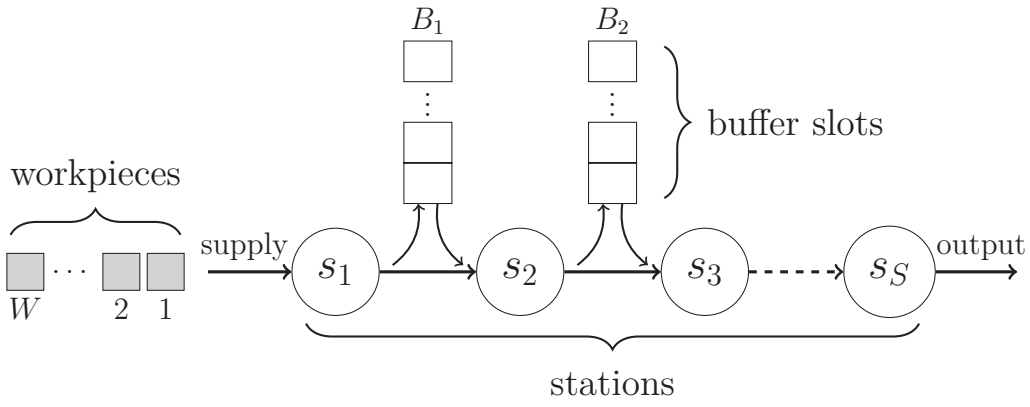
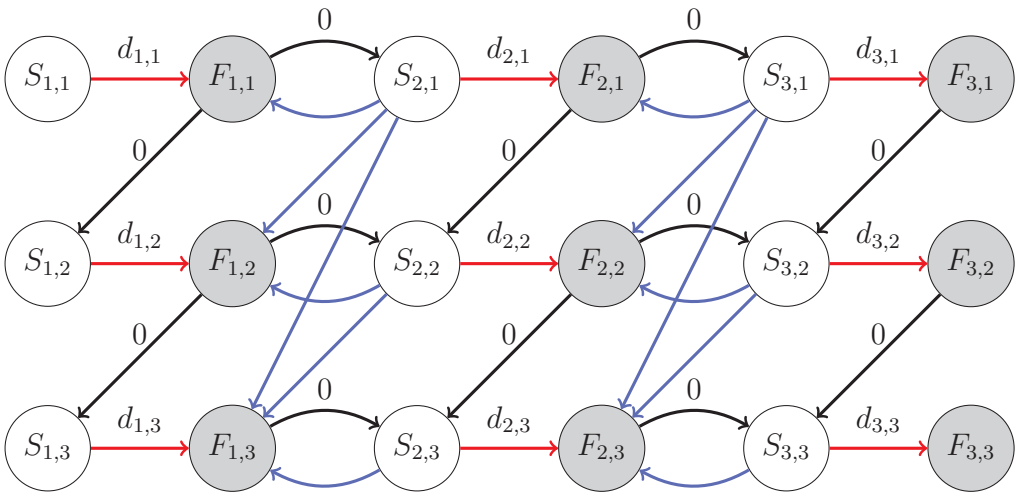
Pascal Wortel

Efficient Algorithms for Production Scheduling





TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

Fraunhofer ITWM

Fraunhofer-Institut für
Techno- und Wirtschaftsmathematik ITWM

Efficient Algorithms for Production Scheduling

Pascal Wortel

FRAUNHOFER VERLAG

Kontakt:
Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM
Fraunhofer-Platz 1
67663 Kaiserslautern
Telefon +49 631/31600-0
Fax +49 631/31600-1099
E-Mail info@itwm.fraunhofer.de
URL www.itwm.fraunhofer.de

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.
ISBN (Print): 978-3-8396-1609-3

D 386

Zugl.: Kaiserslautern, TU, Diss., 2019

Titelbild: © Pascal Wortel

Druck: Mediendienstleistungen des
Fraunhofer-Informationszentrum Raum und Bau IRB, Stuttgart

Für den Druck des Buches wurde chlor- und säurefreies Papier verwendet.

Fraunhofer-Informationszentrum Raum und Bau IRB
Postfach 80 04 69, 70504 Stuttgart
Nobelstraße 12, 70569 Stuttgart
Telefon 07 11 9 70-25 00
Telefax 07 11 9 70-25 08
E-Mail verlag@fraunhofer.de
URL http://verlag.fraunhofer.de

Efficient algorithms for production scheduling

Beim Fachbereich Mathematik
der Universität Kaiserslautern
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften
(Doctor rerum naturalium, Dr. rer. nat.)
genehmigte

Dissertation

von

Pascal Cléments Wortel

1. Gutachter: Prof. Dr. Sven O. Krumke
2. Gutachter: Prof. Dr. Marco Lübbecke

Datum der Disputation: 18. Dezember 2019

D 386

Aan mijn lieve ouders,
zonder wiens steun dit proefschrift niet
had kunnen bestaan.

Abstract

Scheduling and allocation problems have come to play a central role in the area of production planning, where they possess numerous applications. They have been extensively studied for over half a century, and come in a wide range of types and variations. Unfortunately, real world applications are often complex, and as a result, the best suitable formulations are $\mathcal{NP}$-hard. Adding additional constraints to such already hard to solve problems may result in computationally intractable models. Therefore, within the scheduling community, the study of such problems has often been avoided. This thesis can be seen as an attempt to close the gap between scheduling and other areas of optimization; we consider two such additional restrictions, namely robustness and inventory constraints. Our goal is to present new methods of dealing with these constraints within scheduling problems.

Firstly, we present new results in the field of robust allocation. In particular, we focus on the Buffer Allocation Problem (BAP). In the BAP, a production line consisting of machines and buffers is considered. In this scenario, if there is insufficient buffer between two machines, a station may be blocked from further processing while waiting for a downstream machine. On the other hand, adding buffers is associated with some cost. A compromise is sought between the total amount of buffers involved, and the increase of the throughput rate they account for. We propose models for a robust version of the BAP. For their resolution, different algorithms are derived using techniques from integer programming and graph theory. The efficiencies of the resulting implementations are compared and discussed.

Furthermore, we define a new theoretical framework for inventory-constrained scheduling. This is a relatively young area that was mostly studied in the context of constrained programming. In this setting, jobs are assumed to add or remove a given amount of material from a common stack. A schedule is only considered feasible when the stock of each material remains non-negative at all times. Based on applications from manufacturing industry, we justify the necessity for a new class of inventory-constrained problems, where the objective function does only depend on the consuming jobs, not the producing ones. We focus on the single-machine case, and provide complexity results for variations of the problem with different objective functions and constraints.

Zusammenfassung

Planungs- und Zuordnungsprobleme spielen seit langem eine große Rolle in der Optimierung, wo sie zahlreiche Anwendungen haben. Sie werden seit über fünf Jahrzehnten umfassend untersucht und kommen in einer Vielzahl von Variationen vor. Leider sind die realen Anwendungen häufig so komplex, dass die entsprechenden mathematischen Formulierungen $\mathcal{NP}$-schwer sind. Das Hinzufügen von zusätzlichen Einschränkungen zu solchen Problemen, die bereits in ihrer Grundform schwer sind, führt oft zu Modellen, die in praktikabler Zeit nicht zu lösen sind. Deshalb wird in der Scheduling-Community das Untersuchen solcher Probleme oft vermieden. In diser Dissertation werden zwei Arten von zusätzlichen Bedingungen betrachtet, nämlich Robustheits- und Vorfertigungsbedingungen. Unser Ziel ist es, neue Methoden für den Umgang von Planungsproblemen mit diesen Bedingungen zu entwerfen.

Zuerst präsentieren wir neue Ergebnisse im Bereich robuster Zuordnung. Der Fokus liegt dabei auf dem Puffer Zuordnungsproblem (BAP). Im BAP betrachtet man eine Fertigungslinie, die aus Maschinen und Puffern besteht. Falls in diesem Szenario zu wenige Puffer-Slots zwischen den Stationen verteilt sind, kann eine Maschine blockiert sein, weil sie auf eine nachgelagerte Maschine warten muss. Andererseits ist das Hinzufügen von Puffern mit Kosten verbunden. Gesucht ist ein Kompromiss zwischen der Anzahl an Puffer-Slots und dem Durchsatz der Linie. Wir schlagen Modelle für eine robuste Version des BAP vor; um diese zu lösen, werden verschiedene Algorithmen mittels Techniken aus der ganzzahligen Optimierung und der Graphentheorie abgeleitet. Die Effizienz der daraus resultierenden Lösungsverfahren wird verglichen und diskutiert.

Im zweiten Teil der Arbeit behandeln wir Planungsprobleme mit Vorfertigung; ein relativ neues Forschungsthema, das hauptsächlich im Kontext von Constraintprogrammierung untersucht wurde. Vorfertigungsbedingungen bestehen, wenn jeder Auftrag (oder Job), eine bestimmte Menge Rohstoffe entweder produziert oder verbraucht. Ein Produktionsplan ist nur dann zulässig, wenn die gesamte vorrätige Menge Rohstoffe zu keinem Zeitpunkt negativ ist. Anhand von Anwendungen aus der Industrie wird die Notwendigkeit einer neuen Klasse von Planungsproblemen mit Vorfertigung begründet. Die Zielfunktion soll nur von den konsumierenden Aufträgen abhängen, nicht von den produzierenden. Wir untersuchen die Komplexität verschiedener Varianten des Problems mit verschiedenen Zielfunktionen und Zusatzbedingungen.

Acknowledgements

This work was done with the financial support of the Department Optimization of the Fraunhofer Institute for Industrial Mathematics ITWM. I would therefore like to sincerely thank Prof. Dr. Karl-Heinz Küfer, head of the Optimization Department at Fraunhofer ITWM, for believing in my project and for offering me the opportunity to carry out my research within his department.

Furthermore, my deepest gratitude goes to my supervisor, Prof. Dr. Sven Krumke. With his vast academical knowledge, his seemingly inexhaustible ideas and his ever enthusiastic attitude, he has been of great support for the completion of this work.

Also, I would like to thank Prof. Dr. Marco Lübbecke for his time, and his willingness to serve as a co-referee for my thesis.

A special thanks goes to the members of the Optimization Department at the ITWM, who all contributed to create a very productive yet pleasant working atmosphere. Spending this time in their midst allowed me to keep learning all sorts of things. In particular, I would like to express my gratitude to Tobias Fischer, Michael Helmling and Neil Jami for their admirable patience when faced with my countless questions during these three years.

Last but not least, I am grateful to all those kind souls who accepted, or even offered to serve as proofreaders for parts of my thesis. A special thanks to Holger Berthold, Helene Krieg, Alexandre Vieira, Christian Weiß, and to my dear sister Erica.

Contents

Chapter 1

Introduction

1.1 Motivations

The improvement of competitiveness is a permanent concern for all manufacturing companies. While this basic statement has been true for over a century, the way it is implemented is constantly evolving. Prior to the 1950s, the focus of production planning mainly laid on internal manufacturing efficiency [73]. Some of the methods that are still in use today, such as economic order quantity and Gantt charts, date back to this time. Nonetheless, optimization was essentially understood as ordering decisions and inventory control on the shop floor, within a factory. The external conditions of production were not questioned.

This gradually changed in the second half of the twentieth century, due to increasing competition, but also due to the new possibilities offered by major technological advances [89]. Firms were required to take into account more diverse aspects of the supply chain, such as delivery, quality and flexibility [73]. Scheduling no longer affected operations alone, but encompassed the production environment as a whole. For instance, firms became concerned with reducing lead times and dealing with uncertainty. They also aimed to minimize their work in process inventories, that is, the amount of products being processed but not yet finished. This paradigm shift raised the need for tools and methods that address these challenges. Robust scheduling, as well as inventory constrained scheduling, are two specific attempts to answer the new expectations of manufacturing industry.

Motivations for addressing data uncertainty in general, and for robust optimization in particular, can be found in Ben-Tal et al. [10]. The authors considered the standard PILOT4 problem from the Netlib library, and sought to quantify the effect of uncertainty on constraint violation. They concluded

that a variation of only 0.1% on the coefficients of a particular constraint (#372) suffices to occasion a violation of up to 450% of the right hand side. On average, the relative violation amounts to 125%. Since it may be argued that for this particular problem, the coefficients are indeed uncertain, these results potentially make nominal solutions meaningless.

Historically, the first attempt to deal with uncertainty was not robust, but stochastic optimization. It was initiated in the 1950s by Dantzig [34], and its central idea resides in the assumption that the uncertain data follows a known distribution of values. Solutions are then required to meet a set of constraints in the average case, or in a given percentage of all scenarios. Stochastic optimization was extensively studied and applied in the following decennia, yet it comes with a few drawbacks [14]. To start with, it can be very hard to know which distribution is followed by data; in many cases, a distribution is set arbitrarily. Justifying its relevance may be challenging. Moreover, the enumeration of all possible scenarios that capture the chosen distribution may result in an optimization model of intractable size.

To overcome these disadvantages, the concept of robust optimization was introduced by Soyster [82] in the 1970s. His approach aims only for solutions that are feasible for any realization of the uncertain data, as long as they remain in a specified interval. Robust optimization is of particular use for problem instances where infeasible solutions cannot be accepted, even if the scenarios they appear in are highly improbable. Examples of such instances can be found in the building sector; a new construction should be reliable at all times, even during very specific climate conditions that only occur with a 1% or 0.1% probability. In the case of manufacturing industry, production delays or material shortage may incur prohibitive costs. Moreover, the nature of the product itself may justify the need for a robust approach; pharmaceutical or perishable products, for instance.

Scheduling with inventory constraints was introduced around the turn of the century by Neumann and Schwindt [71]. It was originally thought of as a generalization of resource-constrained project scheduling, where a cumulative resource can be replenished or consumed from a stack by the completion of activities. The same concept was later extended to the field of machine scheduling by Briskorn et al. [21]. Surprisingly, the literature on inventory-constrained scheduling remains relatively scarce at the present day, and we believe it still has unexplored opportunities to offer. The approach used by Briskorn et al. [21] is based on applications from the logistics sector. Machines represent warehouses, where trucks, which can be seen as the jobs, deliver or

pick up goods that need to be brought elsewhere. In their case, producing and consuming jobs are of equal significance. In contrast, our starting point is an application from the manufacturing industry where producing jobs are to *prefabricate* a set of intermediate products. These are then, at a later stage, required by the consuming jobs to be transformed into the final products. Since in this setting, only the consuming jobs are of actual importance to the objective function, this application results in different mathematical formulations and solution approaches.

The aim of this thesis is to further develop the concepts of robust scheduling and material prefabrication. Using the classical problems of buffer allocation and inventory constraints as starting points, we build models that adapt or extend the ideas behind both concepts, and propose different algorithms to solve them. Particular attention is paid to the complexity and practical efficiency.

1.2 Outline of the Thesis

This thesis is organized in four parts and contains six chapters. Part I, the introductory part, presents the fundamentals of this work. Part II is concerned with the nominal and the robust buffer allocation problem, while scheduling with prefabrication is dealt with in Part III. Conclusions, and an outlook on further research, are provided in Part IV.

The notational conventions adopted throughout this work are presented in **Chapter 2**. We also provide theoretical background in a few key areas, in particular with regard to previous results which are used in this thesis. The discussed areas are integer programming, robust optimization, graph theory and scheduling theory.

Chapter 3 is concerned with robust buffer allocation. We pay particular attention to the question of how uncertainty can appear in the standard buffer allocation model. A recurrent problem with robust optimization is that of *over-conservatism*. Indeed, since every scenario must be covered by the final solution, the latter may have an unnecessarily "bad" objective value. To overcome this drawback, we consider a Γ-robust approach. This method allows to define a parameter Γ that represents the degree of robustness of the problem. The resulting model is split following a generative-evaluative approach. We then show that the evaluating part of the solution process can be reduced to a longest

path problem in a graph. Different algorithms are eventually implemented, compared and discussed.

One of the main issues raised when considering robustness for the buffer allocation problem is that of computational tractability. Therefore, in **Chapter 4** we investigate new techniques to speed up buffer allocation. These techniques can be applied to the model both with and without robustness. For the sake of simplicity, we focus on the nominal model in this chapter. We note that fast combinatorial cuts and lower bounds have been described in literature, yet may only be applied to a special case. Firstly, we adapted these cuts and bounds so as to be applicable in the general case. Secondly, a different methodology is proposed for the computing of lower bounds. We show that this method can be further extended to compute solutions for the Buffer Allocation Problem (BAP). The computational efficiency of the resulting bounds and algorithms are demonstrated and compared.

Chapter 5 addresses the concept of scheduling with prefabrication. We justify the need for this new class of scheduling problems with inventory constraints by giving applications from the manufacturing industry. The problem is formally defined and a model is provided. While some of the results from literature can be transposed to our new setting, we show that this is not generally the case. We then investigate the problem in terms of complexity for a wide range of objective functions and constraints. As scheduling with prefabrication turns out to be hard in many cases, we restrict the scope of our study to the one machine case with one type of material.

1.3 Credits

All the contents presented in this thesis are the fruits of joint work with my supervisor, Prof. Dr. Sven O. Krumke.

Moreover, the results in Chapter 5 were achieved in cooperation with my colleagues at Fraunhofer ITWM, Dr. Michael Helmling, Dr. Sebastian Velten and Dr. Christian Weiß.

Part I

Fundamentals

Nomenclature

General Notations

$\mathcal{NP}$	Class of problems that are verifiable in polynomial time
$\mathcal{P}$	Class of problems that are solvable in polynomial time
$\mathbb{N}$	Set of natural numbers
$\mathbb{Z}$	Set of integer numbers
$\mathbb{R}$	Set of real numbers
$\mathbb{R}^n$	Set of n-dimensional real valued vectors
$\mathbb{R}^{n \times m}$	Set of $n \times m$-dimensional real valued matrices

Buffer Allocation

$\mathcal{S}$	Set of all stations
$\mathcal{W}$	Set of all workpieces
B_s	Largest allowed size for the buffer behind station s
TH^*	Goal throughput to be achieved by a solution
S	Total amount of stations
W	Total amount of workpieces
W_0	Amount of workpieces used for the warm-up phase
$F_{s,w}$	Finishing time of workpiece w on station s

$d_{s,w}$	Processing time of workpiece w on station s
$S_{s,w}$	Starting time of workpiece w on station s
β	Vector of integers representing a buffer allocation, or distribution
$\mathscr{S}$	Schedule, represented by a set of starting and finishing times
$\mathscr{S}(\mathcal{S}, \mathcal{W}, \beta)$	Fastest no-wait schedule defined by a set of stations $\mathcal{S}$, of workpieces $\mathcal{W}$ and a distribution β
$\mathcal{L}$	Flow line, represented by a set of stations
$\mathcal{L}[a, b]$	Sub-line of $\mathcal{L}$, consisting of stations $\{a, a+1, \ldots, b\}$
$\mathrm{TH}(\mathscr{S}, \mathcal{L})$	Throughput of line $\mathcal{L}$ with schedule $\mathscr{S}$

Graph Theory

A	Set of all arcs (i.e. constraints) of a graph
V	Set of all vertices (i.e. starting or finishing dates) of a graph
$G(V, A)$	Graph defined by the vertices in V and the edges of A
$G(k)$	k^{th} subgraph of a robust graph $G(\Gamma)$
$a^i_{s,w}$	Arc of type i representing the corresponding constraint related to station s and workpiece w
$p^i_{s,w}$	Weight of arc $a^i_{s,w}$

Robust Optimization

$\mathcal{U}$	Uncertainty set
Γ	Maximum amount of uncertain data entries
$\overline{d_{s,w}}$	Nominal value for the processing time of workpiece w on station s
$\widetilde{d_{s,w}}$	Maximum allowed deviation for the processing time of workpiece w on station s

$\epsilon_{s,w}$	Actual deviation for the processing time of workpiece w on station s

Classical Scheduling Theory

$\mathcal{J}$	Set of all jobs
$\mathcal{M}$	Set of all machines
m	Total amount of machines
n	Total amount of jobs
J_j	Job of index j
M_i	Machine of index i
d_j	Due date of job J_j
p_j	Processing time for job J_j
r_j	Release date of job J_j
w_j	Weight for the completion of job J_j
$\mathcal{H}^{\text{reg}}$	Set of regular objective functions
L_j	Lateness of job J_j
U_j	Unit penalty of job J_j
σ	Solution sequence that contains each job J_j exactly once
$\sigma(k)$	Job scheduled in slot k by solution σ
C_j	Completion time of job J_j in a feasible schedule σ
S_j	Starting time of job J_j in a feasible schedule σ

Scheduling with Inventory Constraints

$\mathcal{G}$	Set of all goods

$\mathcal{J}^{g+}$	Set of all jobs that produce goods of type g
$\mathcal{J}^{g-}$	Set of all jobs that consume goods of type g
δ_j^g	Netto quantity of material g produced by job J_j
Δ_j^g	Cumulated stock of material g at the completion time of job J_j
$B_{g,m}^{\mathrm{c}}$	Consumer block of resource g on machine m
$B_{g,m}^{\mathrm{p}}$	Producer block of resource g on machine m

Chapter 2

Definitions and Preliminaries

In this chapter, we briefly review some background knowledge and basic literature about the main topics that are addressed in this work. In particular, we give a few fundamentals on integer programming, robust linear optimization, graph theory and scheduling theory. While our review is not intended to be exhaustive, it highlights the key concepts referred to in this research.

2.1 Linear and Integer Programming

Linear and integer programming are fundamental concepts in the field of optimization, in the sense that many problems can be modelled as Linear Programs (LPs) or Mixed-Integer Linear Programs (MILPs). Moreover, a great deal of the methods that are used to solve more general problems make use of techniques from linear or integer programming. Therefore, we briefly present some of the main concepts and methods in this section. For a more thorough introduction to this wide subject, we refer to the books by Schrijver [79] and Nemhauser and Wolsey [70].

A *Linear Program*, in short form LP, is an optimization problem that is concerned with finding a vector $x \in \mathbb{R}^n$ that minimizes a linear functional and such that x satisfies a set of linear constraints. In *standard form*, an LP is written as follows:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax = b \\ & x \geq 0 \end{aligned} \tag{2.1}$$

for some $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. It can be showed that the formulation in (2.1) is polynomially equivalent to the following linear program:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b \end{aligned} \tag{2.2}$$

This simpler form is preferred throughout this thesis. Note that any maximization problem may be written as a minimization problem, and vice versa, since

$$\max\{c^T x : Ax \leq b\} = -\min\{-c^T x : Ax \leq b\} \tag{2.3}$$

Before moving on to the solution of such a problem, a concept that is worth mentioning is that of *LP-duality*.

Definition 1: The linear programming dual of an LP in standard form, as written in (2.2), is given by the following formulation:

$$\begin{aligned} \min \quad & b^T y \\ \text{s.t.} \quad & A^T y \leq c \end{aligned} \tag{2.4}$$

The original form is then called the *primal* problem.

Note that the dual can always be computed for any LP in standard form. A dual to the formulation (2.4) itself can be computed; doing so results in the primal form again. As the following theorem shows, LP-duality is useful both for solving an LP and for proving the optimality of a solution x^*.

