15) ensures the existence of the closure.

Several algorithms may be used to find redundant, but *not strongly* redundant arcs. An arc $(q1, q2)$ in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, WT)$ is redundant iff the closure WT^* is equal to the closure WT'^* of a graph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}', WT')$ with $\mathcal{E}' = \mathcal{E} \setminus (q1, q2)$. Faster algorithms reduce the problem to a *shortest path problem*, see e. g. [10]. However, redundant, but not strongly redundant arcs constitute a singularity in the parameter space of the problem definition, which will only accidentally be hit by the user. Searching for redundant, but not strongly redundant arcs will therefore normally not pay off by significant reduction of graph size.

Algorithm to remove self-loops according to Fact 4:

This can be performed by trivially setting

$$\mathrm{diag}(WT) = -\inf \ .$$

Algorithm to find circuits of weight zero according to Fact 2:

Again, the closure WT^* can be used for a simple algorithmic method: all pairs of nodes $(q1, q2)$ with

$$wt_{q2,q1} = -wt^*_{q1,q2} \tag{B.2}$$

can be contracted.

Algorithm for node contraction (2 nodes):

The new set of arcs $\mathcal{E}^{(k+1)}$ after contraction of node $q1$ and node $q2$ can be obtained from $\mathcal{E}^{(k)}$ by the following steps:

- replace every arc $(q2, qi)$ with a new arc $(q1, qi)$ with weight $wt'_{qi,q1} = \max(wt_{q2,q1} + wt_{qi,q2}, wt_{qi,q1})$.
- replace every arc $(qi, q2)$ with a new arc $(qi, q1)$ with weight $wt'_{q1,qi} = \max(wt_{q2,qi} - wt_{q2,q1}, wt_{q1,qi})$.
- remove arc $(q1, q2)$ and node $q2$.

The mapping $\mathcal{M} = (\mathcal{Q}[e], \Delta[e])$ has to be adjusted accordingly:

$$\mathcal{Q}[e]^{(k+1)} = \begin{cases} q1 & \text{for } \mathcal{Q}[e]^{(k)} = q2 \\ \mathcal{Q}[e]^{(k)} & \text{for } \mathcal{Q}[e]^{(k)} \neq q2 \ , \end{cases} \tag{B.3}$$

$$\Delta[e]^{(k+1)} = \begin{cases} \Delta[e]^{(k)} + wt_{q2,q1} & \text{for } \mathcal{Q}[e]^{(k)} = q2 \\ \Delta[e]^{(k)} & \text{for } \mathcal{Q}[e]^{(k)} \neq q2 \ . \end{cases} \tag{B.4}$$

Appendix C

Problem instances

This appendix provides the data for all problem instances that are presented in detail in Chapter 7. The single batch time scheme definition is given in the form of time window precedence networks equivalent to Fig. 4.2. The batch start node and the batch terminal node as well as the arcs connecting those two nodes with the activitiy start nodes resp. activity release nodes are omitted.

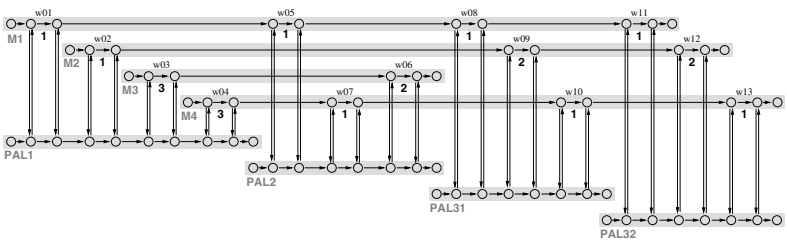

Figure C.1. Time window precedence network for FMS example from Section 7.2 (fixed sequence of worksteps for all machines).

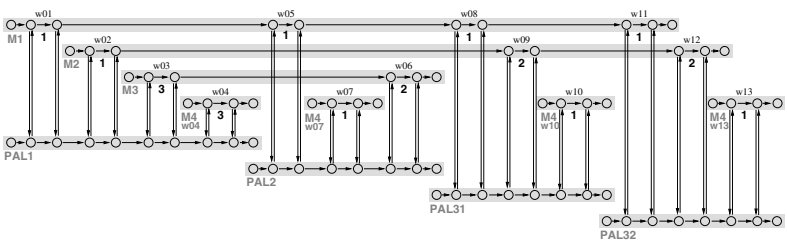

Figure C.2. Time window precedence network for FMS example from Section 7.2 (free sequence of worksteps for machine M4).