Theorem 1 (Duality theorem): For $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, assume that both sets $P = \{x : Ax \leq b\}$ and $Q = \{y : y^T A = c^T,\ y \geq 0\}$ are nonempty. Then the following statements hold:

- if $x \in P$ and $y \in Q$, then $c^T x \leq b^T y$ (*weak duality*)
- $\max\{c^T x : x \in P\} = \min\{b^T y : y \in Q\}$ (*strong duality*) □

This fundamental theorem lays the basis for the *Simplex Algorithm*, a very commonly used method for the solving of LPs. Although its worst-case complexity is exponential, it turns out to be efficient in practice. For a detailed introduction to Computation Theory, we refer to Aho et al. [1], Garey and Jonhson [42] and Grötschel et al. [48].

We speak of an *integer program*, or IP, when the solution vector x of an LP is further constrained to be integral. Similarly to LPs, an IP can be written as follows:

$$\begin{aligned} \min \quad & c^T x && (2.5)\\ \text{s.t.} \quad & Ax \leq b && (2.6)\\ & x \geq 0 && (2.7)\\ & x \in \mathbb{Z}^n && (2.8) \end{aligned}$$

This class of problems, in contrast to linear programs, can be proved to be $\mathcal{NP}$-hard. Hence, unless $\mathcal{P} = \mathcal{NP}$, one cannot hope to find a polynomial time algorithm to compute an optimum for an IP. Moreover, the Simplex Algorithm is not suitable for this case, as it does not compute integer solutions. The intuitive idea of relaxing the integrality constraint (2.8), and rounding the corresponding solution to the nearest integer, is erroneous; the resulting solution may not be feasible, and is not optimal in general.

The following theorem lays the foundations for an iterative method that can be used to solve IPs.

Theorem 2: Consider the following LP:

$$\begin{aligned} \min \quad & c^T x && (2.9)\\ \text{s.t.} \quad & Ax \leq b \\ & x \geq 0 \end{aligned}$$

Assume that $x^* \in \mathbb{R}^n$ is optimal for (2.9). If x^* is integer, then it is also optimal for the corresponding IP (2.5). □

Consider the following procedure:

1. solve the LP-relaxation (2.9) of the original IP
2. if solution x^* is not integer, add a new constraint to (2.9) that is violated by x^*, but remains valid for any feasible integer solution
3. repeat the previous steps until x^* is integer, and therefore optimal by Theorem 2

Constraints as defined in Step 2 are called *cutting-planes*, or *cuts*. It can be shown that if they are wisely chosen, an optimum is computed in a finite number of steps. Different generic methods for the computation of cutting planes were

described, e.g. Gomory's cutting-plane algorithm. However, specific methods often perform better depending on the structure of the problem.

In this work, we utilize cut-generation approaches to solve *mixed-integer programs*; a generalization of IPs that involves both real value and integer variables. We apply a decomposition technique, *Benders' Decomposition*, to separate the integer variables from the real value variables. The resulting sub-problems are then used to produce cuts, and solve the original problem in a similar way. For a more detailed approach on Benders' decomposition method, see Benders [13] and Geoffrion [43].

2.2 Robust Linear Optimization

While robust optimization is notorious for "dealing" with uncertainty, the way this is exactly done can be understood in different manners. Literature on this subject is dense, and many different concepts, with their benefits and drawbacks, have been described. In this section, we motivate our choice for the concept of strict robustness with Γ-parameter, and introduce its main principles. For an in-depth treatment on robust optimization, the reader is referred to the book by Ben-Tal et al. [10] and to the surveys of Beyer and Sendhoff [16] and Goerigk and Schöbel [46].

Robust optimization is used when a set of input data is considered to be uncertain, and the solution is required to be *immune* against this uncertainty. In other words, whatever the true outcome of the data is, the solution may not become "too bad". For this purpose, an *uncertainty set* is defined, that is, a set of scenarios that should be covered by the robustness. It is denoted by $\mathcal{U}$, and may take many forms, discrete or continuous. How the expression "too bad" is understood differs according to the different concepts. A straightforward idea, the so-called *strict* robustness, is to assume the following statement: A solution vector x is considered to be acceptable only if it is feasible for each scenario contained in $\mathcal{U}$.

In the case of linear robust optimization, with the same notations as in Section 2.1, the *robust counterpart* can be given by:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \leq b, \quad \forall A \in \mathcal{U} \end{aligned} \tag{2.10}$$

In this case, only the matrix A is uncertain, and $\mathcal{U} \subset \mathbb{R}^{n\times m}$. Note that the vectors $b \in \mathbb{R}^m$ and $c \in \mathbb{R}^n$ are generally subject to perturbations as well. However, it can be shown [10] that no generality is lost when assuming that only the matrix $A \in \mathbb{R}^{m\times n}$ is uncertain. We also assume that the uncertainty is affine linear, hence we may write:

$$\mathcal{U} = \{A \in \mathbb{R}^{m\times n} : \exists Z \in \mathcal{Z},\ A = \overline{A} + \widetilde{A}Z\} \tag{2.11}$$

where $\widetilde{A} \in \mathbb{R}^{m\times n}$, $\overline{A} \in \mathbb{R}^{m\times n}$ are some given matrices, and $\mathcal{Z} \subset \mathbb{R}^{n\times m}$. The $\overline{a_{i,j}}$ entries of $\overline{A}$ are called *nominal values* while Z is the *primitive uncertain matrix*.

According to the definition that is chosen for the *uncertainty region* $\mathcal{Z}$, the robust counterpart (2.10) may be easy to solve or not. It should however be noted that a robust formulation is always at least as hard to solve as the original *nominal* problem, that is, without uncertainty. Since in this thesis, we deal with problems that are $\mathcal{NP}$-hard in the nominal case, we opt for a simple yet realistic definition of $\mathcal{U}$, the so-called *box uncertainty*.

In this setting, the uncertainty has the following structure:

$$\mathcal{Z} = \{Z \in \mathbb{R}^{n\times m} : \forall(i,j), \quad |Z_{ij}| \leq r\} \tag{2.12}$$

for a given $r \in \mathbb{R}$. The robust counterpart then becomes:

$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & \overline{A}x + r\|\widetilde{A}x\|_1 \leq b \end{aligned} \tag{2.13}$$

This formulation is considered to be among the most tractable, computationally speaking, yet also excessively conservative. Indeed, the solution x is required to remain feasible when each entry of Z takes its worst possible value. This may result in an objective value that is of considerably lower quality than it would be for the certain problem.

The concept of Γ-*robustness* is an extension of robust linear optimization that addresses the problem of over-conservatism by only allowing up to $\Gamma \in \mathbb{N}$ entries of Z to take nonzero values simultaneously. The uncertainty set is then written:

$$\mathcal{U}(\Gamma) = \left\{A \in \mathbb{R}^{m\times n} \colon a_{ij} = \overline{a_{ij}} + \widetilde{a_{ij}}Z_{ij}, \quad Z \in \mathcal{Z}(\Gamma)\right\} \tag{2.14}$$

$$\mathcal{Z}(\Gamma) = \left\{ Z \in \mathbb{R}^{m \times n} \colon \forall (i,j),\ |Z_{ij}| \leq 1,\ \sum_{i=1}^{n} \sum_{j=1}^{m} |Z_{ij}| \leq \Gamma \right\} \tag{2.15}$$

Although the resulting model is generally harder to solve than the worst-case approach, it has two advantages over classical strict robustness. First, the optimal solution can be expected to be less conservative. Secondly, the level of robustness is not fixed, and may be chosen and adjusted by the end user. More about the concept of Γ-robustness can be found in Bertsimas and Sim [14, 15].

2.3 Graph Theory

Graph theory is a field of optimization that provides powerful frameworks and tools for the modelling and solving of discrete problems. As graph representations and algorithms are occasionally utilized throughout the thesis, this section presents a few basics on graph theory in general, and shortest paths in particular. For a complete introduction to graph theory, we refer to Harary [54], Ahuja et al. [2] and Krumke and Noltemeier [61].

Definition 2: A *directed graph* is defined by a quadruplet $G = (V, A, \alpha, \omega)$ where V is a set of *vertices*, A a set of *arcs* such that $V \cap A = \emptyset$ and where $\alpha \colon A \mapsto V$ resp. $\omega \colon A \mapsto V$ denote two mappings that associate a *predecessor* resp. *successor* to each arc.

Often, α and ω are omitted, and the notation for graphs is shortened $G = (V, A)$. An arc a whose predecessor node is u and successor node is v may be written $a = (u, v)$. If such an arc exists, the nodes u and v are said to be *adjacent*. Note that for a directed graph, $(u, v) \neq (v, u)$. An example of a directed graph is given by Figure 2.1.

Definition 3: A graph $G' = (V', A')$ is called a *subgraph* of $G = (V, A)$ if $V' \subset V$, $A' \subset A$ and the following property holds:
For each arc $(u, v) \in A$, if $u \in V'$ and $v \in V'$ then $(u, v) \in A'$.

Definition 4: In a graph $G = (V, A, \alpha, \omega)$, a *directed path*, or simply path, is defined by a finite sequence of arcs $\{a_1, \ldots, a_n\}$ such that for each integer $1 \leq k < n$, $\alpha(a_{k+1}) = \omega(a_k)$. Its length corresponds to the amount of arcs it consists of, that is, n. A *simple path* has the additional restriction that vertices may not be repeated in the sequence.

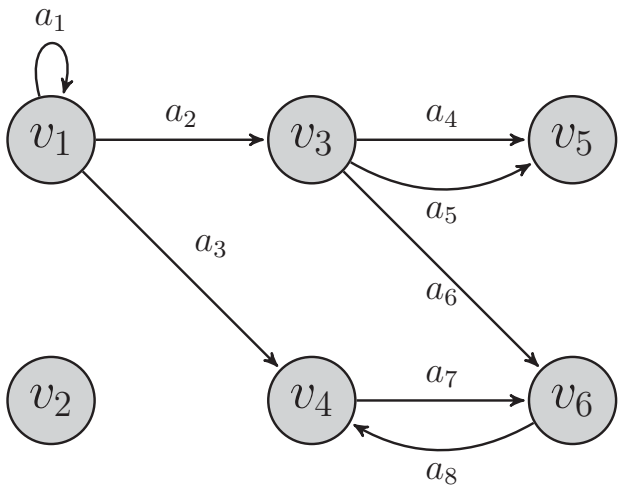


Figure 2.1: Example of a directed graph with vertex set $V = \{v_1, \ldots, v_6\}$ and arc set $A = \{a_1, \ldots, a_8\}$

A path $p = \{a_1, \ldots, a_n\}$ can be referred to as a path from $\alpha(p) = \alpha(a_1)$ to $\omega(p) = \omega(a_n)$. A particular path where $\alpha(p) = \omega(p)$ is called a *cycle*. A common way to prove that a graph $G = (V, A)$ is *acyclic*, that is, it contains no cycles, is to define a *topological order* on G; a value $f(u)$ is associated with each node $u \in V$ under the constraint that if there exists an arc $(u, v) \in A$, then it holds true that $f(u) < f(v)$. Topological orders are generally not unique, and only exist in acyclic directed graphs.

Let $G = (V, A)$ be a directed graph such that each arc $a_i \in A$ has an associated weight $w_i \in \mathbb{R}$. The *shortest path problem* deals with computing a path $P = \{p_1, \ldots, p_k\}$ from a *source* node $s \in V$ to a *sink* node $t \in V$, such that the value of $\sum_{a_i \in P} w_i$ is minimized. The shortest path problem is a well studied polynomial problem; it was historically solved in $\mathcal{O}(|V|^2)$ time by Dijkstra [37] for non-negative weights. The algorithm of Bellman and Ford [9] solves the same problem for any weights in $\mathbb{R}$ in $\mathcal{O}(|V| \times |A|)$ time. More efficient algorithms for the shortest path problem can be found in Ahuja et al. [3] and Thorup [85].

The *longest path problem* can be defined analogously; a path $P = \{p_1, \ldots, p_k\}$ is computed such that $\sum_{a_i \in P} w_i$ is maximized. It can be shown that the longest path problem is $\mathcal{NP}$-hard using a reduction from the standard Hamiltonian path problem. Nonetheless, computing a longest path in a graph G with weights (w_i) is strictly equivalent to computing a shortest path in G with weights $(-w_i)$. Hence, for graphs that do not contain cycles of strictly positive weight, longest paths can be computed in polynomial time using the same algorithms.

2.4 Scheduling Theory

Scheduling theory can be seen as a very vast theoretical framework that essentially serves the purpose of modelling. It is often used for applications where operations are to be allocated to scarce resources, e.g. in manufacturing industry, or project scheduling. It comes with specific notations and numerous fundamental results. This section presents some prerequisites to inventory constrained scheduling. The reader is referred to Jackson [57], Kan [58], Pinedo [75] and Brucker [22] for a more exhaustive introduction to scheduling theory.

Let $\mathcal{M}$ denote a set of *machines* $\{M_1, \ldots, M_m\}$ and $\mathcal{J}$ denote a set of *jobs* $\{J_1, \ldots, J_m\}$. A job J_i consists of m distinct *operations* (i, j). A processing time $p_{ij} \in \mathbb{R}$ is given for each operation $(i, j) \in \mathcal{J} \times \mathcal{M}$. In some instances, due dates d_i are also given for the jobs. We assume that at any time, only one job can be scheduled on a given machine M_j, and only one machine may process a job J_i.

Definition 5: The *technical sequence* of a job J_i is given by a sequence $\{M_{j_1}, \ldots, M_{j_{m_i}}\}$ of m_i machines. Analogously, the *organisational sequence* of a machine is a sequence $\{J_{i_1}, \ldots, J_{i_{n_j}}\}$ of n_j jobs.

A *schedule* is defined by a set of *starting times* S_{ij} and *completion times* C_{ij} for each operation (i, j). For a job J_i, we shall use $S_i = \min_j S_{ij}$ and $C_i = \max_j C_{ij}$. Assume that an operation (i, j) always starts processing without delay when both J_i and M_j are available. Then, the combined technical sequences of all jobs and organisational sequences of all machines define a unique schedule.

Note that the completion time of jobs is not the only quality measure for a schedule. Measures may be relevant or not according to the problem that is being studied. Another common measure is given by the *lateness* L_i of the jobs, which corresponds to the difference $C_i - d_i$. Note that latenesses may be negative when a job is completed early. In some cases, the value of the lateness is not so important as the amount of jobs that are late; it may therefore be interesting to compute the *unit penalties* U_i for the jobs. U_i equals 1 if the job has a strictly positive lateness, 0 otherwise

In this work, we adopt the three field notation $\alpha \mid \beta \mid \gamma$ that was introduced by Graham et al. [47] for theoretic scheduling problems. Its basic principles can be summarized as follows:

The α-field describes the *machine environment* of the problem. Common *single stage problems*, that is, problems in which jobs consist of only one operation, are denoted by 1 (single machine problems) or P_m (m parallel identical machines). The most common *multi-stage problems* are classified as follows. In a *job shop problem*, denoted by J_m, a technical sequence is given for each job. The aim is to define optimal organisational sequences. In an *open shop problem*, denoted by O_m, no technical sequence is fixed, yet each job is to be processed on each machine exactly once. A *flow shop problem*, denoted by F_m, is a particular job shop problem in which all technical sequences are identical.

The β-field gives the job characteristics and constraints of the problem. A common example is the indicator "pmtn", preemption, when jobs are allowed to interrupt and possibly resume their processing on a machine. Another example is given by "r_j", which indicates that jobs have *release dates*, before which they may not be processed. Through this thesis, we shall often make use of "inv", which indicates the use of *inventory constraints*, and of notations such as "$p_{ij} = p$", which means that all processing times are equal.

Finally, the γ-field shows the objective function of the problem. In this work, the most relevant objective functions are the following:

- the maximum lateness $L_{\max} = \max_i L_i$ of a schedule
- the weighted sum of tardy jobs $\sum_i w_i U_i$
- makespan $C_{\max} = \max_i C_i$ of a schedule
- the weighted sum of completion times $\sum_i w_i C_i$

For example, $P_3 \mid r_i,\ p_i = p \mid L_{\max}$ denotes a scheduling problem with three parallel machines and where jobs have release dates and equal processing times. Its objective is to determine a schedule that minimizes the maximum lateness value.

The following class of objective functions plays an important role in the determination of complexity classes.

Definition 6: An objective function:

$$\begin{aligned} h\colon \mathbb{R}^n &\mapsto \mathbb{R} \\ (C_1, \ldots, C_n) &\to h(C_1, \ldots, C_n) \end{aligned}$$

is called a *regular objective* if h is non-decreasing with respect to any C_i. The set of regular objective functions is denoted by $\mathcal{H}^{reg}$.

Part II

Buffer Allocation in Flow Lines

Chapter 3

Robust Buffer Allocation

In this chapter, we consider the problem of allocating a minimal amount of buffer slots to the servers of a manufacturing flow line such that a given throughput is achieved. We solve this problem to optimality taking into account any uncertainties during the manufacturing process, while keeping a level of flexibility in the degree of robustness. To this end, we derive a mixed-integer linear program (MILP) to solve the robust counterpart of the problem. Such an approach provides exact solutions, but notoriously implies large models, and a great computational effort for long lines. We cope with this drawback by applying a decomposition technique, in which the subproblem is reduced to solving of a longest path in an acyclic graph. Since such a path may be computed efficiently, computation times are reduced substantially. We also propose initial bounds to further speed up the process. A numerical study is carried out to compare the different derived algorithms, as well as to show the impact of the level of robustness on optimal solutions and on computation times.

3.1 Introduction

In the manufacturing industry, many production processes can be modelled as flow lines [23]; in this setting, a set of machines, or *stations*, are placed in a fixed sequence. Jobs, or *workpieces*, which are also in a fixed order, then visit each station to be processed for a given time. As jobs cannot overtake each other, substantial amounts of time are wasted through *blocking* (a workpiece cannot proceed to next station, as it is not idle) and *starvation* (a station remains idle, since no workpiece has yet reached it). In order to counter these unwanted phenomena, a common technique that is utilized in industry [4, 29] is

the use of buffers. Each time a workpiece is blocked, it is temporarily stored in the buffer so as to keep its current station free for the next coming workpieces. Buffering can reduce both blocking and starvation, and thus greatly improve productivity [24]. However, this comes at a cost; a direct cost for the buffer slot itself, as well as a cost that follows from the increased work-in-process induced by the buffer. How undesired a large work-in-process inventory may be in applications is best described by Conway et al. [30]. This leads to the Buffer Allocation Problem (BAP); in its primal formulation, it is concerned with the minimization of the amount of buffers that suffices to achieve a given goal *throughput*, that is, the amount of processed jobs per time unit. In its dual formulation, the throughput is maximized using a given amount of buffers. Macgregor Smith [64] presents a joint optimization of the buffer amount and the work-in-process. In this chapter, only the primal problem is addressed. We also assume that the supply of workpieces at the first station is unlimited. A model with limited supply was studied by Weiss et al. [86].

The BAP was extensively studied in literature over the last few decades, and the reader can refer to the reviews of Park [74] and Gershwin and Schor [45] for publications prior to 1998, and to the study of Weiss et al. [87] for recent literature. Motivations for the BAP are given by Dallery and Gershwin [33] and Burman et al. [24], who show the effect of buffering on system efficiency, while Conway et al. [30] highlight the benefits of a limited in-process inventory. Concerning solution approaches, Alfieri et al. [6] and Amiri and Mohtashami [7] advocate simulation based approaches. Demir et al. [35] note that the BAP is generally solved by an iterative procedure: a generative method is applied to produce an objective value for a given input, say a buffer allocation. This objective is then passed to an evaluative method, analytical or based on simulation, that provides a performance measure, such as the line's throughput for the given buffers. The process is repeated until a stopping condition is met. A time-efficient exact method for long lines is given by Weiss and Stolletz [88]: they optimally solve a Mixed-Integer Linear Program (MILP) described by Matta [66] using fast combinatorial cuts designed by Codato and Fischetti [28], in combination with lower bounds suggested by Levantesi et al. [63] to further speed up the process.

Nonetheless, supply chains in the industrial production are naturally subject to uncertainty, both due to external causes such as material shortage, fluctuations in supply or demand, as well as to difficulties on the line itself, i.e. server breakdown. Actual processing times therefore cannot be exactly known in advance, and the optimal buffer allocation for a given set of processing times may be substantially worse, if not infeasible, for a slightly different set [12].

Different authors [88] achieve some kind of *robustness* by running simulations on a sufficiently large sample of workpieces. The main difficulties for this approach are well-known [14, 15]: One generally does not know the exact distribution followed by the data in practice, which makes the generation of accurate scenarios a hard task. Moreover, the size of the optimization model may quickly become intractable when great amounts of scenarios are to be considered. An alternative was given by Soyster [82], followed by Ben-Tal and Nemirovski [11, 12], who proposed a framework to construct solutions that are feasible not only for given scenarios, but for a whole set of values the data may take. Their approach raises the problem of (over-)conservatism: a solution that is robust against any uncertainties has, in many cases, an unsuitable objective value for practice. Bertsimas and Sim [14] defined the concept of Γ-robustness, whose main idea is to allow the user to chose the level of conservatism, by means of parameters, while being still computationally tractable. Their concept was applied to different fields such as communication network design (see Koster and Kutschka [59], Koster et al. [60]), as well as portfolio decision analysis (see Fliedner and Liesiö [39]). Our goal throughout this work is to compute robust solutions in the sense of Bertsimas and Sim for the BAP as efficiently as possible.

This chapter is organized as follows: The robust optimization model is defined and written down as a generative and an evaluative method in the next section. The evaluative method is proved to be reducible to a longest path problem in a graph in Section 3.3. In Section 3.4, a second algorithm is derived by rewriting both generative and evaluative methods as an MILP. Section 3.5 presents different ways to substantially increase the computational effort for those methods, including initial bounds. Finally, numerical experiments showing the method's efficiency and comparing both algorithms can be found in Section 3.6.

3.2 Definition of the Model

3.2.1 The Buffer Allocation Problem in the Nominal Case

In the Buffer Allocation problem (BAP), we are given a flow line $\mathcal{L}$ of S servers $1, \ldots, S$ where W workpieces $1, \ldots, W$ are processed sequentially. We assume that the incoming flow of workpieces is unlimited, that is, a new workpiece starts processing on the first station as soon as it is idle. This setting is depicted on Figure 3.1.

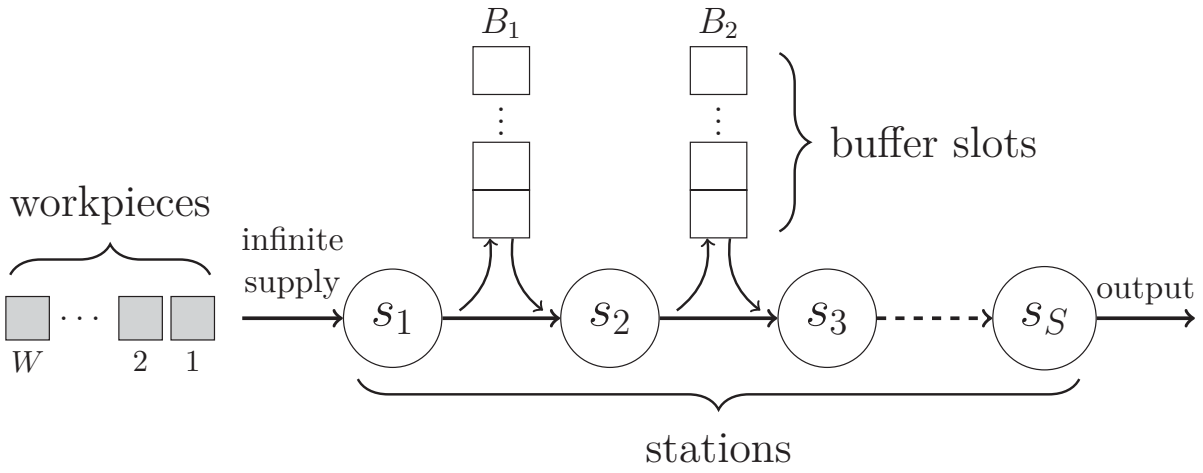


Figure 3.1: Diagram of a flow line with buffering after process

Two equivalent approaches exist for the BAP: some authors choose to maximize the throughput such that the total amount of buffers is bounded (*dual problem*), whereas others opt for a minimization of the buffers while maintaining a given performance (*primal problem*). In this work, we only solve the primal problem. The performance of a line is measured by its throughput, that is, the amount of workpieces that were processed over the time needed to process them. It can be written:

$$\mathrm{TH} = \frac{n}{f(w_n) - s(w_1)} \tag{3.1}$$

where n is the total amount of workpieces, w_1 and w_n denote the first and last workpiece respectively, and the mappings $s(\cdot)$ and $f(\cdot)$ represent the starting and finishing times of a workpiece.

To start with, we give a Mixed-Integer Linear Program (MILP) formulation of the problem without robustness (nominal case). It was first introduced by Matta [66], based on the methodology developed by Chan and Schruben [27]. It was then rewritten by Weiss and Stolletz [88], whose notations we shall use in this work:

$$\text{Minimize} \sum_{s=1}^{S-1} \sum_{b=0}^{B_s} b \cdot Y_{s,b} \tag{BAP}$$

$$\text{s.t.} \quad F_{S,W} \leq \frac{W}{\mathrm{TH}^*} \tag{3.2}$$

$$S_{s,w} - F_{s,w} \leq -d_{s,w} \qquad \forall s,\ \forall w \tag{3.3}$$

$$F_{s,w} - S_{s+1,w} \leq 0 \qquad \forall s \leq S-1,\ \forall w \tag{3.4}$$

$$F_{s,w} - S_{s,w+1} \leq 0 \qquad \forall s,\ \forall w \leq W-1 \tag{3.5}$$

$$S_{s+1,w} - F_{s,w+b} \leq M \cdot (1 - Y_{s,b}) \qquad \forall s \leq S-1,\ \forall b,\ \forall w \leq W-b \tag{3.6}$$

$$\sum_{b=0}^{B_s} Y_{s,b} = 1 \qquad \forall s \leq S-1 \tag{3.7}$$

$$Y_{s,b} \in \{0,1\} \qquad \forall s \leq S-1,\ \forall b \tag{3.8}$$

The integers S and W stand for the amount of servers and workpieces, respectively. Constants B_s and TH^* represent the maximum amount of buffer slots allowed behind server s and the goal throughput of the line. The input of the problem is given by the processing time values $d_{s,w}$. The output binary variables $Y_{s,b}$ represent whether or not there are exactly b buffer spaces behind server s. The idea of this formulation is to precisely measure time by introducing, for each server and each workpiece, a starting date $S_{s,w}$ and a finishing date $F_{s,w}$. Constraint (3.2) ensures that the goal throughput is reached, constraints (3.3) state that a workpiece's processing on a server must last at least $d_{s,w}$ time before it is finished. (3.4) and (3.5) warrant that a workpiece must finish on a server before it can move to the next, and that a server must have finished processing the predecessor workpiece for the successor to start processing on it. Finally, the notion of buffer size is translated into Contraints (3.6): on each server s, when workpiece w finishes processing, up to b further workpieces may have their processing carried out and finished on s before it is required that workpiece w moves on to the next server. A large constant integer M, a so-called "big M", can be used in MILPs to express the idea of conditionality while conserving linearity. The idea is that the constraint should apply when the right-hand side equals 0, yet should become superfluous in case the right-hand side equals M. For this purpose the value of M should be "sufficiently large". A possibility (Weiss and Stolletz [88]) is given by:

$$M = \max_{s,w} d_{s,w} \cdot \max_{s} B_s \tag{3.9}$$

3.2.2 Definition of the Uncertainty Set

In order to introduce robustness in the nominal model, we now assume that the $d_{s,w}$ are unknown, but may only take values from an given uncertainty

set $\mathcal{U}$, which is to be specified. Our approach is as follows: we define for each nominal processing time $\overline{d_{s,w}}$ an additional value $\widetilde{d_{s,w}} \geq 0$ representing a maximum allowed deviation of the processing time from the nominal value. These deviations may be discrete, or continuous:

- discrete case: we assume that each realized processing time $d_{s,w}$ can only attain one of the values $\left\{\overline{d_{s,w}},\ \overline{d_{s,w}} + \widetilde{d_{s,w}}\right\}$.
- continuous case: $\forall s,\ \forall w, \quad d_{s,w} = \overline{d_{s,w}} + \epsilon_{s,w}$, where $\quad 0 \leq \epsilon_{s,w} \leq \widetilde{d_{s,w}}$

In order to avoid too conservative results, a possible technique [14] consists in defining a parameter $\Gamma \in \mathbb{N}$ representing the maximum amount of variables that may deviate from their nominal value. This idea can be extended to the continuous case when considering the proportions of the deviations instead of the cardinality of deviating variables:

$$\mathcal{U}(\Gamma) = \left\{d \in \mathbb{R}^{S\times W} \colon \left|\left\{(s,w) \colon d_{s,w} > \overline{d_{s,w}}\right\}\right| \leq \Gamma\right\} \quad \text{(discrete case)} \tag{3.10}$$

$$\mathcal{U}(\Gamma) = \left\{d \in \mathbb{R}^{S\times W} \colon \sum_{s=1}^{S}\sum_{w=1}^{W} \frac{\epsilon_{s,w}}{\widetilde{d_{s,w}}} \leq \Gamma\right\} \quad \text{(continuous case)} \tag{3.11}$$

This definition may be simplified by the introduction of new decision variables $Z_{s,w}$, representing the proportions of the maximal deviation $\widetilde{d_{s,w}}$ that is added to the nominal value. In order to apply the Γ-bound for the amount of deviating processing times $d_{s,w}$, we shall define a distinct space $\mathcal{Z}(\Gamma)$ for the Z variables.

$$\mathcal{U}(\Gamma) = \left\{d \in \mathbb{R}^{S\times W} \colon d_{s,w} = \overline{d_{s,w}} + Z_{s,w} \cdot \widetilde{d_{s,w}}, \quad Z \in \mathcal{Z}(\Gamma)\right\} \tag{3.12}$$

$$\mathcal{Z}(\Gamma) = \left\{Z \in \mathbb{R}^{S\times W} \colon \forall (s,w),\ |Z_{s,w}| \leq 1, \quad \sum_{s=1}^{S}\sum_{w=1}^{W} |Z_{s,w}| \leq \Gamma\right\} \tag{3.13}$$

Observation 1: One may assume, without loss of generality, that $\mathcal{Z}(\Gamma) \subset \mathbb{R}_{+}^{S\times W}$. Indeed, a negative $Z_{s,w}$ leads to a smaller $d_{s,w}$ value, which in turn can never lead to a higher $F_{S,W}$. Although negative $Z_{s,w}$ values may reduce the total throughput by decreasing the value of F_{S,W_0}, we consider that this effect is undesirable and do not wish to make the model robust against changes in the warm-up phase.

Observation 2: One may then assume, without loss of generality, that $\mathcal{Z}(\Gamma) \subset \{0,1\}^{S\times W}$. Indeed, the $Z_{s,w}$-variables define a bounded polyhedron, thus, it is sufficient to consider its extreme points.

This restriction makes sense for the Buffer Allocation Problem (BAP) in particular, since one may consider, for instance, that it is very unlikely that more than Γ machines break down at the same time.

3.2.3 Formulation of the Robust Counterpart

Having defined the uncertainty set $\mathcal{U}$, the Robust Counterpart (RC) of an uncertain optimization problem then follows a min-max approach. In the present case, for a BAP with uncertain processing times, it consists in minimizing the amount of buffer slots such that the obtained solution is feasible for any of the possible realizations of $d \in \mathcal{U}$. These are finitely many in the light of Observation 2. An MILP-formulation of the RC is then obtained by replacing constraints (3.3) in BAP by the following set of constraints:

$$S_{s,w} - F_{s,w} \leq -d_{s,w} \quad \forall d \in \mathcal{U},\ \forall s,\ \forall w \tag{3.14}$$

Note that this reformulation of BAP is of purely theoretical nature due to the intractable amount of constraints of type (3.14). Due to the Γ-bound on the amount of deviating processing times, one cannot replace each $d_{s,w}$ by its corresponding $\overline{d_{s,w}} + \epsilon_{s,w}$ value. Therefore, the total number of constraints (3.14) amounts to $\binom{S \cdot W}{\Gamma}$ [possible d realizations in $\mathcal{U}$] times $(S \cdot W)$ entries for each vector.

To cope with this problem, we apply the idea behind Bender's Decomposition and separate the integer Y buffer variables from the real $S_{s,w}$, $F_{s,w}$ and $d_{s,w}$ values. Namely, an integer *master solution* Y is computed first (by means of RC-Master). Then, for some given $d \in \mathcal{U}$, a sub-problem (see RC-Sub) computes the corresponding optimal $S_{s,w}$ and $F_{s,w}$ dates. Finally, the obtained throughput is compared with TH^*.

The resulting algorithm is depicted on Figure 3.2. Note that this iterative procedure can be applied indifferently to the nominal or to the robust problem. In Chapter 4, the same decomposition is applied again to a different instance of the problem.

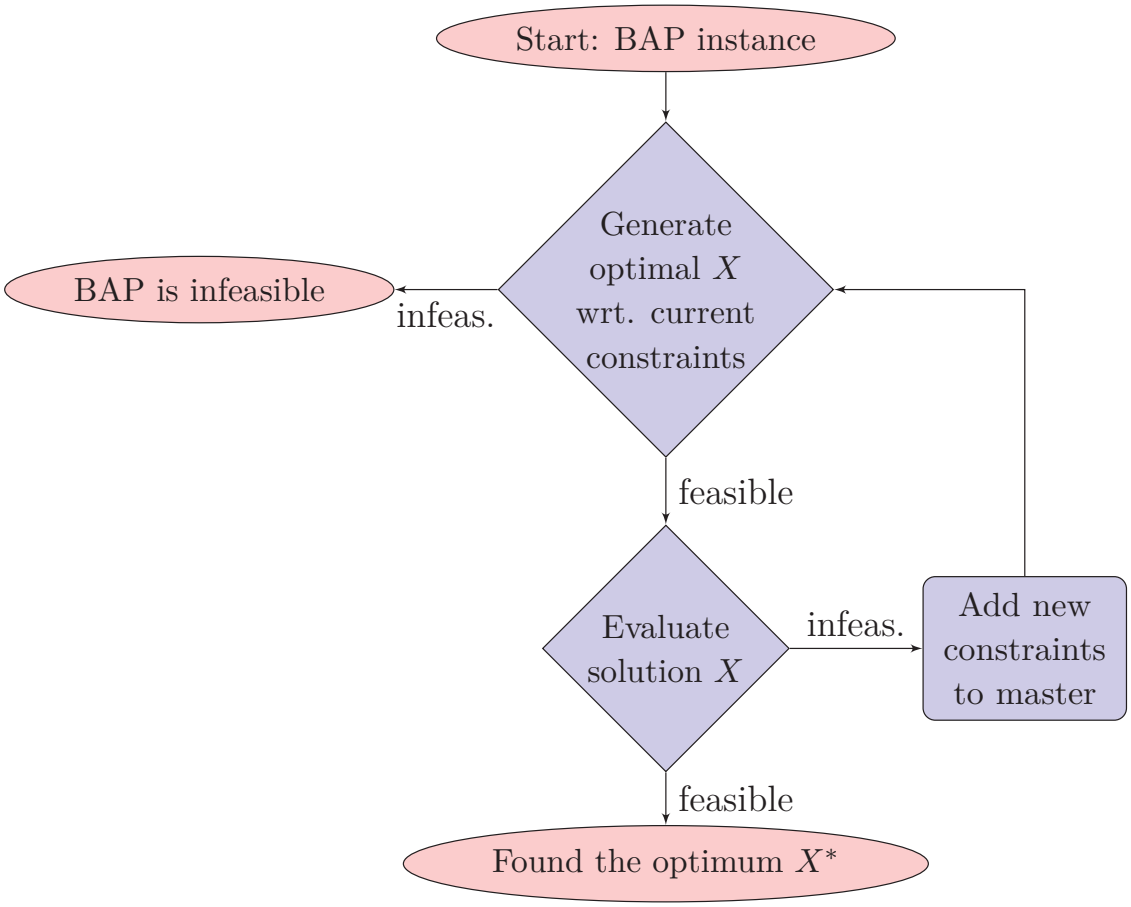


Figure 3.2: Illustration of a common iterative generation/evaluation solution process

In case the throughput is insufficient, an efficient combinatorial cut is given by (Codato and Fischetti [28]):

$$1 \leq \sum_{s=1}^{S-1} \sum_{b=b_s+1}^{B_s} Y_{s,b} \tag{3.15}$$

where b_s denotes the currently used buffer capacity. The motivation behind their cut is that the current buffer allocation was insufficient. Assuming that a decrease of buffers can only worsen the throughput, it is necessary to increase at least one buffer for the goal performance to be achieved. Yet for this method to work efficiently, it is required to be able to select a scenario $d \in \mathcal{U}$ that induces a minimal throughput (i.e. a maximal finishing date $F_{S,W}$). This algorithm can be formulated as the following program:

solve:

$$\text{Minimize} \sum_{s=1}^{S-1} \sum_{b=0}^{B_s} Y_s \qquad \text{(RC-Master)}$$

$$\text{s.t.} \quad \sum_{b=0}^{B_s} Y_{s,b} = 1 \qquad \forall s \leq S-1 \qquad (3.16)$$

$$Y_{s,b} \in \{0,1\} \qquad \forall s \leq S-1,\ \forall b \qquad (3.17)$$

until:

$$\max_{d \in \mathcal{U}} \{F_{S,W}(Y,d)\} \leq \frac{W}{\mathrm{TH}^*} \qquad (3.18)$$

where:

$$F_{S,W}(Y,d) = \text{Minimize } F_{S,W} \qquad \text{(RC-Sub)}$$

$$\text{s.t.} \quad S_{s,w} - F_{s,w} \leq -d_{s,w} \qquad \forall s,\ \forall w \qquad (3.19)$$

$$F_{s,w} - S_{s+1,w} \leq 0 \qquad \forall s \leq S-1,\ \forall w \qquad (3.20)$$

$$F_{s,w} - S_{s,w+1} \leq 0 \qquad \forall s,\ \forall w \leq W-1 \qquad (3.21)$$

$$S_{s+1,w} - F_{s,w+b} \leq M \cdot (1 - Y_{s,b}) \qquad \forall s \leq S-1,\ \forall b,\ \forall w \leq W-b \qquad (3.22)$$

In this three-stage model, the RC-Sub does not require sophisticated solving since the optimum can be computed in $\mathcal{O}(S \times W)$ (see [88]). However, the main difficulty of the previous MIP persists in the present formulation: the maximization of end date $F_{S,W}$ over all $d \in \mathcal{U}$, in the end condition (3.18), implies the testing of exponential amounts of d-vectors. The complete sub-problem, which is required to be efficiently solvable, consists, in the present case, of a two-stage: $\max\{\min\{F_{S,W}\}\}$-problem.

3.2.4 A Note on Warm-up

In literature [76, 65, 90], a warm-up phase, during which the workpieces that are being processed are not taken into account for the computation of the throughput, is generally observed. Its necessity is based on the assumption that a steady-state for the line exists and should be reached before any throughputs may be measured. One therefore defines a number W_0 of workpieces which should be processed first. Only when each of the W_0 workpieces has been fully processed on every server, one may start measuring time to compute the line's

throughput. Under such an approach, Constraint (3.2) would state:

$$F_{S,W} - F_{S,W_0} \leq \frac{W - W_0}{\mathrm{TH}^*} \tag{3.23}$$

Although the relevance of this idea is not questioned here, it should be stressed that in practice, such an assumption poses some serious challenges.

1. Employing a warm-up phase induces some undesired effects in the resolution of the MILP formulation of the BAP. To satisfy constraint (3.23), it is no longer required to keep the completion time of the last workpiece at a minimal level. It is possible to artificially delay workpieces in order to increase the value of F_{S,W_0} as well. In practice, this excessive freedom causes solvers to output solutions that are infeasible if a workpiece must always proceed as soon as it can.

2. Employing a warm-up phase contradicts the assumption by Codato and Fischetti [28] stating that the throughput of a line is monotonously growing with the total buffer. It may very well happen that suppressing a buffer diminishes F_{S,W_0} in such proportions that overall, the throughput rises. As a result, their powerful combinatorial cut (3.15) is inexact in combination with a warm-up phase.

For those reasons, we assume throughout this chapter that $W_0 = 0$. Note that this model is adapted to the general case (with warm-up) in Chapter 4.

3.3 Reduction to a Longest Path Problem

3.3.1 Graph Representation

The key idea to make the evaluative subproblem tractable, is to translate our problem into graph theoretical terms. Accordingly, let us interpret the starting and finishing dates as nodes:

$$V = \{S_{s,w},\ F_{s,w} \quad \forall s \in \mathcal{S},\ \forall w \in \mathcal{W}\}$$

Each constraint in RC-Sub involves exactly two dates (i.e. nodes), and establishes an ordering upon them. Therefore it becomes possible to convert each constraint into an arc, having the earlier date as predecessor node and the later date as successor node. A natural choice would then be to set a weight

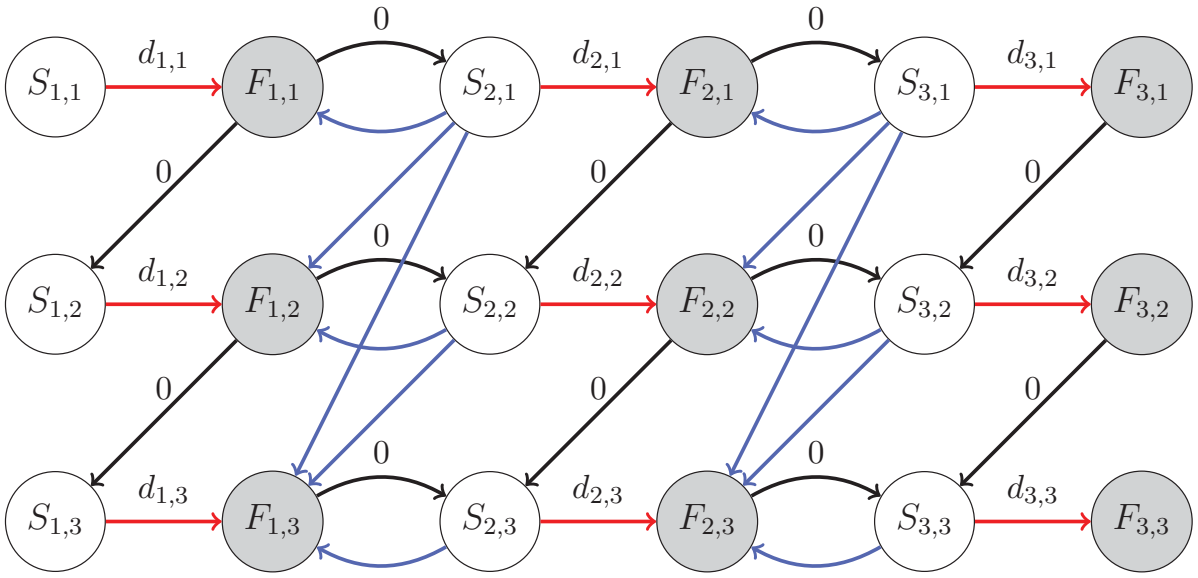


Figure 3.3: Graph representation of a 3-server line with 3 workpieces and undetermined buffers in the nominal case

on each constraint, i.e. on each arc, given by the time that is required to be elapsed between these dates. This corresponds to the opposites of the right hand sides of RC-Sub. Since the subproblem consists of 4 types of constraints: (3.19)-(3.22), we shall distinguish 4 types of arcs, respectively:

$$\begin{aligned} A = &\left\{a^1_{s,w} \quad \forall s,\ \forall w\right\} \\ \cup &\left\{a^2_{s,w} \quad \forall s < S,\ \forall w\right\} \\ \cup &\left\{a^3_{s,w} \quad \forall s,\ \forall w < W\right\} \\ \cup &\left\{a^4_{s,w,b} \quad \forall s \geq 2,\ \forall w,\ \forall b \leq W - w\right\} \end{aligned}$$

For instance, constraint: $S_{3,2} - F_{3,2} \leq -d_{3,2}$ leads to arc $a^1_{3,2} = (S_{3,2}, F_{3,2})$ with weight: $p^1_{3,2} = d_{3,2}$. This means, of course, that processing times d must be given, no deviation is yet taken into account.

An example is given by the graph on Figure 3.3. Note that when no master solution is given beforehand, and thus no $Y_{s,b}$ have been fixed, each arc $a^4_{s,w,b}$ must be present. Each of these arcs has weight: $p^4_{s,w,b} = -M\,(1 - Y_{s,b})$.

This technique was also used by Chan and Schruben [26, 27] to prove a reversibility property for server queues. The graph formulation is an implicit dualization of the original problem. Therefore, it allows us to compute a critical

path of processing times, by means of a longest path in the dual graph, for a given buffer space.

3.3.2 Longest Paths in the Nominal Case

Theorem 3: The previously described graph representation $G(V, A)$ of the line does not contain any cycles of strictly positive weight.

Proof. Let us first consider the subgraph $G'(V, A')$ obtained by removing every arc induced by constraints (3.22) with $b = 0$, that is, $\left\{a^4_{s,w,0} \quad \forall s,\ \forall w\right\}$. Clearly, a topological ordering of G' is given by α where :

- $\alpha\left(S_{1,1}\right) = 1$
- $\alpha\left(F_{s,w}\right) = \alpha\left(S_{s,w}\right) + 1$
- $\alpha\left(S_{s+1,w}\right) = \alpha\left(F_{s,w}\right) + 1$
- $\alpha\left(S_{s,w+1}\right) = \alpha\left(S_{s,w}\right) + 2S$

In G, a trivial cycle $C_{s,w}$ is given by any pair: $\left\{a^2_{s,w}, a^4_{s,w,0}\right\}$. The arc $a^2_{s,w}$ involved in this cycle is the only directed path from $F_{s,w}$ to $S_{s+1,w}$, since $\alpha\left(S_{s+1,w}\right) = \alpha\left(F_{s,w}\right) + 1$. Therefore, the adding of the $a^4_{s,w,0}$-arcs to G' induces no other cycles.
Since for any pair $(s, w) :\ s < S$ we have $p^2_{s,w} = 0$, and since $p^4_{s,w,0} \in \{0, -M\}$, such a cycle fulfills $p(C_{s,w}) \in \{0, -M\}$ and therefore has no strictly positive weight, which proves the claim. □

Corollary 1: It makes sense to mention a "longest weighted path" between two nodes in $G(V, A)$. This path can be computed in linear time by means of a critical path method.