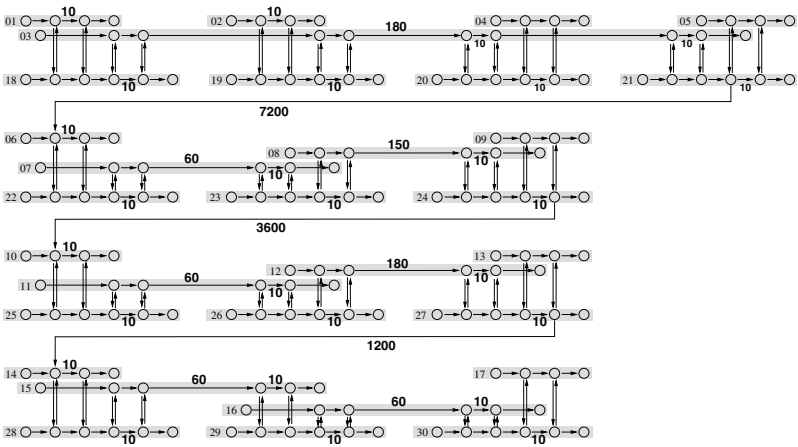

Figure C.3. Time window precedence network for ELISA example from Section 7.1 (activity numbers are defined in Fig. C.4. Arcs without labels have weight 0).

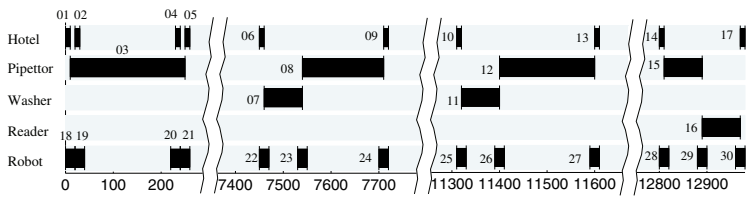

Figure C.4. Single batch time scheme (Gantt chart) with activity numbers for the ELISA example.

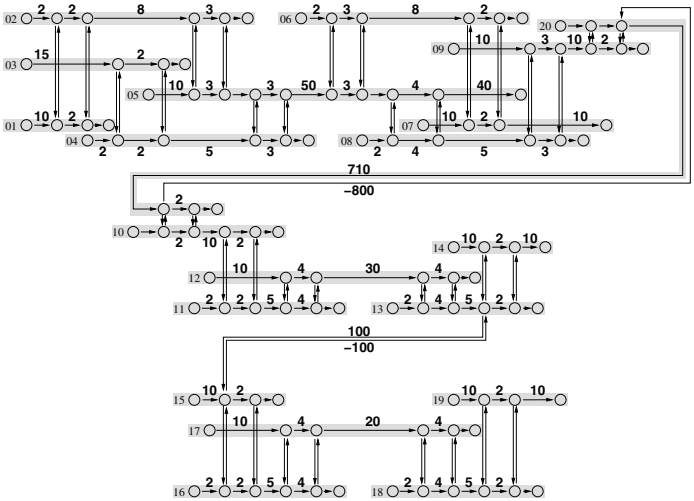

Figure C.5. Time window precedence network for 'HTS f' example from Section 7.1 (activity numbers are defined in Fig. C.6. Arcs without labels have weight 0).

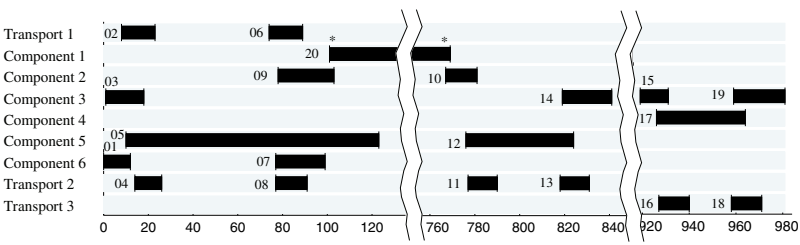

Figure C.6. Single batch time scheme (Gantt chart) with activity numbers for the 'HTS f' example.

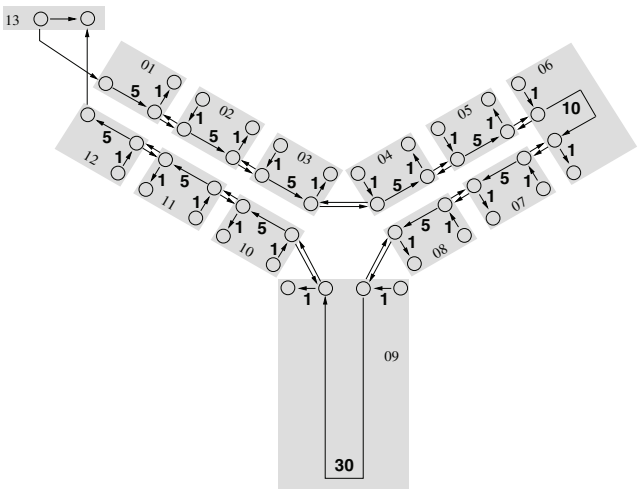

Figure C.7. Time window precedence network for the simple track network example from Section 7.3 (activity numbers are defined in Fig. C.8. Arcs without labels have weight 0).

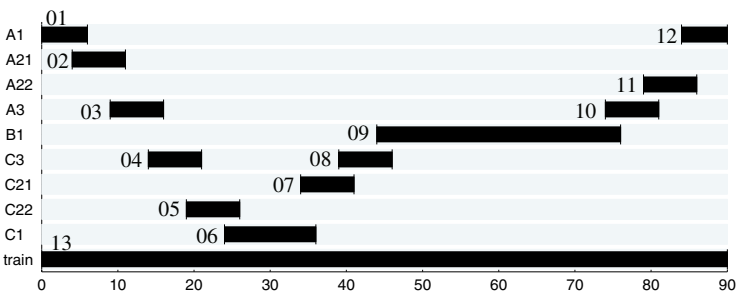

Figure C.8. Single batch time scheme (Gantt chart) with activity numbers for the simple track network example.