Theorem 4: The minimal time that suffices to reach a particular date, $S_{s,w}$ or $F_{s,w}$, equals the total weight of the longest path in $G(V, A)$ from $S_{1,1}$ to the corresponding date vertex. In particular, the longest path from $S_{1,1}$ to $F_{S,W}$ yields an optimum of RC-Sub.

Proof. We denote $T_{s,w}$ the chosen time $S_{s,w}$ or $F_{s,w}$, and $t_{s,w}$ the corresponding vertex in G. Let $P = (a_1, \ldots, a_n)$ be a path of length $n \in \mathbb{N}$ from $S_{1,1}$ to $t_{s,w}$, with $a_i = (v_i, v_{i+1}) \in A$. By construction, for any arc $a \in A$ from node u to node v, we know that:

$$T_u + p(a) \leq T_v$$

where T_u denotes the minimal time needed to reach node u. Therefore, since P is a path leading to $t_{s,w}$:

$$\begin{aligned} T_{s,w} = T_{v_{n+1}} &\geq T_{v_n} + p(a_n) \\ &\geq 0 + \sum_{i=1}^{n} p(a_i) = \sum_{i=1}^{n} p(a_i) \end{aligned}$$

Let us now denote $P_{\max}$ the longest path in G from $S_{1,1}$ to $t_{s,w}$. To prove the claim, it is left to show that:

$$T_{s,w} \leq \sum_{a_i \in P_{\max}} p(a_i)$$

This inequality follows from the minimality of the dates. Since we assumed that no job is allowed to stand by when it may be processed, there must exist a "critical" path P from $S_{1,1}$ to $T_{s,w}$ such that each constraint is tight. Thus, the total duration to reach $T_{s,w}$ equals the sum of the weights on P. □

Observation 3: When testing a given master solution Y by means of a longest path in $G(V, A)$, the graph may be considerably simplified. In particular, for each node $S_{s,w}$, exactly one element of the set of arcs $\left\{a^4_{s,w,b},\ \forall b\right\}$ equals 0 when $Y_{s,b} = 1$. For any other b, the value of the corresponding arc weight is $-M$. As the existence in G of a path from $S_{1,1}$ to any other node, without using any $a^4_{s,w,b}$-arc, is guaranteed, each arc of weight $-M$ may be removed from G. An example, using the same line as Figure 3.3 is given by Figure 3.4.

3.3.3 Introduction of the Uncertainty

In the previous section, we have shown that for a given master solution Y and a given vector of processing times d, an alternative to solving the subproblem is given by the computation of a longest path from $S_{1,1}$ to $F_{S,W}$ in G. This is not yet of great use since the algorithm given by Weiss and Stolletz [88] also solves the same problem efficiently. Nonetheless, in this section, we shall introduce uncertainty in the model. In other words, we now wish to solve a minimal buffer allocation for any d in the uncertainty set $\mathcal{U}(\Gamma)$ presented in Section 3.2.2.

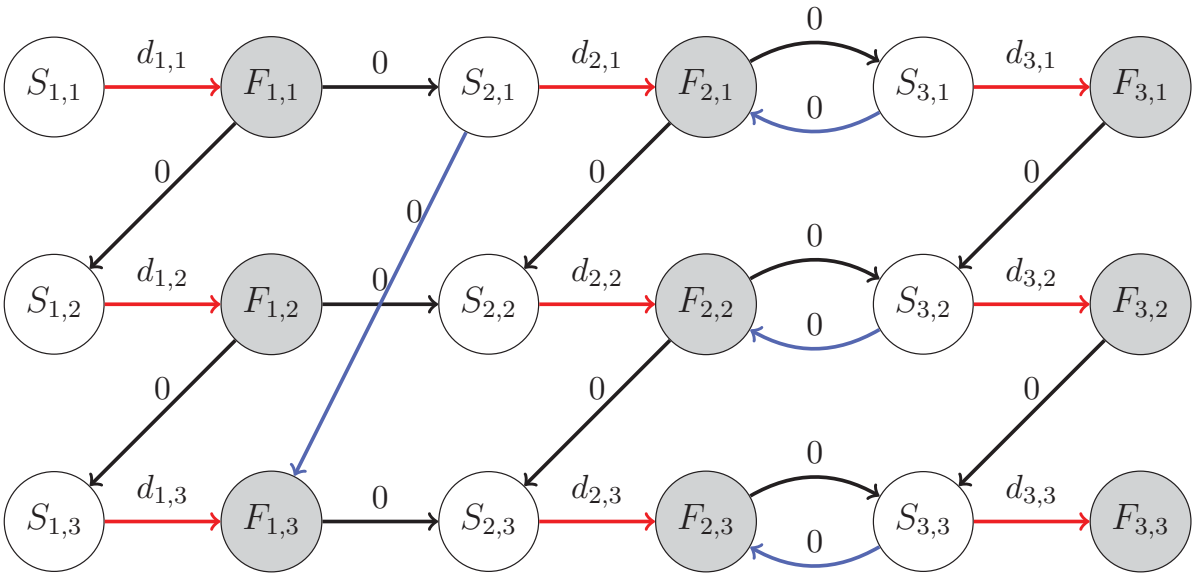


Figure 3.4: Graph representation of a 3-server line with 3 workpieces and fixed buffers $X_1 = 2$, $X_2 = 0$ in the nominal case

To this end we propose an extension to the previously presented graph G. Each node is now reproduced Γ-times, and denoted:

$$S_{g,s,w},\ F_{g,s,w} \quad \forall g \in \{0,\dots,\Gamma\},\ \forall s \in \{1,\dots,S\},\ \forall w \in \{1,\dots,W\} \tag{3.24}$$

Similarly, each arc is repeated for each value of g:

$$\begin{aligned}
&a^1_{g,s,w} = S_{g,s,w} \to F_{g,s,w}, && p = \overline{d_{s,w}}, && \forall g,\ \forall s,\ \forall w\\
&a^2_{g,s,w} = F_{g,s,w} \to S_{g,s+1,w}, && p = 0, && \forall g,\ \forall s < S,\ \forall w\\
&a^3_{g,s,w} = F_{g,s,w} \to S_{g,s,w+1}, && p = 0, && \forall g,\ \forall s,\ \forall w < W\\
&a^4_{g,s,w,b} = S_{g,s,w} \to F_{g,s-1,w+b}, && p = -M\left(1 - Y_{s,b}\right), &&\\
&\qquad \forall g,\ \forall s \geq 2,\ \forall w,\ \forall b \leq W - w &&&&
\end{aligned}$$

To connect these Γ copies of G with one other, a last type of arcs is added:

$$a^5_{g,s,w} = S_{g,s,w} \to F_{g+1,s,w}, \quad p = \overline{d_{s,w}} + \widetilde{d_{s,w}}, \quad \forall g < \Gamma,\ \forall s,\ \forall w$$

Now the g-index of a node symbolizes the amount of processing times that were lengthened so far. Naturally, each node of the form $F_{g,S,W}$ represents a sink in the graph. For simplification purposes, we may get rid of the multiple sinks by adding an artificial vertex t to which each node of the form $F_{g,S,W}$ is connected by a 0-weighted arc. We denote the resulting graph $G(\Gamma)$. An

example is given in Figure 3.5.

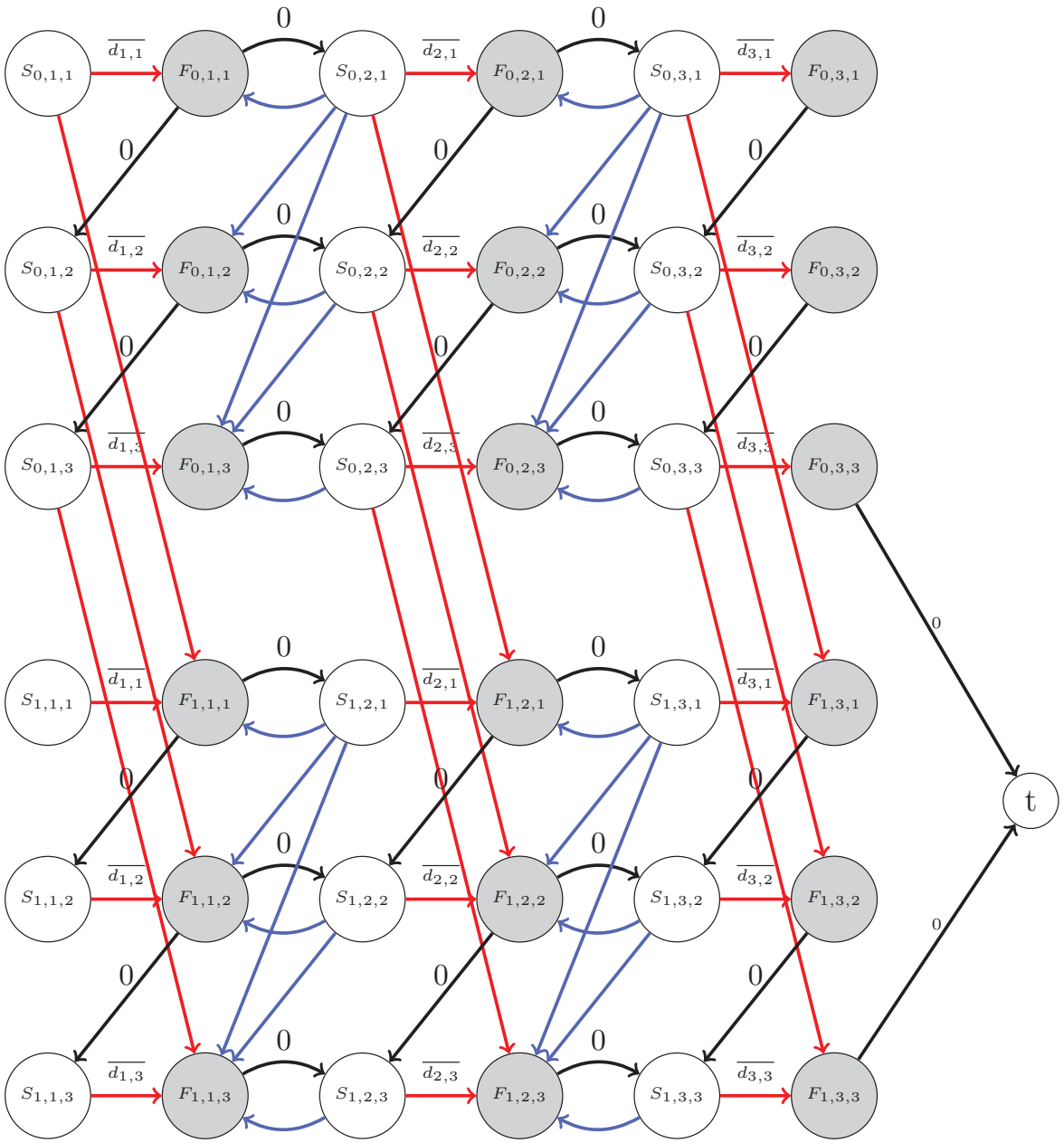


Figure 3.5: Graph representation of a $\Gamma = 1$-robust 3-server line with 3 work-pieces and undetermined buffers

Theorem 5: $G(\Gamma)$ does not contain any cycle of strictly positive weight.

Proof. Let us distinguish sub-graphs $G(k)$, $k \in \{0, \ldots, \Gamma\}$, defined by the set of vertices of $G(\Gamma)$ where $g = k$. We know from Theorem 3 that no such $G(k)$ may contain a cycle of strictly positive weight. The only arcs that link

the $G(k)$ sub-graphs with one another are the arcs of type $a^5_{g,s,w}$, which for a given predecessor in $G(k)$, always have their successor in $G(k+1)$ but never in $G(g)$, where $g < k$. In other words, a topological ordering for $G(\Gamma) \setminus \left\{a^4_{g,s,w,b}\right\}$ is obtained from α as defined in the proof of Theorem 3 when adding the following rule:

$$\alpha\left(S_{g+1,s,w}\right) = \alpha\left(S_{g,s,w}\right) + 2SW \tag{3.25}$$

□

Theorem 6: The minimum time that suffices to reach any particular date, $S_{s,w}$ or $F_{s,w}$, for any $d \in \mathcal{U}(\Gamma)$, equals the total weight of the longest path in $G(\Gamma)$ from $S_{0,1,1}$ to the corresponding vertex. In particular, the longest path from $S_{0,1,1}$ to any $F_{g,S,W},\ g \in \{0,\ldots,\Gamma\}$ gives a vector $d \in \mathcal{U}(\Gamma)$ such that the minimal time $F_{S,W}$ is maximized, as well as the line's shortest time $F_{S,W}$ for d.

Proof. Again, we denote $T_{g,s,w}$ the chosen date, that is, the minimal time that suffices, for any $d \in \mathcal{U}(\Gamma)$, to reach any $S_{g,s,w}$ or $F_{g,s,w}$. Let $t_{g,s,w}$ be the corresponding vertex in G, and $P = \{a_1,\ldots,a_n\}$ a path of length $n \in \mathbb{N}$ from $S_{0,1,1}$ to $t_{g,s,w}$.

Obviously, a node in sub-graph $G(k),\ k \in \{0,\ldots,\Gamma\}$ is reached if and only if at least k entries $d_{s,w}$ of vector $d \in \mathcal{U}(\Gamma) \subset \mathbb{R}^{S\times W}$ deviate from their nominal value, that is, equal $\overline{d_{s,w}} + \widetilde{d_{s,w}}$. Since $k \leq \Gamma$, each path to a node in a sub-graph $G(k),\ k \in \{0,\ldots,\Gamma\}$ renders a vector $d \in \mathcal{U}(\Gamma)$.

We may then, similarly as in the proof of Theorem 4, state that:

$$T_{g,s,w} \geq \sum_{a_i \in P} p(a_i) \qquad \text{by induction over the arc sequence of } P \tag{3.26}$$

$$T_{g,s,w} \leq \sum_{a_i \in P_{\max}} p(a_i) \qquad \text{by minimality of } T_{g,s,w} \tag{3.27}$$

□

This provides us a powerful algorithm to compute robust buffer allocations: for each candidate allocation Y given by RC-Master, the left-hand side in (3.18), i.e.

$$\max_{d\in\mathcal{U}} \{F_{S,W}(Y,d)\}$$

can be computed by finding a critical path in $G(\Gamma)$. This path is bound to exist and is computed in linear time by Theorem 5. The resulting throughput is then compared with TH^*, and combinatorial cuts can be added to the master problem if needed.

3.4 General MIP-Formulation

3.4.1 Longest Path Formulation

We formulate the problem of finding a longest path in $G(\Gamma)$ as a flow problem. One unit of flow is sent from the source $S_{0,1,1}$ to the sink t, the objective is the maximization of the cumulative weight of all traversed arcs. Thus we introduce binary decision variables $x^i_{g,s,w}$, which are set to 1 when $a^i_{g,s,w}$ is traversed, 0 otherwise. The only constraints required are the traditional flow conservation constraints:

$$\text{Maximize} \sum_{\substack{g\in\{0,\dots,\Gamma\},\ s\in\{1,\dots,S\},\\ w\in\{1,\dots,W\},\ i\in\{1,\dots,5\}}} p^i_{g,s,w}\cdot x^i_{g,s,w} \tag{LPP}$$

s.t. $\quad \forall (g,s,w)\neq(0,1,1),$

$$x^1_{g,s,w}+x^5_{g,s,w}+\sum_{b=0}^{W-w} x^4_{g,s,w,b}-x^2_{g,s-1,w}-x^3_{g,s,w-1}=0 \tag{3.28}$$

$$\forall (g,s,w)\neq(\Gamma,S,W),$$

$$x^2_{g,s,w}+x^3_{g,s,w}-x^1_{g,s,w}-x^5_{g-1,s,w}-\sum_{b=0}^{w-1} x^4_{g,s+1,w-b,b}=0 \tag{3.29}$$

$$x^1_{0,1,1}+x^5_{0,1,1}=1 \tag{3.30}$$

$$-x^1_{\Gamma,S,W}-x^5_{\Gamma-1,S,W}=-1 \tag{3.31}$$

$$\forall g,\ \forall s,\ \forall w, \qquad 0\le x^i_{g,s,w}\le 1 \tag{3.32}$$

In addition, any $x^i_{g,s,w}$ is defined to be 0 if the corresponding arc $a^i_{g,s,w}$ does not exist in $G(\Gamma)$. Constraint (3.28) maintains the incoming and outgoing flow balanced for each starting node other than the source, constraint (3.29) plays the same role for the finishing nodes other than the sink, and the next two constraints (3.30) and (3.31) generate or absorb 1 unit of flow at the source or at the sink, respectively. Note that due to total dual integrity, we may replace the condition $x^i_{g,s,w}\in\{0,1\}$ by $x^i_{g,s,w}\in[0,1]$.

3.4.2 Dualization and General Formulation

This previously introduced method presents the advantage of maintaining the separation between a generative method, and an evaluative method that can be

reduced to a pure graph theoretical algorithm, which is provably efficient. The computational effort for one evaluation increases polynomially with the size of the graph, which in turn increases linearly with parameter Γ. Nonetheless, one can easily imagine that for great buffer sizes, even when using efficient cuts, a line may offer a too large amount of combinations to be evaluated. As an alternative, one could utilize the longest path formulation LPP in order to rewrite the robust counterpart as a MILP than can be solved in reasonable time. The first step is to replace the expression $\max_{d\in\mathcal{U}}\{F_{S,W}(Y,d)\}$ in (3.18) by LPP, which results in the following general formulation:

$$\text{Minimize} \sum_{s=1}^{S-1}\sum_{b=0}^{B_s} b\cdot Y_{s,b} \tag{GF-P}$$

$$\text{s.t.} \quad \max \sum_{\substack{g\in\{0,\dots,\Gamma\},\ s\in\{1,\dots,S\},\\ w\in\{1,\dots,W\},\ i\in\{1,\dots,5\}}} p^i_{g,s,w}(Y)\cdot x^i_{g,s,w} \le \frac{W}{\mathrm{TH}^*}$$

This program would be simpler if the maximization problem on the left hand side of the constraint were replaced by a minimization problem. Hence, we propose a dualization of the LPP problem, introducing a variable $S_{g,s,w}$ for each flux conservation constraint of a starting node, as well as a variable $F_{g,s,w}$ for each flux conservation constraint of a finishing node. The general formulation GF-P then becomes:

$$\text{Minimize} \sum_{s=1}^{S-1}\sum_{b=0}^{B_s} b\cdot Y_{s,b} \tag{GF-D}$$

$$\text{s.t.} \quad \sum_{b=0}^{B_s} Y_{s,b} = 1 \qquad \forall s \le S-1 \tag{3.33}$$

$$Y_{s,b} \in \{0,1\} \qquad \forall s \le S-1,\ \forall b \tag{3.34}$$

$$S_{0,1,1} - F_{\Gamma,S,W} \le \frac{W}{\mathrm{TH}^*} \tag{3.35}$$

$$F_{g,s,w} - S_{g,s,w} \le p^1_{g,s,w} \qquad \forall g,\ \forall s,\ \forall w \tag{3.36}$$

$$S_{g,s+1,w} - F_{g,s,w} \le 0 \qquad \forall g,\ \forall s \le S-1,\ \forall w \tag{3.37}$$

$$S_{g,s,w+1} - F_{g,s,w} \le 0 \qquad \forall g,\ \forall s,\ \forall w \le W-1 \tag{3.38}$$

$$F_{g,s-1,w+b} - S_{g,s,w} \le p^4_{g,s,w,b} \qquad \forall g,\ \forall s > 1,\ \forall w < W,\ \forall b \in \{0,\dots,W-w\} \tag{3.39}$$

$$F_{g+1,s,w} - S_{g,s,w} \le p^5_{g,s,w} \qquad \forall g < \Gamma,\ \forall s,\ \forall w \tag{3.40}$$

A different version of the BAP is obtained. Note that Constraints (3.36)-(3.40) now have an additional index g, which makes their total count $(\Gamma+1) \times S \times W$ each, compared to the previous $S \times W$. In exchange for this effort, this MILP provides solutions whose degree of conservatism may be chosen or modified by the user. It remains free of any combinatorial constraint of the form $\forall d \in \mathcal{U}$, which keeps computation tractable. The performance of this formulation is to be tested against that of the decomposed approach presented in Section 3.3. Nonetheless, both algorithms can be further refined. In the next section, initial lower and upper bounds for both approaches are derived, and a simplification method is proposed to greatly reduce the size of the graph $G(\Gamma)$. Finally, the choice of a reasonable Γ is discussed.

3.5 Reducing Computation Times

The introduction of uncertainty increases by a factor Γ the size of the graph in the primal formulation, and the amount of time variables S and F in the dual formulation. In practice, the impact on computation times is strong. Therefore, we seek to accelerate the solving process both by generalizing the lower bounds that are used for the nominal BAP, and by directly reducing the size of the robust formulation.

3.5.1 Derivation of Initial Bounds

In this subsection we aim to accelerate the solving process of the robust BAP by computing lower and upper bounds before the actual solving, by either method, is carried out. Note that the concepts and algorithms in this subsection are presented for the sake of completeness. A more thorough review of bounds and cuts for the BAP, along with their generalization to the case with $W_0 > 0$ is found in Chapter 4.

We start by making the following observations:

Observation 4: Assuming that $W_0 = 0$:

1. Any lower bound for a nominal BAP using processing times $\overline{d_{s,w}}$ is also a lower bound for the robust BAP using nominal processing times $\overline{d_{s,w}}$ in combination with nonnegative deviations $\epsilon_{s,w}$.

2. Any upper bound for a nominal BAP using processing times $d_{s,w}$ is also an upper bound for the robust BAP using nominal processing times $\overline{d_{s,w}}$ and deviations $\epsilon_{s,w}$ such that $\forall s$, $\forall w$, $\epsilon_{s,w} \geq 0$ and $\overline{d_{s,w}} + \epsilon_{s,w} \leq d_{s,w}$

Proof. Since $W_0 = 0$, the throughput of any line $\mathcal{L}_N$ solely depends on the value of $F_{S,W}$, which, in turn, is monotonously increasing with each value $d_{s,w}$. Therefore, a line $\mathcal{L}_R$ with processing times no lower than those of $\mathcal{L}_N$ needs at least as many buffers as the latter. Conversely, a line $\mathcal{L}_R$ with processing times no higher than those of $\mathcal{L}_N$ needs, in turn, at most as many buffers. □

Observation 5: It turns out in practice that the nominal solving of a BAP is negligible compared to the solving of a robust BAP. Therefore, solving nominal BAPs in order to obtain better bounds for a robust BAP is admitted.

Subsequently, we may now focus on the computation of initial lower and upper bounds for nominal BAPs.

In their numerical results, Weiss and Stolletz [88] emphasize that proper bounds, particularly lower bounds, dramatically improve the speed of a such decomposed approach. In fact, they show that good lower bounds for each server are by far more valuable than initial values that are close, or even equal, to the optimum. Global bounds on the objective function, in contrast, are not of great use. This is inherent to the objective: even while assuming that no more, or no less, than N buffer spaces are needed, one still has to evaluate each possible combination of $N-1$ or $N+1$ buffers over all servers. Based on this observation, Levantesi et al. [63] proposed a fast heuristic algorithm that iteratively solves sub-lines of the initial flow line. They compute the minimal amount of buffers required for each two-server sub-line to reach the goal throughput, and set them as local lower bounds for the full server BAP. Weiss and Stolletz [88] generalized their algorithm by applying the same idea recursively over each line size: for size k from 2 to $S-1$, each k-server line is solved and the resulting solutions are used as lower bounds for every further solving. In both cases, the exactness of the algorithm relies on the following assumption:

Claim 1: If a buffer distribution β achieves a given goal throughput TH^* on a S-server line $\mathcal{L}$, then β also achieves TH^* on any k-server sub-line of $\mathcal{L}$, where $k < S$.

Claim 1, despite being very intuitive, does not hold in general (see Theorem 15). Nonetheless, Theorem 7 shows that it is true in our particular case.

Theorem 7: Claim 1 holds for any line when $W_0 = 0$.

Proof. Let us consider a line $\mathcal{L}$ composed of n stations $1, 2, \ldots, n$ respectively. We shall denote $\mathcal{L}[i,j]$, $i \leq j$ any sub-line of $\mathcal{L}$ using stations i through j. For any given buffer distribution β, since $W_0 = 0$, it holds true that:

$$\mathrm{TH}(\mathcal{L}[i,j],\beta) = \frac{W}{F_{j,W}(\mathcal{L}[i,j],\beta) - S_{i,1}(\mathcal{L}[i,j],\beta)} \tag{3.41}$$

Proving the two following statements suffices to prove the claim:

1. $\forall (i,k),\ 1 \leq i \leq k < n,\ \mathrm{TH}(\mathcal{L}[i,k],\beta) \geq \mathrm{TH}(\mathcal{L}[i,k+1],\beta)$
2. $\forall (k,j),\ 1 < k \leq j \leq n,\ \mathrm{TH}(\mathcal{L}[k,j],\beta) \geq \mathrm{TH}(\mathcal{L}[k-1,j],\beta)$

Indeed, for each sub-line $\mathcal{L}[i,j]$, $j - i = k$, $0 \leq k < n$, there always exists a sub-line of greater length that shares the same first or the same last station: $\mathcal{L}[i-1,j]$ or $\mathcal{L}[i,j+1]$. By induction, one can then ensure that for any sub-line $\mathcal{L}[i,j]$, $j - i < n$, $\mathrm{TH}(\mathcal{L}[i,j],\beta) \geq \mathrm{TH}(\mathcal{L},\beta)$ which proves the claim.

Statement (1) can be shown as follows: As both lines $\mathcal{L}[i,k]$ and $\mathcal{L}[i,k+1]$ are identical until server k, the following holds true:

$$F_{k,W}(\mathcal{L}[i,k],\beta) = F_{k,W}(\mathcal{L}[i,k+1],\beta) \leq F_{k+1,W}(\mathcal{L}[i,k+1],\beta) \tag{3.42}$$

Assuming without loss of generality that $S_{i,1}(\mathcal{L}[i,k],\beta) = S_{i,1}(\mathcal{L}[i,k+1],\beta) = 0$ this implies that: $\mathrm{TH}(\mathcal{L}[i,j],\beta) \geq \mathrm{TH}(\mathcal{L}[i,k+1],\beta)$. Statement (2) follows analogously by noticing that:

$$\begin{aligned} & F_{j,W}(\mathcal{L}[k-1,j],\beta) - S_{k,1}(\mathcal{L}[k-1,j],\beta) && (3.43)\\ = {} & F_{j,W}(\mathcal{L}[k,j],\beta) - S_{k,1}(\mathcal{L}[k,j],\beta) && (3.44)\\ \leq {} & F_{j,W}(\mathcal{L}[k-1,j],\beta) - S_{k-1,1}(\mathcal{L}[k-1,j],\beta) && (3.45) \end{aligned}$$

□

In the light of the previous theorems, one may conclude that the subsystems method can be used for exact resolution of the BAP in the special case where $W_0 = 0$.

In addition, the idea behind the latter algorithm may be used to produce more, yet not quite as tight, lower bounds. Instead of solving a subsystem of length k, $k < S$, another possibility is to simulate the full line with its S servers, yet where only k buffers are variables and the other $S - k$ buffers are set as (well-chosen) constants. When setting each non-variable buffer at its upper

bound, i.e. $\min\{W-2, B_s\}$, and solving the resulting subsystem, server-specific lower bounds are also obtained. This new methodology presents three main advantages over the isolated subsystems approach:

1. Its exactness no longer relies on Claim 1. Instead, it relies on the assumption that throughput increases monotonously with the total amount of buffers. Since this assumption is already needed for the combinatorial cuts (3.15), one may consider that this price is acceptable. In the particular case $W_0 = 0$, exactness is guaranteed.

2. Unlike the isolated subsystems, full-line subsystems may be used to compute specific upper bounds, namely by setting all constant buffers to their respective lower bounds. Note that an iterative process alternating the computing of lower and upper bounds, using the best available upper and lower bounds respectively, may provide powerful bounds especially for lines of small S-value.

3. In full-line subsystems, the variable buffers need not be consecutive: any combination of fixed and variable buffers is feasible and therefore, a greater amount of bounds is potentially available for solving the MILP.

More about bounds for the BAP in general and the subsystems methods in particular are found in Chapter 4. It should be noted that in practice, commercial solvers may be very sensitive to the means by which bounds are included in the optimization model, both positively or negatively. As an illustration, solving GF-P as described in Section 3.4.2 directly, using Gurobi 7.0.1, is in some cases up to a factor 10 slower when upper bounds are entered as additional constraints, compared to the solving without any bounds at all. This side effect can be limited or remedied by declaring an additional variable and by constraining the new variable instead of the affected buffer variables. When a bound applies to the objective function, i.e., to the sum of all buffers, it is preferable to add it to the model by cutting off a part of the objective values.

3.5.2 Simplifications of the Graph

The efficiency of the previously given algorithms greatly depends on the size of the used graphs, or, in the case of the general MIP, of the amount of node variables. In this section, we aim to reduce the size of the graphs (or node variables equivalently), both in vertices and in arcs. $G(\Gamma)$ as described so far

contains exactly $2 \cdot \Gamma SW$ nodes. The parameter Γ itself may be chosen, but since it corresponds to the number of durations $d_{s,w}$ that are allowed to deviate, it may vary from 0 to SW, say $\mathcal{O}(S \times W)$. The number of vertices of $G(\Gamma)$ is therefore in the order of:

$$|V| \leq 2 \cdot SW\Gamma \in \mathcal{O}(SW\Gamma) \tag{3.46}$$
$$\in \mathcal{O}(S^2W^2) \tag{3.47}$$

As for the edges, each node $F_{g,s,w}$ has at most 2 outgoing arcs $a^2_{g,s,w}$ and $a^3_{g,s,w}$. A starting node $S_{g,s,w}$ has at most: 1 arc $a^1_{g,s,w}$, 1 arc $a^5_{g,s,w}$ as well as $\mathcal{O}(W)$ arcs $a^4_{g,s,w,b}$ (unknown buffers). This results in:

$$|A| \in \mathcal{O}(SW\Gamma) \text{ vertices } \times \mathcal{O}(W) \text{ arcs per vertex} \tag{3.48}$$
$$\in \mathcal{O}(SW^2\Gamma) = \mathcal{O}(S^2W^3) \tag{3.49}$$

Since a buffer allocation Y is provided by the master problem at the time the graph is being constructed, it is to be noticed that the simplification pointed out by Observation 3 applies here. Ignoring every arc $a^4_{g,s,w,b}$ of nonzero weight leaves a starting node with at most 3 outgoing arcs and thus brings the amount of edges back to:

$$|A| \leq 5 \cdot \Gamma SW \leq 5 \cdot S^2W^2 \tag{3.50}$$

Nonetheless, it is possible to further simplify any graph $G(\Gamma)$, when assuming the following basis:

Claim 2: The following modifications can be applied to the graph without affecting the longest path:

- Any node other than the source or a sink which has no successor or no predecessor may be discarded.
- Any cycle C whose nodes have no other outgoing or no other incoming arc than the arc which is part of C, may be discarded.
- Any arc adjacent to a discarded node may be discarded.

Theorem 8: Following the assumptions of Claim 2, any node in the form:

$$S_{i,j,k},\ i \in \{1, \ldots, \Gamma\},\ j > 0,\ k > 0\text{:} \qquad j + k \leq i + 1 \tag{3.51}$$
$$F_{i,j,k},\ i \in \{1, \ldots, \Gamma\},\ j > 0,\ k > 0\text{:} \qquad j + k \leq i \tag{3.52}$$

may be discarded.

Proof. Let us prove the claim by an induction on i. Any starting node $S_{g,s,w}$ has at most 2 incoming arcs, $a^2_{g,s-1,w}$ if $s > 1$ and $a^3_{g,s,w-1}$ if $w > 1$. Obviously, for all $g > 0$, $S_{g,1,1}$ has no incoming arc and thus is superfluous.

Now let us assume that until a given $i \in \{1, \ldots, \Gamma - 1\}$, for each $g \in \{i, \ldots, \Gamma\}$ every node $S_{g,j,k}$ where $j > 0,\ k > 0,\ j + k \leq i + 1$, as well as every node $F_{g,j,k}$ where $j > 0,\ k > 0,\ j + k \leq i$ has been removed. We consider the incoming arcs of any $F_{g+1,j,k}$ where $j > 0,\ k > 0,\ j + k = i + 1$. These can be of three types:

- $a^1_{g+1,j,k}$, which have been removed along with the corresponding $S_{g+1,j,k}$ by induction hypothesis since: $j + k = i + 1 (\leq i + 1)$
- $a^5_{g,j,k}$, which have been removed too, for the same reason
- $a^4_{g+1,j+1,k-b,b}$: by induction hypothesis, each arc of this type where $b \geq 1$ has been removed since in that case, $j + 1 + k - b \leq i + 1$. However, arc $a^4_{i+1,j+1,k,0}$ is still present as long as its predecessor $S_{g+1,j+1,k}$ remains. As it happens, $S_{g+1,j+1,k}$ has itself:
  - always at least a predecessor, $F_{g+1,j,k}$ by means of arc $a^2_{g+1,j,k}$
  - another predecessor $F_{g+1,j+1,k-1}$ by means of arc $a^3_{g+1,j+1,k-1}$ when $k > 0$

  None of these finishing nodes has yet been removed since $j + k = i + 1 > i$.

We shall show by a second nested induction over k that the latter predecessors are invalid, and therefore all three nodes $F_{g+1,j,k}$, $F_{g+1,j+1,k-1}$ and $S_{g+1,j+1,k}$ are to be discarded for any combination (j, k) such that $j + k = i + 1$. This would suffice to prove the claim.

Let us set $k = 1$ and consider $S_{g+1,j+1,k}$. It has no incoming arc of type 3 since $k - 1 = 0$. Therefore, its only predecessor is $F_{g+1,j,k}$, and it is the only predecessor of $F_{g+1,j,k}$ as well. Therefore, both nodes form a cycle that fulfils the conditions of Claim 2 and may be removed from $G(\Gamma)$.

Let us now assume that until a given $k \in \{1, \ldots, i - 1\}$ all three nodes $F_{g+1,j,k}$, $F_{g+1,j+1,k-1}$ and $S_{g+1,j+1,k}$, where j is such that $j + k = i + 1$, are absent from $G(\Gamma)$. The next starting vertex such that $j + k = i + 1$, $S_{g+1,j,k+1}$, has only one predecessor left, $F_{g+1,j-1,k+1}$, since by induction hypothesis, $F_{g+1,j,k}$ is no longer part of the graph. Therefore, the cycle formed by $S_{g+1,j,k+1}$ and $F_{g+1,j-1,k+1}$ may be cut off the graph, which concludes both inductions and therefore the proof. □

At this stage, one may be tempted to apply the same idea to cut off any nodes that cannot lead to a sink-vertex. Due to the fact that every $F_{g,S,W}$ is a sink-vertex, however, the previous theorem may not be adapted directly. In fact, a symmetric version of Theorem 8 exists if and only if each vertex $F_{g,S,W},\ g \neq \Gamma$ may be discarded, that is, if for a given graph $G(\Gamma)$, there exists a longest path which leads from $S_{0,1,1}$ to $F_{\Gamma,S,W}$. A straightforward idea would be to use the following observation:

Observation 6: Let $P_{(\Gamma-1)}$ be a path from $S_{0,1,1}$ to $F_{\Gamma-1,S,W}$, such that: $\exists(g,s,w)\colon a^1_{g,s,w} \in P_{(\Gamma-1)}$. Then it is possible to build another path P_Γ from $S_{0,1,1}$ to $F_{\Gamma,S,W}$, by replacing arc $a^1_{g,s,w}$ by arc $a^5_{g,s,w}$, and further replace every successor $a^i_{g,s,w}$ by the corresponding $a^i_{g+1,s,w}$. This new path is no shorter than $P_{(\Gamma-1)}$.

This provides a solution: for any given graph $G(\Gamma)$, by Theorem 5, there exists a longest path from $S_{0,1,1}$ to $F_{\Gamma-i,S,W}$ for some $i \leq \Gamma$. If this longest path includes at least i arcs $a^1_{g,s,w}$, then, without loss of generality, using the previous observation, one may discard every node $F_{g,S,W}\colon g \neq \Gamma$, which leads to an equivalent of Theorem 8:

Theorem 9: Following the assumptions of Claim 2, when using $F_{\Gamma,S,W}$ as only sink vertex, any node in the form:

$$F_{\Gamma-i,j,k},\ i \in \{1,\dots,\Gamma\},\ j \in \{1,\dots,S-1\},\ k \in \{1,\dots,W-1\}\colon \quad j+k \geq i \tag{3.53}$$

$$S_{\Gamma-i,j,k},\ i \in \{1,\dots,\Gamma\},\ j \in \{1,\dots,S-1\},\ k \in \{1,\dots,W-1\}\colon \quad j+k \geq i+1 \tag{3.54}$$

$$F_{\Gamma-i,S-j,W-k},\ i \geq 0,\ j \geq 0,\ k \geq 0\colon \qquad j+k < i \tag{3.55}$$

$$S_{\Gamma-i,S-j,W-k},\ i \geq 0,\ j \geq 0,\ k \geq 0\colon \qquad j+k < i+1 \tag{3.56}$$

may be discarded.

Proof. Analogously to the proof of Theorem 8, by induction on increasing i initialized with $i = 1$, $\forall g \leq \Gamma - 1$ $F_{g,S,W}$ can be discarded. Subsequently, in each iteration over i, by means of a second induction on decreasing k, one may remove a cycle $F_{g-1,j-1,k}$-$S_{g-1,j,k}$ that fulfils the conditions of Claim 2. □

Unfortunately, the assumption of the existence of arcs of type 1 in path P, put forward in Observation 6, may not be omitted as the following minimal example shows:

(s,w)	$(1,1)$	$(1,2)$	$(1,3)$	$(1,4)$	$(2,1)$	$(2,2)$	$(2,3)$	$(2,4)$
$\overline{d_{s,w}}$	1	1	1	10	10	1	1	1
$\widetilde{d_{s,w}}$	+0.1	+0.1	+0.1	+1	+1	+0.1	+0.1	+0.1

Table 3.1: nominal processing times and deviations of the line

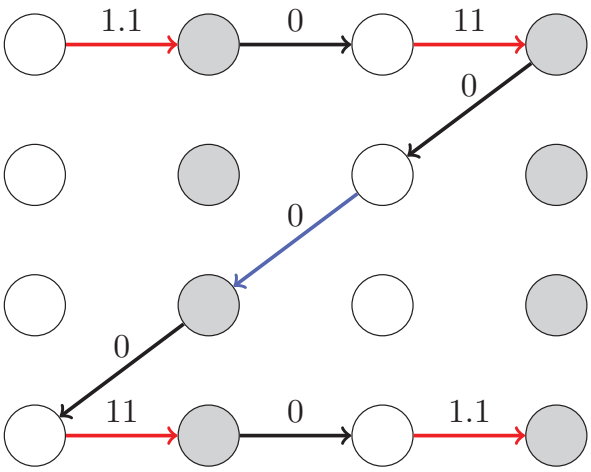


Figure 3.6: Critical path for 2 servers, 4 workpieces from $S_{0,1,1}$ to $F_{4,2,4}$

Example 1: Let us consider a line composed of $S = 2$ servers, $W = 4$ workpieces, robustness $\Gamma = 5$, $X_1 = 1$ buffer slot, and the following processing times:

As it includes both arcs of length 11, one may easily see that the path depicted in Figure 3.6 is critical.

This path has a total length of 25.3. Note that the use of both 11-weighted arcs could only be enabled by means of a $a^4_{g,s,w,1}$ arc ($a^4_{2,2,2,1}$, in the present case). It turns that if we now use vertex $F_{5,2,4}$ as our only sink, the longest path is bound to use exactly 5 nonzero weighted arcs (type 5) in order to reach the last sub-graph. It may therefore no longer include any arc of type $a^4_{g,s,w,1}$. Accordingly, only one of the two 11-weighted arcs can be part of the longest path whose total weight amount to $11 + 4 \cdot 1.1 = 15.4$.

It follows that although some powerful simplifications may be brought to $G(\Gamma)$ for any chosen algorithm (based on a master solution or not) and for any value of Γ, the greatest improvements are reached when the buffers are fixed, and when the value of Γ is "relatively low". This means that the longest paths

should contain sufficiently nonzero weighted arcs (type 1) so that any sub-graph of $G(\Gamma)$ may be reached at will.

3.5.3 Discussion on the Influence of the Gamma-Parameter

One can easily imagine that the choice of the parameter Γ is of central importance for the efficiency of the algorithms presented in this chapter. Besides the fact that it defines the level of robustness desired by the user, it has a strong impact on the complexity of the computations. On the one hand, the amount of sub-graphs of $G(\Gamma)$ grows linearly with Γ. On the other hand, Example 1 indicates that there exists a bound, beneath which large amounts of nodes and arcs may be removed from the graph, and above which no such simplification (presented by Theorem 9) may be carried out.

To start with, a closer look on both simplification Theorems 8 and 9 yields a quick but powerful upper bound on Γ:

Corollary 2: An upper bound on the choice of Γ is given by:

$$U = S + W - 1 \tag{3.57}$$

Proof. Let us, for any graph $G(\Gamma) = G(V, A)$, define a partition $\mathcal{L}$ on the vertices of the graph. Each subset $\ell_{g,i} \in \mathcal{L}$ is defined by:

$$\forall g \in \{0, \dots, \Gamma\}, \forall i \in \{1, \dots, S + W\} : \\ \ell_{g,i} = \{S_{g,s,w} \in V : s + w = i + 1\} \cup \{F_{g,s,w} \in V : s + w = i\}$$

$\mathcal{L}$ is a valid partition of V since each vertex belongs to exactly one $\ell_{g,i}$, whose union equals to $\mathcal{L}$. It can be noticed that the subsets of $\mathcal{L}$ have the following properties:

- When $F_{g,s,w} \in \ell_{g,i}$ then all successors of $F_{g,s,w}$ also belong to $\ell_{g,i}$. This follows by the fact that $F_{g,s,w}$ has at most two successors, $S_{g,s+1,w}$ and $S_{g,s,w+1}$. Since $F_{g,s,w} \in \ell_{g,i}$, we know that $s+w = i$, thus $s+1+w = i+1$ holds too.
- The successor node of an arc of type 1 is always included in $\ell_{g,i+1}$ when $\ell_{g,i}$ is the subset that contains its predecessor node. Indeed, when $S_{g,s,w} \in \ell_{g,i}$ then the successor $F_{g,s,w}$ is such that $s + w = i + 1$ and by definition belongs to $\ell_{g,i+1}$.

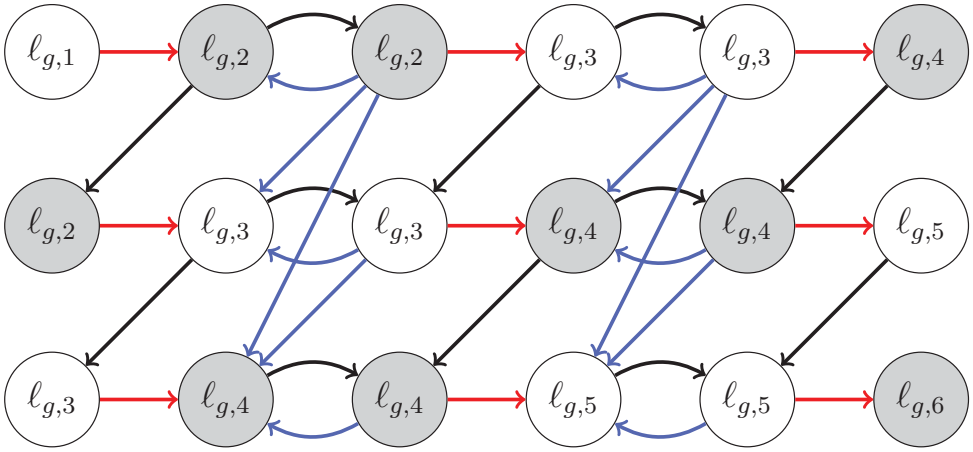


Figure 3.7: Example of a partition $\mathcal{L}$: g^{th} sub-graph of $G(\Gamma)$ where $S = 3$ and $W = 3$

- Similarly, the successor node of an arc of type 5 is always included in $\ell_{g+1,i+1}$ when $\ell_{g,i}$ is the subset that contains its predecessor node. In particular, the source $S_{0,1,1}$ of $G(\Gamma)$ is always in $\ell_{0,1}$ and a sink $F_{g,S,W} \in \ell_{g,S+W}$.
- The successor node of an arc $a^4_{g,s,w,b}$ is always included in $\ell_{g,i+b}$ when its predecessor is in $\ell_{g,i}$. Indeed, we know $s + w = i + 1$, and the successor node is $F_{g,s-1,w+b}$. As $s+w-1+b = i+1-1+b = i+b$, the corresponding subset is $\ell_{g,i+b}$.
- There is no path from a node in $\ell_{g,i}$ to any node in $\ell_{g-k,i}$ or in $\ell_{g,i-k}$, $k > 0 \in \mathbb{N}$. This follows naturally from the previous observations, since all types of arcs have been considered.