Appendix D

Notation

Abbreviations

AGV	Autonomous guided vehicle
DES	Discrete event systems
FMS	Flexible manufacturing system
HTS	High throughput screening
LP	Linear program
MILP	Mixed integer linear program
MINLP	Mixed integer nonlinear program
NLP	Nonlinear program
WIP	Work in progress

Latin Symbols

$act_i(t)$	Number of batches for which activity i is in process at time instant t
bc_j	Parameter for capacity sum constraints
$bz_{s,\iota}$	Definition of cuts in the strengthened problem formulation
cap_j	Capacity resp. multiplicity of resource j
cc	Parameter for capacity sum constraints
cz_s	Definition of cuts in the strengthened problem formulation
$D_{i3,i4,i5,i6}$	Abbreviation, see (5.46)
$\underline{D}_{i3,i4,i5,i6}$	Abbreviation, see (5.48)
$\mathcal{D}$	Set of events in the problem instance definition
$e_o(i)$	Event that marks the start of activity i

Continued on next page

Continued from previous page

$e_r(i)$	Event that marks the end of activity i
e_s	Batch start event
e_t	Batch terminal event
$\mathcal{E}$	Set of arcs in graph $\mathcal{G}$
$\mathcal{E}_a$	Set of arcs in problem definition
$\mathcal{E}_c$	Set of events after completion of model reduction
f_0	Parameter in definition of generalized objective function
fc_j	Parameter in objective functions involving capacities
f_k	Parameter in definition of generalized objective function
g_t	Abbreviation, see (6.64)
$\mathcal{G}$	$= (\mathcal{V}, \mathcal{E}, WT)$. Weighted Graph
$\mathcal{G}^*$	Transitive closure of $\mathcal{G}$
$\mathcal{G}_a$	Weighted graph representation of problem definition
$\mathcal{G}_c$	Weighted graph after completion of model reduction
i	Index of activity within single batch time scheme
j	Index of resource
J_i	Resource allocated by activity i
J_i^*	Resource allocated by activity i of the job time scheme (k-periodic schedules)
k	Index of time variable θ
K	Number of time variables parametrizing the single batch time scheme
K^*	Number of time variables parametrizing the job time scheme for k-periodic schedules
$\hat{K}$	$= K + P$
l_b	Number of batches, if defined
$load_j(t)$	Load of multi-capacity resource j at time instant t
m	Number of resources (with capacity 1)
$\tilde{m}$	Number of resources with capacity $cap_j > 1$
$\mathcal{M}$	Mapping $\mathcal{D} \mapsto \mathcal{V} \times \mathbb{R}$, consisting of $\mathcal{Q}[e]$ and $\Delta[e]$
n_j	Number of activities in the single batch time scheme that take place on resource j
n	Number of activities in the single batch time scheme (taking place on resources with capacity 1)
n^*	Number of activities in the job time scheme for k-periodic schedules

Continued on next page

Continued from previous page

$\tilde{n}$	Number of activities in the single batch time scheme that take place on resources with capacity $cap_j > 1$
$\mathbb{N}$	Set of strictly positive integer numbers
o_i	Starting time of activity i in single batch time scheme (starting event)
o_i^*	Starting time of activity i in job time scheme (k-periodic schedules)
$o_i^{(\rho)}$	Starting time of activity i of batch ρ
p	Index of line in additional constraints $\kappa \cdot \theta \leq \vartheta$
q	Node resp. index of node
$q_o(i)$	Node for starting event of activity i
$q_r(i)$	Node for release event of activity i
q_s	Node representing batch start event e_s
q_t	Node representing batch terminal event e_t
$\mathcal{Q}[e]$	(Number of) node to which event e is associated
r_i	End time of activity i in single batch time scheme (release event)
r_i^*	End time of activity i in job time scheme (k-periodic schedules)
$r_i^{(\rho)}$	End time of activity i of batch ρ
$\mathbb{R}$	Set of rational numbers
$\mathbb{R}_{max}$	$= \mathbb{R} \cup \{-\infty\}$
s	Index of line in cuts for the strengthened problem formulation
$sw_{(i1,i2)}$	Switching time between activity $i1$ and activity $i2$
$swms(j)$	Lower bound for sum of switching times on resource j
S	Number of cuts in the strengthened problem formulation
t_{1st}	Computation time for a first feasible solution
t_{fin}	Computation time, after which the globally optimal solution is shown
t_q	Time instant for node q
t_{solve}	Computation time for the MILP
T	Cycle time
T^*	Inner cycle time for k-periodic schedules
$\bar{T}$	$= 1/T$ Inverse of cycle time (throughput)
T_b	$= x_{e_t} - x_{e_s}$ Batch duration
T_{max}	Upper bound for cycle time
T_{min}	Lower bound for cycle time
T_{lo}	Lower bound for T/Y (k-periodic schedules)
T_{up}	Upper bound for T/Y (k-periodic schedules)

Continued on next page

Continued from previous page

$\tilde{u}_\zeta$	Integer variable in disjunctive constraints for multi-capacity resources
$\tilde{u}_{\zeta,0}$	$= \chi_{i2,0} - \chi_{i1,0}$
$\tilde{u}_{\zeta,k}$	$= \chi_{i2,k} - \chi_{i1,k}$
$v_{\iota,0}$	$= \chi_{i2,0} - \psi_{i1,0}$
$v_{\iota,k}$	$= \chi_{i2,k} - \psi_{i1,k}$
$\tilde{v}_{\zeta,0}$	$= \chi_{i2,0} - \psi_{i1,0}$
$\tilde{v}_{\zeta,k}$	$= \chi_{i2,k} - \psi_{i1,k}$
$\tilde{v}_\zeta$	Integer variable in disjunctive constraints for multi-capacity resources
$\mathcal{V}$	Set of nodes in graph $\mathcal{G}$
$\mathcal{V}_a$	Set of nodes in problem definition
$\mathcal{V}_c$	Set of nodes after completion of model reduction
$\mathcal{V}_{(i)}$	Set of nodes for events in activity i
$\mathcal{V}_{ip}$	Set of input nodes
$\mathcal{V}_o$	Set of nodes for starting events
$\mathcal{V}_r$	Set of nodes for release events
$\mathcal{V}_{op}$	Set of output nodes
$w_{\iota,0}$	$= \psi_{i2,0} - \chi_{i1,0}$
$w_{\iota,k}$	$= \psi_{i2,k} - \chi_{i1,k}$
$wt_{q2,q1}$	Weight of arc $(q1, q2)$
$wt^*_{q2,q1}$	Weight of arc $(q1, q2)$ in closure
$wt_{c,q2,q1}$	Weight of arc $(q1, q2)$ in reduced graph
WT	Weighting matrix of graph
WT^*	Closure of matrix WT
WT_c	Weighting matrix of reduced graph (after completion of model reduction)
x_e	Time instant for event e within single batch time scheme
y_i	Minimum number of jobs in one batch for job i to take place (k-periodic schedules)
Y	Number of jobs per batch for k-periodic schedules
Y_{max}	Maximum number of jobs per batch for k-periodic schedules
$z_{(i1,i2)}$	Nesting level for activities $i1$ and $i2$
z_ι	Nesting level for the activity pair denoted by ι
$\hat{z}_{(i1,h,h+1)}$	Binary variables for additional cuts (5.51)
$\underline{z}_{(i3,i4,i5,i6)}$	Abbreviation, see (5.45)
$\mathbb{Z}$	Set of integer numbers

Greek Symbols

$\hat{\theta}$	$= [\theta^T \ \breve{\theta}^T]^T$
$\breve{\theta}$	Slack variables for P additional constraints on θ
θ_k	Time variables parametrizing the time instants within the single batch time scheme
$\theta_{k,max}$	Upper bound for θ_k in strengthened problem formulation
$\bar{\theta}_k$	Normalized time variables after transformation into MILP
ϑ_p	$= \bar{c}_{K+1+p}$ Definition of additional constraints $\kappa \cdot \theta \leq \vartheta$
ρ	Index of batch
$\Upsilon_{i,\zeta}$	Abbreviation, see (6.25)
$\phi_{i,k}$	Abbreviation, see (5.61)
$\chi_{i,0}$	Abbreviation, see (4.30), resp. (6.54)
$\chi_{i,k}$	Abbreviation, see (4.31), resp. (6.54)
$\chi^*_{i,0}$	Abbreviation, see (4.30)
$\chi^*_{i,k}$	Abbreviation, see (4.31)
$\psi_{i,0}$	Abbreviation, see (4.32), resp. (6.54)
$\psi_{i,k}$	Abbreviation, see (4.33), resp. (6.54)
$\psi^*_{i,0}$	Abbreviation, see (4.32)
$\psi^*_{i,k}$	Abbreviation, see (4.33)
$\gamma_{j,0}$	$= \sum_{i=1}^{n} (\psi_{i,0} - \chi_{i,0}) \delta_{J_i j}$
$\gamma_{j,k}$	$= \sum_{i=1}^{n} (\psi_{i,k} - \chi_{i,k}) \delta_{J_i j}$
$\gamma_{j,0,Y}$	Abbreviation, see (6.58)
$\gamma_{j,k,Y}$	Abbreviation, see (6.58)
$\delta_{J_i j}$	Kronecker symbol
$\Delta[e]$	Time offset for event e in mapping $\mathcal{M}$
$\Delta_{y_i Y}$	Abbreviation, see (6.56)
ζ	Index denoting a sorted pair of activities $(i1, i2)$ on multi-capacity resources
ζ_{max}	Number of sorted pairs of activities $(i1, i2)$ on multi-capacity-resources
ι	Index denoting a sorted pair of activities $(i1, i2)$ within the single batch time scheme
ι_{max}	Number of sorted pairs of activities $(i1, i2)$ within the single batch time scheme for which $J_{i1} = J_{i2}$
$\kappa_{p,k}$	$= \bar{B}_{K+1+p,k}$ Definition of additional constraints $\kappa \cdot \theta \leq \vartheta$
$\hat{\kappa}$	$= [\kappa \ I^{P \times P}]$

Bibliography

[1] I.N.K. Abadi, N.G. Hall, and C. Sriskandarajah. Minimizing cycle time in a blocking flowshop. *Operations Research*, 48(1):177–180, 2000.