It should be observed that any path from the source to a sink must always move from $\ell_{0,1}$ to $\ell_{g,S+W}$ for some $g \leq \Gamma$. The use of each arc of type 1 or 5 increases the i-parameter of $\ell_{g,i}$ by exactly one, and it is impossible to decrease i. Thus, the amount of arcs of type 1 and 5 used by a path from source to sink is bounded from above by $S + W - 1$. □

Theorem 10: A first (trivial) lower lower bound on the amount of nonzero weighted arcs of $G(\Gamma)$, that is, arcs of type 1 and 5, used in a longest path from $S_{0,1,1}$ to $F_{\Gamma,S,W}$ is given by:

$$L = S \tag{3.58}$$

Proof. Let us use the same partition $\mathcal{L}$ of the nodes of $G(\Gamma)$ as in the proof of Corollary 2. Using the same notations, we know that a path from the source

to a sink needs to increment the i-parameter of $\ell_{g,i}$ by exactly $S+W-1$ units. There are exactly 3 types of arcs that increase i: type 1 (by 1 unit), type 5 (by 1 unit) and type 4 (by b units). Nevertheless, type-4 arcs only increase the w index, therefore the use of type-4 index may only increase i by up to $W-1$ units. Subsequently, at least S type-1 and type-5 arcs are needed for the S remaining units of the i-parameter. □

Let us first note that this bound does not require any assumption on the buffer values behind each machine. A possible interpretation of L is as follows: as long as Γ is smaller than this bound, the undesirable effect described in Example 1 does not occur, $S_{\Gamma,S,W}$ may be considered as the unique sink without loss of optimality. This enables us to apply the complete range of simplifications, in particular those described in Theorem 9. When however $\Gamma > L$ then the size of the graph cannot be avoided to be much larger, as there are no nodes without successors that could be cut off. Therefore S is the highest value for Γ such that the computations are still strongly simplified. Nonetheless, $L = S$ remains a poor bound when $W \gg S$: the choice $\Gamma = S \ll W$ could then end up in weak robustness. In practice, when using a decomposition approach, the buffer values behind the servers are known when computing the longest path and are typically very small, since an implicit total enumeration is carried out. In this case, more efficient lower bounds L may be computed.

Theorem 11: A lower bound on the amount of nonzero weighted arcs of $G(\Gamma)$ used in a longest path from $S_{0,1,1}$ to $F_{\Gamma,S,W}$ is given by the following expression:

$$L = U - \left(B \cdot \left\lfloor \frac{W-2}{B+1} \right\rfloor + (W-2) \mod (B+1) - 1 \right) \tag{3.59}$$

where $B = \max_{s \in \{1,\dots,S-1\}} X_s$ and X_s denotes the amount of buffer slots behind server s.

Proof. We proceed similarly as in Corollary 2 and Theorem 10: we know that a path from the source to a sink needs to increment the i-parameter of $\ell_{g,i}$ by exactly $S+W-1$ units. There are exactly 3 types of arcs that increase i: type 1 (by 1 unit), type 5 (by 1 unit) and type 4 (by b units). Therefore, minimizing the amount of positively weighted arcs in a path is equivalent to maximizing the amount of arcs of type 4 with $b > 0$.

It should however be stressed that a longest path imposes additional constraints on the use of type-4 arcs:

- An arc $a^4_{g,s+1,w,b}$, $b \neq 0$ may not be used directly after an arc $a^2_{g,s,w}$. Indeed, this would create a path of weight 0 from $S_{g,s,w}$ to $S_{g,s,w+b}$. Yet, repeating the sequence $\{a^3_{g,s,w+k},\ a^1_{g,s,w+k}\}$ b-times for $k = 0, \ldots, b$ also yields a path from $S_{g,s,w}$ to $S_{g,s,w+b}$, but with positive weight, contradicting the maximal property of the path.
- Similarly, an arc $a^2_{g,s-1,w+b}$ may not be used directly after an arc $a^4_{g,s,w,b}$, $b \neq 0$. Such a sequence would be shorter than an equivalent repeated $\{a^1_{g,s,w+k},\ a^3_{g,s,w+k}\}$ sequence.

It follows that a type-4 arc must be both preceded and followed by a type-3 arc. Each of these arcs points towards a node of strictly higher workpiece index w. Therefore, the increase of i is limited by the "cost" in w. The w index can be increased by $W-1$ units at most, and each sequence of a type-4 arc and a type-3 arc increases i by b and w by $b+1$. Therefore, repeating such a sequence for the highest possible b value yields an increase of i of:

$$\overline{b} \cdot \left\lfloor \frac{W-2}{\overline{b}+1} \right\rfloor \quad \text{where} \quad \overline{b} = \max\left\{b\colon p^4_{g,s,w,b} = 0\right\} \tag{3.60}$$

The maximal b such that $p^4_{g,s,w,b} = 0$ is exactly equal to the amount of buffer slots behind station s, X_s. Since we aim to maximize b, we should use the highest buffer value, i.e. $\max_{s\in\{1,\ldots,S-1\}} X_s = B$. Expression (3.60) can therefore be rewritten as:

$$B \cdot \left\lfloor \frac{W-2}{B+1} \right\rfloor \tag{3.61}$$

This number represents the maximal increase of the w-index which type-4 arcs where $b = B$ may account for. In the maximal case, however, the remaining $W-2-(B+1)\cdot\left\lfloor\frac{W-2}{B+1}\right\rfloor$ may also be used to increase i by means of one last sequence $\{a^4_{g,s,w,b},\ a^3_{g,s-1,w+b}\}$ where $b < B$. This last sequence increases i by exactly $(W-2) \mod (B+1) - 1$. Consequently:

$$L = S + W - 1 - \left(B \cdot \left\lfloor \frac{W-2}{B+1} \right\rfloor + (W-2) \mod (B+1) - 1\right) \tag{3.62}$$

□

This new lower bound L is an attractive candidate for the choice of Γ, since it still allows accelerated solving for a minimal loss of robustness. The quality of this choice solely depends on the size of the largest buffer: when the largest buffer equals 1, one obtains $\Gamma \simeq S + \frac{1}{2}W$ for a sufficiently large amount of workpieces. On the contrary, when the largest buffer amounts to 10, one would end up with $\Gamma \simeq S + \frac{1}{11}W$.

3.6 Numerical Study

3.6.1 Methodology

This section aims to compare both presented solution approaches and to give a first idea of the computational price that is paid for the robustness of solutions. We formulate the hypothesis that this price greatly depends both on the level of robustness, that is, the amplitude of the deviations $\epsilon_{s,w}$ and parameter Γ, as well as on the size of the instance, i.e. constants S and W. Therefore the following methodology is applied: different values of S and W to be tested are defined. For each combination (S, W), 16 samples of randomly-generated nominal processing times are generated. The maximal deviations are equal to a percentage of the corresponding nominal values, which is given by a parameter E. Each sample is then solved by both algorithms for several values of E and Γ, the latter increasing stepwise in a range from $\Gamma = 0$ (nominal case) until $\Gamma = S + W + k$ where $k \geq 1$ is a given integer (fully robust case). The values that are presented next are the mean values over all 16 samples. In order to avoid too long computation times, a time limit is set to 14400 seconds (4 hours) for each computation.

The constants that are common to all instances are chosen to be as close as possible to the numerical study of Matta [66]: in each line, all servers have the same mean processing rate, namely 7.0 workpieces per time unit, except for one bottleneck server, arbitrarily the second, that has a rate of 6.0. The maximal amount of buffer slots B_s is set to 20 for each server s. As justified in Section 3.2.4, no warm-up phase is carried out ($W_0 = 0$). It should be noted that since robustness is achieved through the concept proposed by Bertsimas and Sim [14], and not through solving samples of very large size, it is not necessary that W should take as high values as what can sometimes be found in literature.

All computations were carried out on an Intel(R) Xeon(R) CPU E5-2690 with 2.9 GHz and 192 GB RAM. Implementation was done in Python 2.7, and the solving of the MILPs was performed by Gurobi 7.0 with default settings. Gurobi was restricted to run on a single core. Similarly to the numerical study of Weiss and Stolletz [88], samples were generated using Descriptive Sampling as described by Saliby [77]. This technique consists in computing values that represent a given random distribution using a deterministic input. Next, the obtained values are shuffled to simulate randomness. In the present case, an exponential distribution is simulated by means of the inverse distribution function. Saliby [78] argues that descriptive sampling generally leads to a more

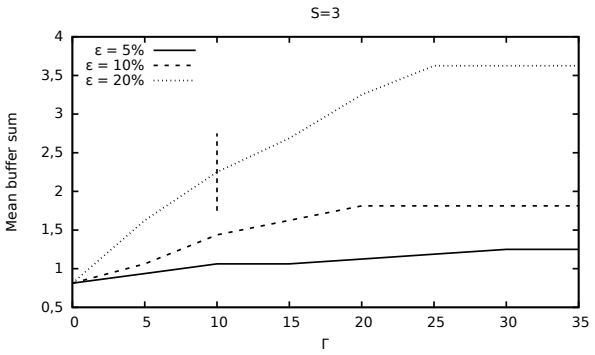


Figure 3.8: $W = 30$, $\mathrm{TH}^* = 4.0$

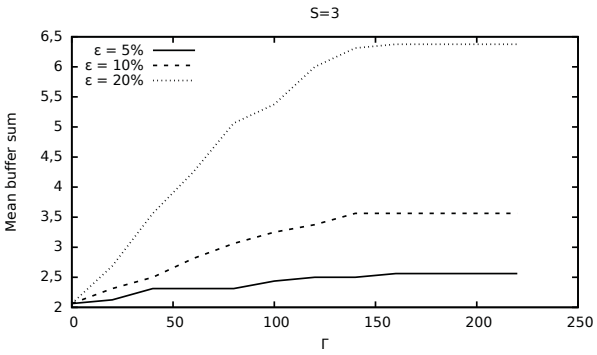


Figure 3.9: $W = 200$, $\mathrm{TH}^* = 4.5$

precise description of the target distribution compared to random sampling, as well as to a reduction of the variability of the results. Stolletz and Weiss [84] underline the benefits of descriptive sampling for the BAP in particular.

3.6.2 Results

Small lines (3 stations)

We first focus on the evolution of the required buffers in small lines, which we define as 3-server lines. Figures 3.8 and 3.9 show the buffer sums over the complete range of Γ, for three particular values of E. Two settings are put to test, a short line with a small amount (30) of workpieces, and a short line with a great amount (200) of workpieces. Although they are not displayed here, the same computations have been carried out for respectively 50 and 100 workpieces.

Note that it may happen, for some of the samples, that the problem becomes infeasible for certain combinations of Γ and E, or that the time limit is exceeded. Hence, the mean buffers or processing times are then computed over subsets of the 16 samples, resulting in unexpected mean values. When this occurs, for each curve, we indicate by a vertical axis the value of Γ until which all 16 values could be used to compute a mean, and therefore until which the curve can be correctly interpreted. Unsurprisingly, the simulations confirm the intuition that the minimal sum of all buffers is monotonously increasing with each parameter, Γ and E. This relation does not seem to be influenced by W, as the shapes of the curves look similar for $W = 30$ and $W = 200$. It is worth mentioning, however, that Γ seems to influence the required sum of buffers only

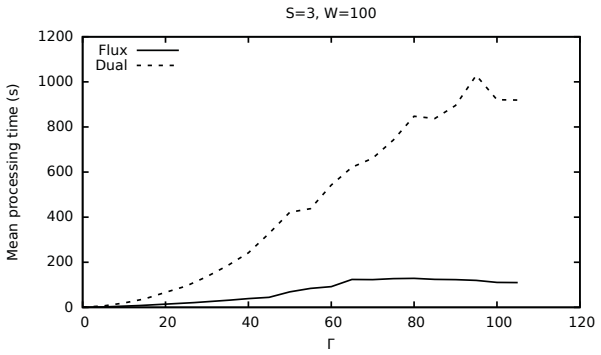


Figure 3.10: $E = 5\%$, $\mathrm{TH}^* = 4.5$

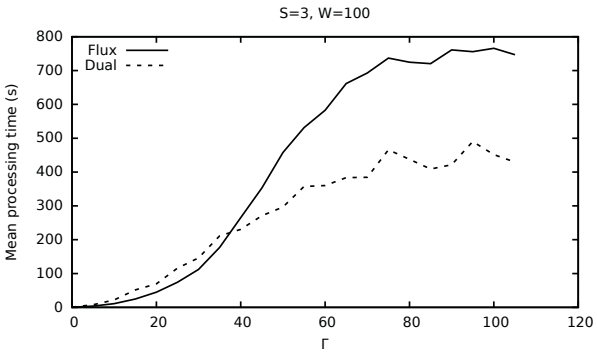


Figure 3.11: $E = 20\%$, $\mathrm{TH}^* = 4.5$

up to a certain threshold. In our experiments, this threshold lies at $\Gamma = 25$ for 30 workpieces and $\Gamma = 140$ for 200 workpieces, representing $25/32 = 78\%$ and $140/202 = 69\%$ of the total range of Γ. This phenomenon was also observed on a line with 50 and 100 workpieces, where the thresholds lie at 67% and 64% respectively. Consequently, on these lines, setting Γ to a value close to 70% is roughly equivalent to the fully robust case.

Let us now focus on computation times on small lines. Figures 3.10 and 3.11 show the computation times over the complete range of Γ, for a particular value of E, resp. 5% and 20%. First, it can be observed that the shape of the curves is similar to that of the total buffers; computing times strongly increase with Γ until a certain value, after which they seem to be capped. However, the influence of E seems to be different according to the resolution algorithm. While an increase of E quickly reduces the efficiency of the iterative solving of flux problems, it seems to slightly facilitate the direct dual resolution. Therefore, as confirmed by the results obtained with 30 and 50 workpieces, faster computation times are obtained with the flux method when E equals 5% or 10%, yet the dual method becomes the better option when $E = 20\%$. An explanation for these trends is that on the one hand, the efficiency of the flux method directly depends on the total amount of buffers. For very small amounts, only a few different buffer distributions are to be tested, whereas the amount of flux problems to be solved quickly explodes with larger buffers. Since the required buffers directly increase with E, it is logical that this impacts computation time. On the other hand, the influence of E on the dual algorithm is less clear, as it only impacts the values of the $p^5_{g,s,w}$-weights in GF-D. Hence, no major fluctuations in complexity are expected. In return, the amount of workpieces seems to counteract this phenomenon; the results obtained with $W = 200$ show that the value of E after which the dual algorithm is faster is

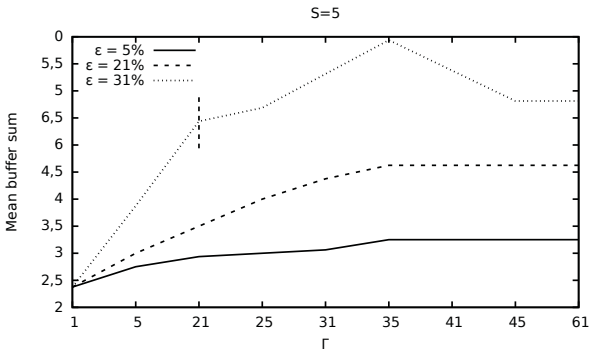


Figure 3.12: $W = 30$, $\mathrm{TH}^* = 3.5$

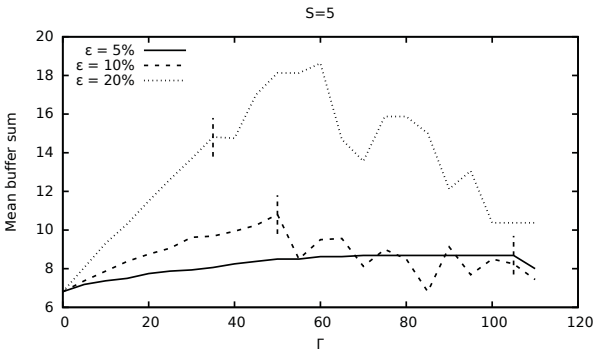


Figure 3.13: $W = 100$, $\mathrm{TH}^* = 4.5$

increased. These parameters should therefore be carefully weighted against each other when choosing the faster method.

Long lines (5 stations and more)

Results have been obtained for settings with both 5 and 7 stations in combination with 30, 50 and 100 workpieces. However, those results where $S = 7$ are unfortunately hard to interpret due to the high amount of incomplete samples, caused by time limit exceeding. Therefore, in this section, we shall mainly present results from 5-station lines. The trends we describe, nonetheless, seem to be confirmed by the partial information that can be gained from the 7-station lines. As previously, we first analyse the evaluation of required buffers. Figures 3.12 and 3.13 show the total buffers on a 5-server line with respectively 30 and 100 workpieces: Similar trends as for small lines are observed: buffers grow monotonously with Γ and with E. It should be noted that the amount of stations S itself appears to have a strong increasing impact on required buffers. While, at constant goal throughput $\mathrm{TH}^* = 4.5$, gamma $\Gamma = 35$, deviation $E = 20\%$ and workpieces $W = 100$, our 3-server line required 5.438 buffers in average, this value amounts to 14.813 buffers for our 5-server line. As for computational effort on longer lines, the computing times for a 5-server line with 100 workpieces are depicted in Figures 3.14 and 3.15 for E values of respectively 5% and 10%: It can be read from these curves that with longer lines, the flux algorithm becomes less competitive against the dual method. Figure 3.14 shows that even for the smallest values of E that was tested, i.e. 5%, both algorithms seemed roughly equally efficient. With increasing values of E, as observed previously, the dual method becomes increasingly efficient

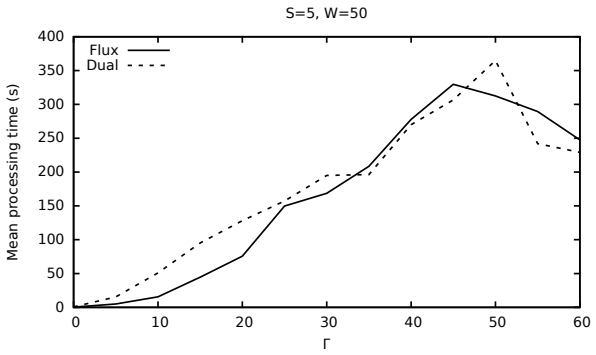


Figure 3.14: $E = 5\%$, $\mathrm{TH}^* = 4.0$

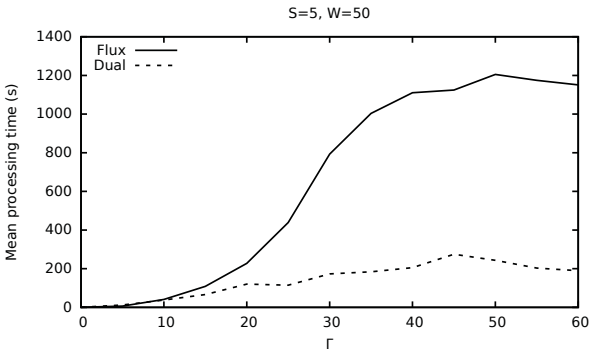


Figure 3.15: $E = 10\%$, $\mathrm{TH}^* = 4.0$

relatively to the flux method. This effect should be taken into account when choosing a proper solving method.

3.7 Conclusions

We extended an already existing, efficient model [88] for the exact solving of large instances of the BAP. We take into account uncertainties over the data set, thereby achieving robustness [11, 12]. We propose two flexible algorithms allowing the user to set the required level of conservatism of the solution. The first algorithm consists in iteratively solving a flux problem in a graph, and is relatively efficient for short lines or small deviation values $\epsilon_{s,w}$. The second algorithm is the direct solving of an MILP, and yields the faster results for long lines or greater deviations. Since the uncertainty is part of the model, it is not needed to try out very large samples in order to achieve robustness.

However, computation times can quickly become very large even for moderate values of servers and workpieces, when the required level of robustness is high. This drawback is significantly mitigated by the reduction of the size of the graphs in which the flow problems are solved, as well as by the set of new initial bounds that were derived in Section 3.5.1. Nonetheless, the difference with the computation times obtained by [88] remains relatively large. This can be explained on the one hand by the robust nature of this new problem, and therefore by its very large size. On the other hand, the efficiency for the nominal BAP was greatly improved by concepts, such as the use of callbacks, or the initial bounds of Levantesi et al. [63]. While callbacks become counter-productive in the robust case, since the sub-problem is much harder to solve

than the master problem, it seems the lower bounds by Levantesi et al. [63] cannot be generalized to the robust problem (see Section 3.5.1).

Hence, further research should be directed towards reducing the computational effort. We believe that stronger initial bounds may be derived, or extended from the nominal problem. The case where a warm-up phase is employed deserves particular attention, since as pointed out in Section 3.2.4, it rules out most of the improvements that were brought to the computation speed.

Chapter 4

Speeding up Buffer Allocation

The Buffer Allocation Problem (BAP) is notoriously hard. Due to tractability issues with analytical methods, sample-based approaches are commonly utilized. This motivates the need for fast algorithms in the deterministic case. In addition, as shown in Chapter 3, methods for the deterministic case typically form the basis of approaches from robust optimization where uncertainty is modelled by uncertainty sets. Recent advances in literature improved solving speed for single samples of processing times. However, these methods cannot be applied in case a warm-up phase is performed during the throughput evaluation, which is very common for the BAP. In this chapter we generalize the existing fast exact procedures to the case with warm-up. We also present an alternative algorithm which uses an approximate version of throughput in order to derive bounds. A brief numerical study compares the efficiency of both methods, and illustrates the benefits of our new algorithm.

4.1 Introduction

The BAP is $\mathcal{NP}$-hard, and it has been extensively studied in literature. Classifications of the research work can be found in Park [74], Demir et al. [35] and Weiss et al. [87]. The two major streams of research that emerge from these reviews are allocation optimization and performance evaluation. Typically, the BAP is solved by combining a solution-generating strategy with an evaluative method. Optimal solutions are only guaranteed if both stages are carried out in an exact way.

Optimization methods are concerned with the computation of new buffer allocations to be evaluated. Since most exact methods, such as complete enumeration, are only applicable to small instances, numerous approximations

are proposed in literature. Typical methods include gradient searches [63], time buffers [5], or meta-heuristics such as tabu search [31] and simulated annealing [83]. Recently, exact methods are experiencing renewed interest; Matta [66] presents an exact MIP formulation that is then extended by Stolletz and Weiss [84]. Weiss and Stolletz [88] speed up the optimization by applying combinatorial cuts from Codato and Fischetti [28] and by computing lower bounds.

Evaluation methods focus on measuring throughput rates or average queues. One of the most common approximations is the decomposition method developed by Gershwin [44]. The main idea is to break down the flow line into smaller sub-lines, that can be easily evaluated separately. It was applied and extended by Dallery et al. [32], Gershwin and Schor [45], Demir et al. [36] and Nahas [69]. Another popular method, aiming for realistic or more detailed modelling, is simulation. Its main advantage, compared to analytical methods, is that it can be applied to large instances. In a simulation approach, the problem is solved for individual samples whose behaviour is assumed to be close to that of real systems. The stochastic nature of the problem is taken into account by considering different samples, or a sample of larger size. The individual problems, on the other hand, are deterministic. How well their solutions may be generalized depends on our ability to quickly solve large deterministic samples. For applications of simulation for the BAP, we refer to Gürkan [49], Helber et al. [55] and Weiss and Stolletz [88].