[2] W.P. Adams and H.D. Sherali. Linearization strategies for a class of zero-one mixed integer programming problems. *Operations Research*, 38(2):217–226, 1990.

[3] A. Agnetis, M. Lucertini, and F. Nicol. Flow management in flexible manufacturing cells with pipeline operations. *Management Science*, 39(3):294 –306, 1993.

[4] K.A. Aldakhilallah and R. Ramesh. Cyclic scheduling heuristics for a re-entrant job shop manufacturing environment. *International Journal of Production Research*, 39(12):2635– 2657, 2001.

[5] A. Alle, L.G. Papageorgiou, and J.M. Pinto. A mathematical programming approach for cyclic production and cleaning scheduling of multistage continuous plants. *Computers & Chemical Engineering, UK*, 28(1-2):3 – 15, 2004.

[6] AMPL. A Modeling Language for Mathematical Programming, 2006. URL `http://www.ampl.com`.

[7] M. Aramaki, K. Enjohji, M. Yoshimura, M. Sakawa, and K. Kato. HTS (high throughput screening) system scheduling through genetic algorithms. In *Knowledge-Based Intelligent Information Engineering Systems & Allied Technologies. KES'2001. Part 2*, pages 1345–1349, Osaka, Japan, September 2001.

[8] R.D. Armstrong and P. Sinha. The cyclic separation scheduling problem. *Naval Research Logistics Quarterly*, 24(4):609–618, 1977.

[9] F. Baccelli, G. Cohen, G.J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity*. Wiley Series in Probability and Mathematical Statistics. Wiley, 1992. Available online: http://www-rocq.inria.fr/metalau/cohen/SED/book-online.html.

[10] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications*. Springer, 2001.

[11] M. Beggs and J.S. Major. Flexible use of people and machines. In *High Throughput Screening - The Discovery of Bioactive Substances*, pages 471–481. Marcel Dekker Inc., 1997.

[12] D.P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2004.

[13] M.L. Biros. The state of hts automation: comparing integrated robotics and workstations. *The Industrial Robot*, 29(1):38– 42, 2002.

[14] R. Boel, X.R. Cao, G. Cohen, A. Giua, W.M. Wonham, and J.H. van Schuppen. Unity in diversity, diversity in unity: Retrospective and prospectiveviews on control of discrete event systems. *Discrete Event Dynamic Systems*, 12(3):253 –264, 2002.

[15] T. Boudoukh, M. Penn, and G. Weiss. Scheduling jobshops with some identical or similar jobs. *Journal of Scheduling*, 4(4):177 –199, 2001.

[16] J.G. Braker. *Algorithms and Applications in Timed Discrete Event Systems*. PhD thesis, Department of Technical Mathematics and Informatics, Delft University of Technology, Delft, 1993.

[17] P. Brucker. *Scheduling Algorithms*. Springer, 2004.

[18] P. Brucker. Scheduling and constraint propagation. *Discrete Applied Mathematics*, 123:227 – 256, 02.

[19] H. Camus, H. Ohl, O. Korbaa, and J.C. Gentina. Cyclic schedules in flexible manufacturing systems with groups of identical machines. In *Rensselaer's Fifth International Conference on Computer Integrated Manufacturing and Automation Technology CIMAT'96,*, pages 345– 350, Grenoble, France, May 1996.

[20] H. Camus, H. Ohl, O. Korbaa, and J.C. Gentina. Petri net modeling of flexible operating sequences in a fms and implications for the search of cyclic schedules. In *Proc. Computational Engineering in Systems Applications*, pages 218– 225, Lille, France, July 1996.

[21] H. Chen, C. Chu, and J.-M. Proth. Cyclic scheduling of a hoist with time window constraints. *IEEE Transactions on Robotics & Automation*, 14(1):144 – 152, 1998.

[22] P. Chrétienne. On graham's bound for cyclic scheduling. *Parallel Computing*, 26 (9):1163–1174, 2000.

[23] P. Chrétienne. The basic cyclic scheduling problem with deadines. *Discrete Applied Mathematics*, 30:109–123, 1991.

[24] P. Chrétienne. List schedules for cyclic scheduling. *Discrete Applied Mathematics*, 94(1-3):141– 159, 1999.

[25] C. Chudy. System of two cyclic processes with common resource: Modelling and performance analysis. In *Cybernetics and Systems '96. Proceedings of the Thirteenth European Meeting on Cybernetics and Systems Research.*, pages 271–275, Vienna, Austria, April 1996.

[26] Y. Crama and J. van de Klundert. Cyclic scheduling of identical parts in a robotic cell. *Operations Research*, 45(6):952–965, 1997.

[27] Y. Crama, V. Kats, J. van de Klundert, and E. Levner. Cyclic scheduling in robotic flowshops. *Annals of Operations Research*, 96:97– 124, 2000.

[28] CSDES Software. Cyclic Scheduling for Discrete Event Systems, MPI Magdeburg and IFAT, University of Magdeburg, 2006.

[29] R. Cuninghame-Green. *Minimax-Algebra*, volume 166 of *Lecture Notes in Economics and Mathematical Systems*. Springer, 1979.

[30] dash optimization website. dash optimization, 2006.
URL `http://www.dashoptimization.com/`.

[31] A. Donzel, J. Carmona, and L.A. Corkan. Perspectives on scheduling. In *High Throughput Screening - The Discovery of Bioactive Substances*, pages 525 –545. Marcel Dekker Inc., 1997.

[32] W.P. Janzen (ed.). *High Throughput Screening. Methods and Protocols*. Humana Press, 2002.

[33] M. Entzeroth. Real-time scheduling and multitasking at the computer level, management of unplanned situations-a practical approach. *Laboratory Automation & Information Management*, 33(2):87 – 92, 1997.

[34] A. Di Febbraro, S. Grosso, and R. Minciardi. Max-plus algebra and incremental scheduling problems in manufacturing systems. In *Proceedings of the 33rd IEEE Conference on Decision and Control*, pages 1583–1587, Lake Buena Vista, FL, USA, December 1994.

[35] A. Di Febbraro, R. Minciardi, and S. Sacone. Deterministic timed event graphs for performance optimization of cyclic manufacturing processes. *IEEE Transactions on Robotics & Automation*, 13(2):169–181, 1997.

[36] M.N. Feiglin, S. Skwish, M. Laab M, and A. Heppel. Implementing multilevel dynamic scheduling for a highly flexible 5-rail high throughput screening system. *Journal of Biomolecular Screening*, 5(1):39 –47, 2000.

[37] C.A. Floudas. *Deterministic global optimization : theory, methods and applications*. Kluwer, 2000.

[38] G. Frey. Modellierung flexibler Fertigungslinien und Bestimmung gültiger Produktionsfolgen mit Hilfe einer ereignisdiskreten Zustandsbeschreibung. *at - Automatisierungstechnik*, 41(1):16–25, 2000.

[39] GAMS. General Algebraic Modeling System, 2006.
URL `http://www.gams.com`.

[40] F. Geyer. Analyse und Optimierung zyklischer ereignisdiskreter Systeme mit Reihenfolgealternativen. Diplomarbeit, IFAT, Otto-von-Guericke-Universität Magdeburg, 2004.

[41] A. Giua, A. Piccaluga, and C. Seatzu. Optimal token allocation in timed cyclic event-graphs. In *Proc. 4th Int. Workshop on Discrete Event Systems - WODES 2000*, pages 209–218, Ghent, Belgium, August 2000.

[42] A. Giua, A. Piccaluga, and C. Seatzu. Firing rate optimization of cyclic timed event graphs by token allocations. *Automatica*, 38:91 –103, 2002.

[43] F. Glover. Linearization strategies for a class of zero-one mixed integer programming problems. *Management Science*, 22(4):455 – 460, 1975.

[44] J.L. Gross and J. Yellen, editors. *Handbook of Graph Theory*. CRC Press, 2003.

[45] N.G. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Operations Research*, 44(3):510 –525, 1996.

[46] N.G. Hall, T.-E. Lee, and M.E. Posner. The complexity of cyclic shop scheduling problems. *Journal of Scheduling*, 5(4):307 –327, 2002.

[47] S. Hamilton. Introduction to screening automation. In W.P. Janzen, editor, *High Throughput Screening*, pages 169 – 193. Humana Press, 2002.

[48] S.D. Hamilton. Hts automation study: results from a 2001 survey of the current vs. desired state of hts automation. *JALA*, 7(2):78 –83, 2002.

[49] C. Hanen. Study of a np-hard cyclic scheduling problem: the recurrent job-shop. *European Journal of Operational Research*, 72(1):82 – 101, 1994.

[50] C. Hanen and A. Munier. Cyclic scheduling on parallel processors: An overview. In P. Chrétienne, E.G. Coffman, J.K. Lenstra, and Z. Liu, editors, *Scheduling Theory and its applications*, pages 193–226. Wiley, New York, 1995.

[51] C. Hanen and A. Munier. A study of the cyclic scheduling problem on parallel processors. *Discrete Applied Mathematics*, 57(2-3):167–192, 1995.

[52] P. Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, 1999.

[53] P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica: A Modeling Language for Global Optimization*. MIT Press, 1997.

[54] C. Horst. Throughput-optimal cyclic scheduling with pooling resources. Diplomarbeit, IFAT, Otto-von-Guericke-Universität Magdeburg, 2006.

[55] W.T. Huh, G. Janakiraman, P.L. Jackson, and N. Sawhney. Minimizing flow time in cyclic schedules for identical jobs with acyclic precedence: the bottleneck lower bound. *Operations Research Letters*, 31(5):366 –374, 2003.

[56] ILOG website. ILOG, 2006. URL `http://www.ilog.com`.