In this chapter, we solve the BAP to optimality for known deterministic processing times. Such instances arise for example if samples are generated. We therefore combine an exact optimization approach with a sample-exact evaluation procedure. Our main contribution is to speed up the process for the most common case, where a warm-up phase is carried out when measuring throughputs. This chapter is organized as follows: Section 4.2 recalls the notations and the model used for the no warm-up case. Section 4.3 shows why that model is not applicable to the warm-up case, and converts it into a provably exact model. In Section 4.4, a new solving approach is derived using properties of the techniques in Section 4.3. Finally, a short numerical study on the performance of these algorithms is provided by Section 4.5.

4.2 Exact Buffer Allocation without Warm-up

4.2.1 Notations and Modelling

Our starting point is the same problem as described previously in Section 3.2. We are given a flow line $\mathcal{L}$ of S servers $1, \ldots, S$ where W workpieces $1, \ldots, W$ are processed sequentially (see Figure 3.1). This setting is shown on Figure 3.1. We are concerned with the primal formulation of the BAP; the total amount of buffer slots is minimized such that a goal throughput for the sample of processing times is reached. Hence, it is necessary to define the notion of average throughput properly. For a fixed schedule $\mathscr{S}$, let $S_{s,w}$ and $F_{s,w}$ denote the starting and finishing times of a workpiece w on a server s respectively. In the particular case where no warm-up phase is performed, the definition of throughput is:

$$\mathrm{TH} = \frac{W}{F_{S,W}} \tag{4.1}$$

Although starting times, finishing times and throughputs all depend on the setting of the line $\mathcal{L}$ and on the schedule $\mathscr{S}$, for the sake of simplicity, the $\mathscr{S}$ and $\mathcal{L}$ indices are omitted whenever they are not necessary.

Our approach closely resembles Benders' Decomposition; the problem is broken down into a generative method and a performance evaluating problem. The main idea is illustrated on Figure 3.2 in Subsection 3.2.3. An integer master problem generates candidate solutions, while the continuous sub-problem evaluates each solution and produces new constraints for the master problem in case the solution is not satisfactory. Iteratively solving these formulations then provides the optimum. Since one does not know how many iterations are needed, the efficiency of such an approach relies on two factors: both problems should be quick to solve, and the constraints produced by the sub-problem should cut off as many suboptimal solutions as possible.

The MILP formulations of the master and sub-problem are found in [88], whose notations we use throughout this work. They are similar to the formulations presented in Chapter 3, except robustness is not considered here. The values of $d \in \mathbb{R}^{S \times W}$ are not given by an uncertainty set, they are known in advance.

The generative problem is written as follows:

$$\text{Minimize} \sum_{s=1}^{S-1} X_s \qquad \text{(Master Problem)}$$

$$\text{s.t.} \quad \sum_{b=0}^{B_s} b \cdot Y_{s,b} = X_s \qquad \forall s \leq S-1 \qquad (4.2)$$

$$\sum_{b=0}^{B_s} Y_{s,b} = 1 \qquad \forall s \leq S-1 \qquad (4.3)$$

$$Y_{s,b} \in \{0,1\} \qquad \forall s \leq S-1,\ \forall b \qquad (4.4)$$

Recall that the binary decision variables $Y_{s,b}$ are set to 1 in case the buffer size behind server s is equal to b, and 0 otherwise. B_s is the maximum amount of buffer slots allowed behind server s. The sub-problem asks for a feasible schedule specified by the starting and finishing times and obeys the given buffer allocation Y. It is given by the following MILP:

$$\text{Minimize } 0 \qquad \text{(Sub-Problem)}$$

$$\text{s.t.} \quad F_{S,W} \leq \frac{W}{\mathrm{TH}^*} \qquad (4.5)$$

$$S_{s,w} - F_{s,w} \leq -d_{s,w} \qquad \forall s,\ \forall w \qquad (4.6)$$

$$F_{s,w} - S_{s+1,w} \leq 0 \qquad \forall s \leq S-1,\ \forall w \qquad (4.7)$$

$$F_{s,w} - S_{s,w+1} \leq 0 \qquad \forall s,\ \forall w \leq W-1 \qquad (4.8)$$

$$S_{s+1,w} - F_{s,w+b} \leq M \cdot (1 - Y_{s,b}) \qquad \forall s \leq S-1,\ \forall b,\ \forall w \leq W-b \qquad (4.9)$$

Note that constraint (4.9), again, is a "Big M"-constraint, where M is a value that is assumed to be large enough for the inequality to always hold true in case $Y_{s,b}$ equals 1. There are two outcomes of the sub-problem; if it is feasible, then the buffer allocation Y corresponds to the sought optimum, as it is the smallest feasible solution. If the sub-problem is infeasible, this means an additional constraint needs to be added to the master problem.

Solving these formulations as such is no viable way to obtain solutions for larger instances, even in the nominal case. Clever cuts and bounds are required so as to guarantee tractability of the process.

4.2.2 Combinatorial Cuts

Different types of cuts, after the solving of the sub-problem, were suggested in literature, e.g. Benders Cuts [88]. To the best of our knowledge, the fastest known cuts are the combinatorial cuts proposed by Codato and Fischetti [28] and mentioned in Chapter 3:

$$\sum_{s=1}^{S-1} \sum_{b=b_s+1}^{B_s} Y_{s,b} \geq 1 \tag{4.10}$$

They ensure that in case the realized throughput of a given buffer allocation is insufficient, then at least one of the buffers must be increased by at least one unit. These cuts were numerically shown by [88] to strongly speed up the solution process. Of course they do rely on the assumption that the realized throughput is monotonously increasing with the addition of buffer slots. This intuitive assumption can be proved to hold when no warm-up is performed.

Definition 7: A *buffer distribution* β is a vector that represents the amounts of buffer slots behind each server, and is defined as follows:

$$\begin{array}{rccc} \beta & : & \mathcal{S} \setminus \{S\} & \rightarrow & \mathbb{N} \\ & & s & \mapsto & \beta(s) \end{array}$$

Definition 8: A *schedule* (S, F), $S \in \mathbb{R}^{S \times W}$, $F \in \mathbb{R}^{S \times W}$ is defined by two sets of starting and finishing dates, for each workpiece on each server. It is said to be *feasible* if it satisfies constraints 4.6 through 4.9 in Sub-Problem. A *fastest no-wait schedule* is a feasible schedule such that each date is set as early as possible. It can be computed in polynomial time by means of an algorithm given by Weiss and Stolletz [88] (see Algorithm 1). Since a fastest no-wait schedule is uniquely defined given a set of servers, a set of workpieces and a buffer distribution, it shall be denoted $\mathscr{S}(\mathcal{S}, \mathcal{W}, \beta)$.

In the remainder of this section, β and β'_j denote two buffer distributions, such that β'_j is obtained from β by adding 1 buffer slot to server j. Formally, the following holds true:

$$\begin{aligned} \forall i \in \mathcal{S},\ i \neq j,\ \beta'_j(i) &= \beta(i) \\ \beta'_j(j) &= \beta(j) + 1 \end{aligned}$$

Lemma 1: Recall that $S_{s,w}$ and $F_{s,w}$ denote respectively the starting and finishing times of workpiece w on station s. Let $\mathscr{S}(\mathcal{S}, \mathcal{W}, \beta)$ and $\mathscr{S}'(\mathcal{S}, \mathcal{W}, \beta'_j)$

Algorithm 1 Computing the fastest no-wait schedule given allocation β

1: $S_{1,1} \leftarrow 0$
2: **for** $s \leftarrow 1$ to $S-1$ **do**
3: $\quad F_{s,1} \leftarrow S_{s,1} + d_{s,1}$
4: $\quad S_{s+1,1} \leftarrow F_{s,1}$
5: $F_{S,1} \leftarrow S_{S,1} + d_{S,1}$
6: **for** $w \leftarrow 2$ to W **do**
7: $\quad$ **for** $s \leftarrow 1$ to $S-1$ **do**
8: $\quad\quad$ **if** $s = 1$ **then**
9: $\quad\quad\quad S_{s,w} \leftarrow F_{s,w-1}$
10: $\quad\quad$ **else**
11: $\quad\quad\quad S_{s,w} \leftarrow \max\{F_{s,w-1}, F_{s-1,w}\}$
12: $\quad\quad$ **if** $\beta(s) = 0$ **then**
13: $\quad\quad\quad F_{s,w} \leftarrow \max\{S_{s,w} + d_{s,w}, F_{s+1,w-1}\}$
14: $\quad\quad$ **else if** $\beta(s) \geq w$ **then**
15: $\quad\quad\quad F_{s,w} \leftarrow S_{s,w} + d_{s,w}$
16: $\quad\quad$ **else**
17: $\quad\quad\quad F_{s,w} \leftarrow \max\{S_{s,w} + d_{s,w}, F_{s+1,w-\beta(s)}\}$
18: $\quad S_{S,w} \leftarrow \max\{F_{S,w-1}, F_{S-1,w}\}$
19: $\quad F_{S,w} \leftarrow S_{S,w} + d_{S,w}$

denote the two fastest no-wait schedules defined by β and β'_j respectively. If for given $s \in \mathcal{S}$ and $w \in \mathcal{W}$, the following holds true:

$$F^{\mathscr{S}'}_{1,w} \leq F^{\mathscr{S}}_{1,w} \tag{4.11}$$

then it also holds true that:

$$S^{\mathscr{S}'}_{1,w+1} \leq S^{\mathscr{S}}_{1,w+1} \tag{4.12}$$

The implication of this lemma is depicted in Figure 4.1. The symbols S and F in the table mean that the corresponding starting or finishing date in schedule $\mathscr{S}'$ no greater than it is in schedule $\mathscr{S}$. The dates in blue symbolize the hypotheses of the lemma, the dates in red symbolize the conclusion.

Proof. The proof is immediate. Since all workpieces are assumed to be waiting in front of station 1, it holds true that:

$$S^{\mathscr{S}'}_{1,w+1} = F^{\mathscr{S}'}_{1,w} \leq F^{\mathscr{S}}_{1,w} = S^{\mathscr{S}}_{1,w+1} \qquad \square$$

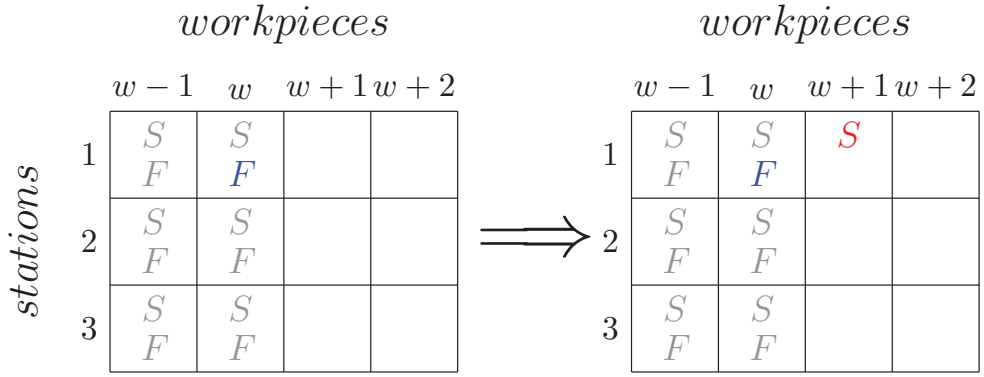


Figure 4.1: Illustration of the implication in Lemma 1

Lemma 2: Let $\mathscr{S}(\mathcal{S}, \mathcal{W}, \beta)$ and $\mathscr{S}'(\mathcal{S}, \mathcal{W}, \beta'_j)$ denote the two fastest no-wait schedules defined by β and β'_j respectively. If for given $s \in \mathcal{S}$ and $w \in \mathcal{W}$, the following holds true:

$$\forall k \leq w,\ S^{\mathscr{S}'}_{s+1,k} \leq S^{\mathscr{S}}_{s+1,k} \tag{4.13}$$

$$F^{\mathscr{S}'}_{s+1,w} \leq F^{\mathscr{S}}_{s+1,w} \tag{4.14}$$

$$S^{\mathscr{S}'}_{s,w+1} \leq S^{\mathscr{S}}_{s,w+1} \tag{4.15}$$

then it also holds true that:

$$F^{\mathscr{S}'}_{s,w+1} \leq F^{\mathscr{S}}_{s,w+1} \tag{4.16}$$

$$S^{\mathscr{S}'}_{s+1,w+1} \leq S^{\mathscr{S}}_{s+1,w+1} \tag{4.17}$$

The implication of this lemma is depicted in Figure 4.2. The symbols S and F in the table mean that the corresponding starting or finishing date in schedule $\mathscr{S}'$ is no greater than it is in schedule $\mathscr{S}$. The dates in blue symbolize the hypotheses of the lemma, the dates in red symbolize the conclusion.

Proof. In order to perform the proof, two cases are distinguished:

- $F^{\mathscr{S}'}_{s+1,w} \leq S^{\mathscr{S}'}_{s,w+1} + d_{s,w+1}$: this means that server $s+1$ is idle when s finishes processing workpiece $w+1$. Hence, using (4.15):

  $$S^{\mathscr{S}'}_{s+1,w+1} = F^{\mathscr{S}'}_{s,w+1} = S^{\mathscr{S}'}_{s,w+1} + d_{s,w+1} \leq S^{\mathscr{S}}_{s,w+1} + d_{s,w+1} \leq F^{\mathscr{S}}_{s,w+1} \leq S^{\mathscr{S}}_{s+1,w+1}$$

  Note that this proves both (4.16) and (4.17)

- $F^{\mathscr{S}'}_{s+1,w} > S^{\mathscr{S}'}_{s,w+1} + d_{s,w+1}$: in this case, $s+1$ is not yet idle after the

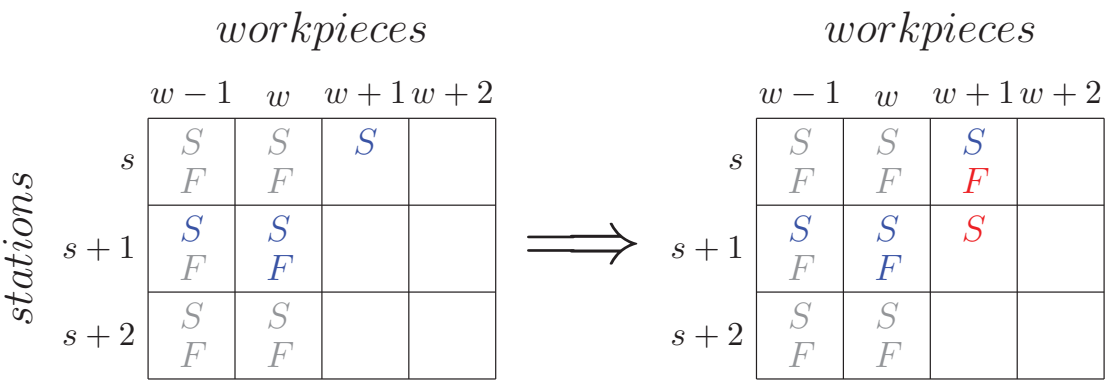


Figure 4.2: Illustration of the implication in Lemma 2

processing of $w+1$ on s. Then, (4.17) can be shown using (4.14):

$$S^{\mathscr{S}'}_{s+1,w+1} = F^{\mathscr{S}'}_{s+1,w} \leq F^{\mathscr{S}}_{s+1,w} \leq S^{\mathscr{S}}_{s+1,w+1}$$

Since the value of $F^{\mathscr{S}'}_{s,w+1}$ now depends on the the buffer size $\beta(s)$, another three cases are distinguished in order to show (4.16):

– $\beta(s) = 0$: then there is no buffer behind server s in $\mathscr{S}$. Using (4.14), we may state:

$$F^{\mathscr{S}'}_{s,w+1} \leq F^{\mathscr{S}'}_{s+1,w} \leq F^{\mathscr{S}}_{s+1,w} \leq F^{\mathscr{S}}_{s,w+1}$$

– $\beta(s) > 0$ and at least one slot behind s is free in $\mathscr{S}$: then server s is freed immediately after processing workpiece $w+1$. (4.15) implies

$$F^{\mathscr{S}'}_{s,w+1} = S^{\mathscr{S}'}_{s,w+1} + d_{s,w+1} \leq S^{\mathscr{S}}_{s,w+1} + d_{s,w+1} \leq F^{\mathscr{S}}_{s,w+1}$$

– $\beta(s) > 0$ and no slot behind s is free in $\mathscr{S}$: then workpiece $w+1$ can proceed only as soon as workpiece $w - (\beta(s) - 1)$ leaves the buffer behind server s, that is, starts on server $s+1$. This corresponds to date $S_{s+1,w-\beta(s)+1}$. Since $\forall s,\ \beta'_j(s) \geq \beta(s)$, using hypothesis (4.13), we may state:

$$F^{\mathscr{S}'}_{s,w+1} = S^{\mathscr{S}'}_{s+1,w-\beta'_j(s)+1} \leq S^{\mathscr{S}}_{s+1,w-\beta(s)+1} = F^{\mathscr{S}}_{s,w+1} \tag{4.18}$$

In conclusion, both (4.16) and (4.17) hold true in each case. □

Theorem 12: Let $\mathscr{S}(\mathcal{S},\mathcal{W},\beta)$ and $\mathscr{S}'(\mathcal{S},\mathcal{W},\beta'_j)$ denote the two fastest no-wait schedules defined by β and β'_j respectively. Then, it holds true that every processing date in $\mathscr{S}'$ is at least as early as the corresponding processing date in $\mathscr{S}$, i.e.:

$$\forall s \in \mathcal{S},\ \forall w \in \mathcal{W},\ S^{\mathscr{S}'}_{s,w} \leq S^{\mathscr{S}}_{s,w} \tag{4.19}$$

$$\forall s \in \mathcal{S},\ \forall w \in \mathcal{W},\ F^{\mathscr{S}'}_{s,w} \leq F^{\mathscr{S}}_{s,w} \tag{4.20}$$

Proof. The claim is proved by an induction on the number of workpieces. Clearly, workpiece 1 is not blocked on any station, and no buffer comes into play. Therefore:

$$\forall s \in \mathcal{S},\ S^{\mathscr{S}'}_{s,1} = S^{\mathscr{S}}_{s,1} \text{ and } F^{\mathscr{S}'}_{s,1} = F^{\mathscr{S}}_{s,1} \tag{4.21}$$

We now assume that for all workpieces up to a certain $w \in \mathcal{W}$, the following property holds:

$$\forall s \in \mathcal{S}\ S^{\mathscr{S}'}_{s,w} \leq S^{\mathscr{S}}_{s,w} \text{ and } F^{\mathscr{S}'}_{s,w} \leq F^{\mathscr{S}}_{s,w} \tag{4.22}$$

Lemma 1 states that $S^{\mathscr{S}'}_{1,w+1} \leq S^{\mathscr{S}}_{1,w+1}$. Moreover, since we also have:

- $\forall s \in \mathcal{S},\ \forall k \leq w,\ S^{\mathscr{S}'}_{s+1,k} \leq S^{\mathscr{S}}_{s+1,k}$ and
- $\forall s \in \mathcal{S},\ F^{\mathscr{S}'}_{s+1,w} \leq F^{\mathscr{S}}_{s+1,w}$

one may use Lemma 2 to successively show for $i = 1,\ldots,S-1$ that:

$$F^{\mathscr{S}'}_{i,w+1} \leq F^{\mathscr{S}}_{i,w+1} \text{ and } S^{\mathscr{S}'}_{i+1,w+1} \leq S^{\mathscr{S}}_{i+1,w+1}$$

The induction of the proof is depicted in Figure 4.3. The symbols S and F in the table mean that the corresponding starting or finishing date in schedule $\mathscr{S}'$ is no greater than it is in schedule $\mathscr{S}$. The dates in blue symbolize the necessary and sufficient conditions for the next step.

$F^{\mathscr{S}'}_{S,w+1} \leq F^{\mathscr{S}}_{S,w+1}$ then trivially follows from $S^{\mathscr{S}'}_{S,w+1} \leq S^{\mathscr{S}}_{S,w+1}$. □

Corollary 3: Let two schedules $\mathscr{S}(\mathcal{S},\mathcal{W},\beta)$ and $\mathscr{S}'(\mathcal{S},\mathcal{W},\beta')$ be defined as in Theorem 12. If $W_0 = 0$, then the throughput $\mathrm{TH}(\mathscr{S}')$ realized by schedule $\mathscr{S}'$ is no smaller than $\mathrm{TH}(\mathscr{S})$.

Proof. This follows from the definition of a throughput.

$$\mathrm{TH}(\mathscr{S}') = \frac{W}{F^{\mathscr{S}'}_{S,W}} \geq \frac{W}{F^{\mathscr{S}}_{S,W}} = \mathrm{TH}(\mathscr{S})$$ □

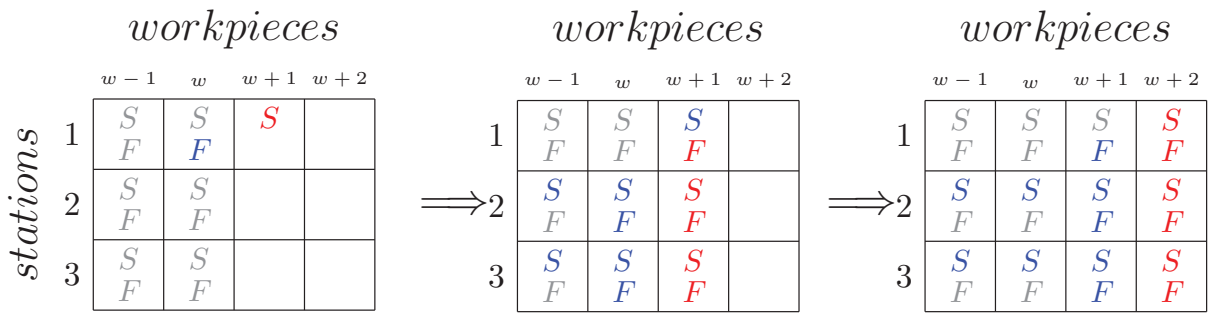


Figure 4.3: Illustration of the induction steps, combining Lemmata 1 and 2

Hence, in the no-warm-up case, the combinatorial cuts (4.10) may be used to speed-up the solution process.

4.2.3 Individual Bounds by Subsystems

Weiss and Stolletz [88] underline the need for strong lower bounds for the resolution process. Indeed, finding a feasible solution implies that only smaller buffer allocations need to be considered. On the other hand, it may only be concluded that a given amount of buffers is insufficient after cutting off all the different combinations. Therefore, lower bounds potentially yield greater time savings. Furthermore, they argue that the impact of lower bounds is stronger when they apply to only one variable ($X_i \geq L$), rather than to the sum over the whole line ($\sum_{s=1}^{S-1} X_i \geq L$). They numerically show that such "individual" lower bounds offer better performances than using the actual optimum as a "global" lower bound.