[57] I. Ioachim, E. Sanlaville, and M. Lefebvre. The basic cyclic scheduling model for robotic flow shops. *INFOR*, 39(3):257– 277, 2001.

[58] J. Kallrath. *Gemischt-ganzzahlige Optimierung: Modellierung in der Praxis*. Vieweg, 2002.

[59] S. Karabati and P. Kouvelis. Cyclic scheduling in flow lines: Modeling observations, effective heuristics and a cycle time minimization procedure. *Naval Research Logistics*, 43:211– 231, 1996.

[60] V. Kats and E. Levner. Cyclic scheduling in a robotic production line. *Journal of Scheduling*, 5(1):23 – 41, 2002.

[61] V. Kats and E. Levner. A strongly polynomial algorithm for no-wait cyclic robotic flowshop scheduling. *Operations Research Letters*, 21(4):171–179, 1997.

[62] V. Kats, E. Levner, and L. Meyzin. Multiple-part cyclic hoist scheduling using a sieve method. *IEEE Transactions on Robotics & Automation*, 15(4):704–713, 1999.

[63] O. Korbaa, H. Camus, and J.-C. Gentina. Heuristic for the resolution of the general fms cyclic scheduling problem. In *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation. IEEE Part vol. 3*, pages 2903–2908, Orlando, FL, USA, October 1997.

[64] O. Korbaa, H. Camus, and J.-C. Gentina. A new cyclic scheduling algorithm for flexible manufacturing systems. *International Journal of Flexible Manufacturing Systems*, 14(2):173 –187, 2002.

[65] P. Kumar and K. Kothandaraman an P. Ferreira. Scalable and maximally-permissive deadlock avoidance for fms. In *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, pages 580– 585, Urbana, IL, USA, May 1998.

[66] T.-E. Lee. Stable earliest starting schedules for cyclic job shops: a linear system approach. *International Journal of Flexible Manufacturing Systems*, 12(1):59 – 80, 2000.

[67] T.-E. Lee and M.E. Posner. Performance measures and schedules in periodic job-shops. *Operations Research*, 45(1):72–91, 1997.

[68] E. Levner, V. Kats, and V.E. Levit. An improved algorithm for cyclic flowshop scheduling in a robotic cell. *European Journal of Operational Research*, 97(3):500 – 508, 1997.

[69] D. Li, E. Mayer, and J. Raisch. A novel hierarchical control architecture for a class of discrete- event systems. In *WODES - 7th International Workshop on Discrete Event Systems*, pages 415–420, Reims, France, 2004.

[70] D. Li, E. Mayer, and J. Raisch. A new hierarchical control scheme for a class of cyclically repeated discrete-event systems. In *ICINCO 2005 - Signal Processing, Systems Modeling and Control*, pages 30–36, Barcelona, Spain, 2005.

[71] LINGO. Lindo Systems Inc, 2006. URL `http://www.lindo.com`.

[72] E. Mayer. Ereignisdiskrete Modellierung und Analyse eines Schienennetzes. Studienarbeit, ISR, Universität Stuttgart, 1997.

[73] E. Mayer and J. Raisch. Throughput-optimal scheduling for cyclically repeated processes. In *MMAR2003 - 9th IEEE International Conference on Methods and Models in Automation and Robotics*, pages 871–876, Miedzyzdroje, Poland, August 2003.

[74] E. Mayer and J. Raisch. Time-optimal scheduling for high throughput screening processes using cyclic discrete event models. *MATCOM - Mathematics and Computers in Simulation*, 66(2-3):181–191, 2004.

[75] E. Mayer and J. Raisch. Modelling and optimization for high-throughput-screening systems. In *ADCHEM2003 - International Symposium on Advanced Control of Chemical Processes*, Hong-Kong, January 2004.

[76] E. Mayer, U.-U. Haus, J. Raisch, and R. Weismantel. Throughput-optimal sequences for cyclically operated plants. *Discrete Event Dynamic Systems*. Submitted, 2007.

[77] K. Menke. An industrial engineering approach to laboratory automation for high throughput screening. *Journal of Automated Methods & Management in Chemistry*, 22(5):143–144, 2000.

[78] Z. Michaelewicz and D.B. Fogel. *How to solve it: modern heuristics*. Springer, 2000.

[79] M. Middendorf and V. G. Timkovsky. On scheduling cycle shops: classification, complexity and approximation. *Journal of Scheduling*, 5(2):135–169, 2002.

[80] B.V. Mishra, E. Mayer, J. Raisch, and A. Kienle. Short-term scheduling of chemical processes: an overall optimisation approach. In *ECCE2003 - Fourth European Congress of Chemical Engineering*, 2004.

[81] B.V. Mishra, E. Mayer, J. Raisch, and A. Kienle. Short-term scheduling of batch processes: a comparative study of different approaches. *Industrial and Engineering Chemistry Research*, 44:4022–4034, 2005.

[82] MPL. Mathematical Programming Language, 2006.
URL `http://www.maximal-usa.com/mpl/`.

[83] A. Munier. Résolution d'un problème d'ordonnancement cyclique a itérations indépendantes et contraintes des ressources. *Recherche opérationelle/Operations Research*, 25(2):161–182, 1991.

[84] A. Munier. The complexity of a cyclic scheduling problem with identical machines and precedence constraints. *European Journal of Operational Research*, 91(3): 471–480, 1996.

[85] T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE, vol.77, no.4*, pages 541 – 580, April 1989.

[86] C. Murray and C. Anderson. Scheduling software for high-throughput screening. *Laboratory Robotics & Automation*, 8(5):295– 305, 1996.

[87] W.C. Ng and J. Leung. Determining the optimal move times for a given cyclic schedule of a material handling hoist. *Computers & Industrial Engineering*, 32(3): 595–606, 1997.

[88] A. Obuchowicz and Z. Banaszak. Modelling and performance evaluation of repetitive automated manufacturing: an algebraic approach. In *Proceedings of the Second International Symposium on Methods and Models in Automation and Robotics*, pages 809–814, Miedzyzdroje, Poland, August 1995.

[89] M.A. Odijk. A constraint generation algorithm for the construction of periodic railway timetables. *Transportation Research Part B-Methodological*, 30B(6):455–464, 1996.

[90] H. Ohl, H. Camus, E. Castellain, and J.-C. Gentina. A heuristic algorithm for the computation of cyclic schedules and the necessary wip to obtain optimal cycle time.

In *Proceedings of the Fourth International Conference on Computer Integrated Manufacturing and Automation Technology*, pages 339– 344, Los Alamitos, CA, USA, October 1994.

[91] H. Ohl, H. Camus, E. Catrsellain, and J.-C. Gentina. Petri net modeling of ratio-driven flexible manufacturing systems and implications of the wip for cyclic schedules. In *Proceedings of the 1995 IEEE-Systems Man and Cybernetics Conference. Vol.4*, pages 3081–3086, Vancouver, Canada, October 1995.

[92] OPL Development Studio. ILOG, 2006.
URL `http://www.ilog.com/products/oplstudio/`.

[93] OR/MS. Linear Programming Software Survey, 2005.
`http://www.lionhrtpub.com/orms/surveys/LP/LP-surveymain.html`.

[94] R.G. Parker. *Deterministic scheduling theory*. Chapman & Hall, 1995.

[95] P. Persi, W. Ukovich, and R. Pesenti. Two job cyclic scheduling with incompatibility constraints. *International Transactions in Operational Research*, 8(2):167 – 181, 2001.

[96] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Prentice Hall, 2002.

[97] J.M. Pinto and I.E. Grossmann. Optimal cyclic scheduling of multistage continuous multiproduct plants. *Computers & Chemical Engineering, UK*, 18(9):797 – 816, 1994.

[98] J.-M. Proth, N. Sauer, and X. Xie. Optimization of the number of transportation devices in a flexible manufacturing system using event graphs. *IEEE Transactions on Industrial Electronics*, 44(3):298–306, 1997.

[99] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event systems. *SIAM J. Control and Optimization*, 25:206 –230, 1987.

[100] W. Reisig. *Petrinetze - Eine Einführung*. Springer, 1990.

[101] R. Roundy. Cyclic schedules for job shops with identical jobs. *Mathematics of Operations Research*, 17(4):842 –65, 1992.

[102] N. Sauer. Optimization of cyclic manufacturing systems with stochastic manufacturing times using event graphs. *International Journal of Production Economics*, 46-47:387 – 399, 1996.

[103] G. Schilling and C.C. Pantelides. Optimal periodic scheduling of multipurpose plants. *Computers & Chemical Engineering*, 23(4-5):635–655, 1999.

[104] J.-W. Seo and T.-E. Lee. Steady-state analysis and scheduling of cyclic job shops with overtaking. *International Journal of Flexible Manufacturing Systems*, 14(4): 291–318, 2002.

[105] P. Serafini and W. Ukovich. A mathematical model for periodic scheduling problems. *SIAM J. on Discrete Mathematics*, 2(4):550 –581, 1989.

[106] N. Shah, C.C. Pantelides, and R.W.H. Sargent. Optimal periodic scheduling of multipurpose batch plants. *Annals of Operations Research*, 42(1-4):193–228, 1993.

[107] G. Sierksma. *Linear and integer programming: theory and practice*. Marcel Dekker, 2002.

[108] J. Stańczyk, E. Mayer, and J. Raisch. Modelling and performance evaluation of DES — a Max-Plus Algebra Toolbox for Matlab. In *Proc. 1st Int. Conf.: Informatics in Control, Automation and Robotics, ICINCO'04",*, pages 270–275, Setúbal, Portugal, August 2004.

[109] M. Stobbe and S. Engell. Intelligent scheduling of chemical plants: A constraint programming approach. In *Proceedings of the 1999 IEEE International Symposium on Intelligent Control Intelligent Systems and Semiotics*, pages 242–247, Cambridge, MA, USA, September 1999.

[110] B. Trouillet, A. Benasser, and J.-C. Gentina. Transformation of the cyclic scheduling problem of a large class of fms into the search of an optimized initial marking of a linearizable weighted t-system. In *Proc. 6th International Workshop on Discrete Event Systems - WODES 2002*, pages 83 – 90, Zaragoza, Spain, October 2002.