Levantesi et al. [63] first introduced a method based on subsystems of length 2 to obtain an initial buffer allocation, which they modify in a second step. The main idea of such a method is to solve the problem for a shorter sub-line, composed of servers say i to j. The solution of the sub-line can then be utilized as an individual lower bound of the form $X_i \geq L$ for the original setting. Their method was extended by Weiss and Stolletz [88] to subsystems of increasing lengths: after solving all $S - l + 1$ subsystems of length l, they solve the $S - l$ subsystems of length $l + 1$ using the previous optimal solutions as lower bounds: $\forall i \in \{1, \ldots, S - l + 1\}$, $\sum_{s=i}^{i+l-1} X_s \geq L_i$ (Fig. 4.4). Note that subsystems of length $l > 2$ do not provide individual lower bounds; their optimum is a bound on the sum of all the buffers they contain.

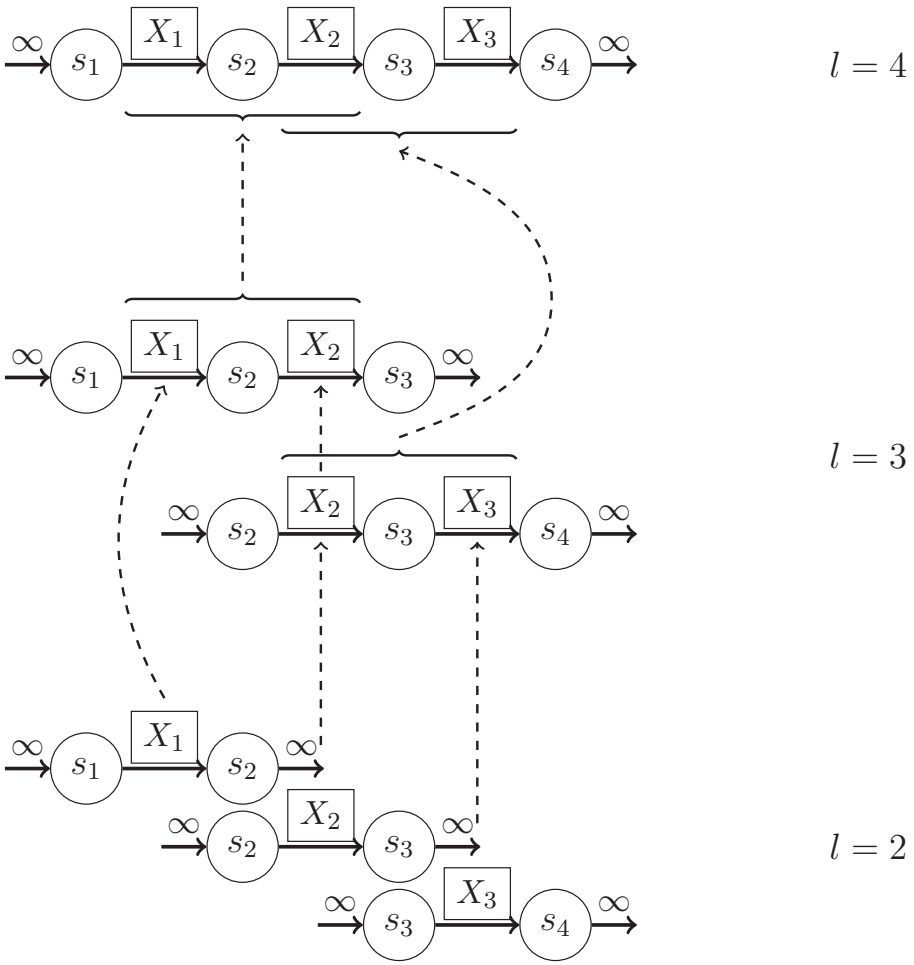


Figure 4.4: Obtaining lower bounds through Weiss and Stolletz' subsystems method [88] for a line of length 4.

This method was experimentally shown [88] to strongly reduce computation times. A subsystem-based method of course relies on the assumption that the throughput of a sub-line is necessarily no smaller than that of the corresponding full line. This can easily be shown to hold true in case no warming-up is performed.

Theorem 13: If $W_0 = 0$, then the throughput of any sub-line $\mathcal{L}[a, b]$ of a line $\mathcal{L}$, consisting of stations $\{a, a+1, \ldots, b\}$, is no smaller than the throughput of $\mathcal{L}$.

Proof. This theorem resembles some of the results presented in Section 3.5. For the sake of completeness, we give a quick proof of the theorem nonetheless. By definition, assuming that workpiece 1 starts processing on server a at time 0:

$$\mathrm{TH}(\mathcal{L}\,[a,b]) = \frac{W}{F_{b,W}(\mathcal{L}\,[a,b])} \tag{4.23}$$

Since for any servers a and b: $F_{S,W}(\mathcal{L}) \geq F_{b,W}(\mathcal{L}\,[a,b])$, one may state that

$$\mathrm{TH}(\mathcal{L}) = \frac{W}{F_{S,W}(\mathcal{L})} \leq \frac{W}{F_{b,W}(\mathcal{L}\,[a,b])} = \mathrm{TH}(\mathcal{L}\,[a,b])$$

which proves the claim. □

4.3 Generalization to the Case Using Warm-up

The most common case in literature, is the use of a warm-up phase before measuring the performance of a flow line. This allows for more realistic throughputs, since substantial amounts of time may be required to fill the line with workpieces and reach a form of steady state. However, a warm-up phase poses some serious computational challenges when looking for exact solutions. In this section, we investigate these difficulties and propose alternatives to overcome them.

4.3.1 Issues with the Previous Methods

First of all, it should be noted that a throughput is now written differently. Suppose that we have $0 < W_0 < W$ workpieces during the warm-up phase. Since the throughput corresponds to the amount of workpieces divided by the time taken to process them on all servers, it should now be written:

$$\mathrm{TH}(\mathscr{S}) = \frac{W - W_0}{F^{\mathscr{S}}_{S,W} - F^{\mathscr{S}}_{S,W_0}} \tag{4.24}$$

We show that the combinatorial cuts (4.10) defined in Section 4.2.2 may no longer be applied to compute exact solutions when $W_0 > 0$.

Theorem 14: Let two schedules $\mathscr{S}(\mathcal{S}, \mathcal{W}, \beta)$ and $\mathscr{S}'(\mathcal{S}, \mathcal{W}, \beta')$ be defined as in Corollary 3. If $W_0 > 0$, then the throughput $\mathrm{TH}(\mathscr{S}')$ realized by schedule $\mathscr{S}'$ may be strictly lower than $\mathrm{TH}(\mathscr{S})$.

Proof. We prove the claim by constructing an example where $\mathrm{TH}(\mathscr{S}') < \mathrm{TH}(\mathscr{S})$. Let $S = 6$, $W = 6$, $W_0 = 3$. Consider the sample of processing

$d_{s,w}$	w_1	w_2	w_3	w_4	w_5	w_6
s_1	0.10	0.11	0.12	0.20	0.20	0.20
s_2	0.10	0.11	0.12	0.20	0.20	0.20
s_3	0.10	0.11	0.12	0.20	0.20	0.20
s_4	0.10	0.005	0.30	0.20	0.20	0.20
s_5	0.30	0.11	0.12	0.20	0.20	0.20
s_6	0.10	0.11	0.12	0.20	0.20	0.20

Table 4.1: sample of processing times of the counter-example in Theorem 14

	s_1	s_2	s_3	s_4	s_5
$\beta(s)$	0	0	0	0	0
$\beta'(s)$	0	0	0	1	0

Table 4.2: buffer distributions used in the counter-example in Theorem 14

times $d_{s,w}$ given by Table 4.1 and the buffer capacity distributions given by Table 4.2.

In this example, the throughput $\mathrm{TH}(\mathscr{S})$ using β amounts to 3.95, whereas the throughput $\mathrm{TH}(\mathscr{S}')$ using β' equals only 3.66, which proves the claim. □

This counter intuitive result can be explained by the definition of the throughput (4.24). For fixed values of W and W_0, the throughput is entirely determined by the difference of both finishing dates $F_{S,W}$ and F_{S,W_0}. While the addition of buffer slots cannot increase any of these values, as stated by Theorem 12, it may however decrease, as in the previous example, only F_{S,W_0} without modifying $F_{S,W}$. This leads to an unwanted decrease of the throughput. Subsequently, when applying the combinatorial cuts (4.10) to an initial buffer allocation that comprises at least one buffer slot, one may cut off optimal solutions, or even erroneously call a feasible problem infeasible. Only when starting from an allocation which was proved to be a lower bound, or from all empty buffers, may the cuts (4.10) be used.

Secondly, the method presented in Section 4.2.3, too, is invalid when $W_0 > 0$. Indeed, the assumption it is based on, and which was stated in Theorem 13, cannot be extended to the case with warm-up.

Theorem 15 (Sub-line Paradox): If $W_0 > 0$, then the throughput of a sub-line $\mathcal{L}\,[a, b]$ of a line $\mathcal{L}$ may be lower than the throughput of $\mathcal{L}$.

$d_{s,w}$	w_1	w_2	w_3	w_4	w_5	w_6
s_1	1.05	0.85	0.40	0.40	0.45	0.45
s_2	1.05	0.85	0.40	0.40	0.45	0.45
s_3	0.30	0.50	0.70	0.70	0.70	0.70
s_4	1.00	0.95	0.40	0.40	0.40	0.45
s_5	1.05	0.85	0.40	0.40	0.45	0.45
s_6	1.05	0.85	0.40	0.40	0.45	0.45

Table 4.3: sample of processing of the counter-example in Theorem 15

	s_1	s_2	s_3	s_4	s_5
$\beta(s)$	0	0	0	0	0

Table 4.4: buffer distribution of the counter-example in Theorem 15

Proof. We prove the claim by constructing an example where $\mathrm{TH}(\mathcal{L}[a,b]) < \mathrm{TH}(\mathcal{L})$. Let $S = 6$, $W = 6$, $W_0 = 2$. Consider the sample of processing times $d_{s,w}$ given by Table 4.3

Note that in this example, all servers have identical mean processing times (0.6). We aim to evaluate and compare, for the buffer distribution given by Table 4.4, the throughput of $\mathcal{L}$, with that of subsystem $\mathcal{L}[3,4]$, which is composed only of servers 3 and 4.

For this sample of processing times, we obtain $\mathrm{TH}(\mathcal{L}[3,4]) = 1.57$ and $\mathrm{TH}(\mathcal{L}) = 1.95$. This proves the claim. □

Once more, the explanation for this surprising result is to be found in the new definition of throughput (4.24). It is the difference between $F_{S,W}$ and F_{S,W_0}, not the makespan $F_{S,W}$ alone, that determines the throughput. In the case of sub-line $\mathcal{L}[3,4]$, due to the high values of $d_{3,w}$ where $w > 2$, the difference $F_{S,W} - F_{S,W_0}$ amounts to 2.55 units of time. On the full line however, the smaller values of $d_{6,w}$, $w > 2$, allow for a difference of only 2.05 units of time. Hence, the bounds computed using the subsystems method may be too strong, and cut off optimal solutions.

Subsequently, neither of the two key elements that made the BAP tractable in the particular $W_0 = 0$, that is, the combinatorial cuts (4.10) and the lower bounds described in 4.2.3, may be generalized to the warm-up case. The former are restricted to uses starting from lower bounds or an empty allocation, while the latter simply become incorrect. Without further improvements to the

solution process, the exact solving of the BAP seems intractable for lines with a greater number of stages.

4.3.2 Alternative Definition for the Throughput

The main lesson that can be drawn from Section 4.3.1 is that the throughput as defined by (4.24) may not be a good measure for the efficiency of buffer allocations. This is due to the fact that when $W_0 > 0$, throughputs are no longer monotonously increasing with the adding of buffer slots. In this section, we aim to find a monotonously increasing measure for the evaluative sub-problem. In particular, we expect such a measure to enable the use of combinatorial cuts for any initial buffer distribution. The definition (4.24) used for the goal throughput TH^* in the formulation of the BAP is of course left unchanged.

Definition 9: We call *zero allocation* and denote β^0 the buffer allocation such that $\forall s \in \mathcal{S}$, $\beta^0(s) = 0$. Similarly, the *infinite allocation* is denoted β^∞ and is defined as follows: $\forall s \in \mathcal{S}$, $\beta^\infty(s) = \min\{B_s, W - 1\}$. The corresponding fastest no-wait schedules are written $\mathscr{S}^0$ and $\mathscr{S}^\infty$ respectively.

Let us now consider a function TH^+ defined as follows:

$$\mathrm{TH}^+(\mathscr{S}) = \frac{W - W_0}{F^{\mathscr{S}}_{S,W} - F^{\mathscr{S}^0}_{S,W_0}} \tag{4.25}$$

Throughout this work, TH^+ is referred to as *augmented throughput*. First of all, it should be noted that in contrast to the throughput as defined in (4.24), an augmented throughput may take a negative value, or be undefined. Indeed, the value of $F^{\mathscr{S}^0}_{S,W_0}$ is possibly greater than, or equal to $F^{\mathscr{S}}_{S,W}$. This is an unwanted side effect. Although this can be the case for any line and any values of W and W_0, the likeliness of TH^+ to be negative becomes very small when the $\frac{W_0}{W}$ ratio is well below 1. As it is always easy enough to verify whether or not it is the case, we assume in the following sections that for any instance we try to solve, the following holds true:

$$F^{\mathscr{S}^\infty}_{S,W} > F^{\mathscr{S}^0}_{S,W_0} \tag{4.26}$$

An interesting property is that TH^+ is always an upper bound for TH. Indeed, using Theorem 12, one may state that for any fastest no-wait schedule $\mathscr{S}$,

$F^{\mathscr{S}^0}_{S,W_0} \geq F^{\mathscr{S}}_{S,W_0}$. Therefore, if (4.26) holds true, then:

$$\forall \mathscr{S},\ \mathrm{TH}^+(\mathscr{S}) \geq \mathrm{TH}(\mathscr{S}) \tag{4.27}$$

Furthermore, it is easily shown that augmented throughputs are monotonously increasing with the adding of buffer slots; indeed, the term $F^{\mathscr{S}^0}_{S,W_0}$ is constant. Adding buffers decreases only the $F^{\mathscr{S}}_{S,W}$ term, thus increasing the value of $\mathrm{TH}^+(\mathscr{S})$, even if $\mathrm{TH}(\mathscr{S})$ is decreased. Hence, Corollary 3 may be extended:

Theorem 16: Let two schedules $\mathscr{S}(\mathcal{S}, \mathcal{W}, \beta)$ and $\mathscr{S}'(\mathcal{S}, \mathcal{W}, \beta')$ be defined as in Theorem 12. Then the augmented throughput $\mathrm{TH}^+(\mathscr{S}')$ realized by schedule $\mathscr{S}'$ is no smaller than $\mathrm{TH}^+(\mathscr{S})$, even when $W_0 > 0$. □

Hence, this new definition offers good potential for the extension of combinatorial cuts. When replacing Constraint (4.5) in (Sub-Problem) by:

$$F^{\mathscr{S}}_{S,W} \leq \frac{W - W_0}{\mathrm{TH}^*} + F^{\mathscr{S}^0}_{S,W_0} \tag{4.28}$$

the obtained evaluative method may no longer cut off optimal allocations, even when applied to an initial non-zero buffer allocation. Of course, the fact that $\mathrm{TH}^+ \geq \mathrm{TH}$ also means that the augmented throughput is a weaker measure, in the sense that it results in weaker lower bounds. Thus, the gain in generality is paid with a loss of performance. Note that a corresponding *lowered throughput* may be defined analogously, and presents some interesting properties as well.

Definition 10: We call *lowered throughput* and denote TH^- the performance measure defined as follows:

$$\mathrm{TH}^-(\mathscr{S}) = \frac{W - W_0}{F^{\mathscr{S}}_{S,W} - F^{\mathscr{S}^\infty}_{S,W_0}} \tag{4.29}$$

In contrast to the augmented throughput, TH^- is strictly positive provided that no processing time is zero, and $W_0 < W$. No assumption of the type (4.26) is required in this case.

Similarly to (4.27) and to Theorem 16, the following properties hold:

Theorem 17: For any two fastest no-wait schedules $\mathscr{S}$ and $\mathscr{S}'$ defined as in Theorem 12, we have:

$$\mathrm{TH}^-(\mathscr{S}') \geq \mathrm{TH}^-(\mathscr{S}) \tag{4.30}$$

$$\mathrm{TH}^-(\mathscr{S}) \leq \mathrm{TH}(\mathscr{S}) \tag{4.31}$$

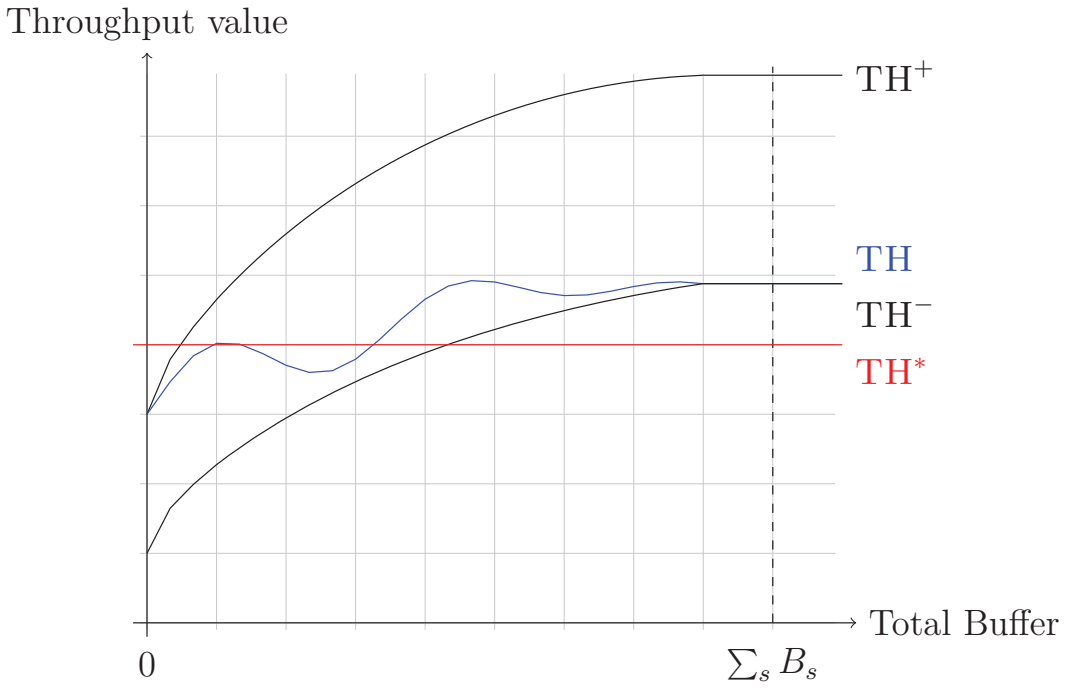


Figure 4.5: Graphical representation of the augmented, normal and lower throughputs when successively adding buffers to the zero allocation

In addition, analogously to Theorem 19, if there exists a fastest no-wait schedule $\mathscr{S}$ whose buffer sum equals k such that $\mathrm{TH}^-(\mathscr{S}) \geq \mathrm{TH}^*$, then k is an upper bound on the total amount of buffer slots. Similarly to the augmented throughput, this property can be used to determine that an initial allocation β_{ini} is a global upper bound.

For the sake of clarity, Figure 4.5 depicts a plotting of TH^-, TH and TH^+ with their known properties for a constructed instance of the BAP, and an arbitrary TH^* value. Note that the total buffer on the x-axis represents one particular way to add buffers successively to the zero allocation. Other allocations of equal buffer sum could lead to different results.

As it can be seen on Figure 4.5, any throughput starts with the same value as TH^+, and takes the value of TH^- at infinite buffers. Once the values of TH and TH^- coincide for a given buffer sum, it remains so when adding more buffers. While both TH^+ and TH^- are non-decreasing, so too is the gap $(\mathrm{TH}^+ - \mathrm{TH}^-)$. In general, TH is not monotonous. All three measures reach a point, at a certain buffer allocation, after which they remain constant. This point is identical for the augmented and lower throughputs, and may be reached either at exactly $\sum_{s=1}^{S-1} B_s$ or before this value.

4.3.3 Extending the Subsystems' Method

As seen in Section 4.3.1, it is incorrect to use evaluations on subsets of the servers for obtaining lower bounds on the whole line. However, the idea behind that method, namely, successively solving the BAP on smaller instances, may still be transferred to the case $W_0 > 0$.

Definition 11: Let $\mathcal{U} = \{s_1, \ldots, s_l\} \subset \mathcal{S}$ be a subset of l consecutive stations. For $a_1, \ldots, a_l \in \mathbb{N}$, we denote by $\beta_{\mathcal{U}}[a_1, \ldots, a_l]$ the buffer allocation β such that:

- $\forall s \in \mathcal{U},\ \beta_{\mathcal{U}}(s) = a_s$
- $\forall s \notin \mathcal{U},\ \beta_{\mathcal{U}}(s) = \min\{B_s, W - 1\}$

The corresponding fastest no-wait schedule is then denoted $\mathscr{S}_{\mathcal{U}}[a_1, \ldots, a_l]$.
Note that any buffer allocation β may be written in the form: $\beta_{\mathcal{S}}[a_1, \ldots, a_S]$.

Theorem 18: For a given instance I of the BAP, let $\beta_{\mathcal{U}}$ be the buffer allocation minimizing the buffer sum over all TH^+-feasible allocations of the form $\beta_{\mathcal{U}}[a_1, \ldots, a_l]$ where $a_1, \ldots, a_l \in \mathbb{N}$. Then the sum $A = \sum_{i=1}^{l} a_i$ is a lower bound on the amount of buffers behind the stations of $\mathcal{U}$ for any TH^+-feasible allocation.

Proof. Let $\mathcal{U} \subset \mathcal{S}$ such that $|\mathcal{U}| = l$ and $|\mathcal{S}| = S$.
Let $\beta_{\mathcal{U}}$ minimize $\beta_{\mathcal{U}}[a_1, \ldots, a_l]$ and β_S minimize $\beta_{\mathcal{S}}[b_1, \ldots, b_S]$.
Assume that there exists a station $s^* \in \mathcal{U}$ such that $\beta_{\mathcal{S}}(s^*) < \beta_{\mathcal{U}}(s^*)$. Then a new allocation $\beta'_{\mathcal{U}}$ can be defined as follows:

- $\forall s \neq s^* \in \mathcal{U},\ \beta'_{\mathcal{U}}(s) = \beta_{\mathcal{U}}(s)$
- $\beta'_{\mathcal{U}}(s^*) = \beta_{\mathcal{S}}(s^*)$

By definition, it holds true that:

$$\forall s \in \mathcal{S} \setminus \mathcal{U},\ \beta'_{\mathcal{U}}(s) > \beta_{\mathcal{S}}(s)$$

Moreover, by construction, it holds true that:

$$\forall s \in \mathcal{U},\ \beta'_{\mathcal{U}}(s) > \beta_{\mathcal{S}}(s)$$

Hence, Theorem 16 states that $\mathrm{TH}^+(\mathscr{S}'_{\mathcal{U}}) \geq \mathrm{TH}^+(\mathscr{S}_{\mathcal{S}})$. This implies that $\beta'_{\mathcal{U}}$ is also TH^+-feasible, which contradicts the minimality of $\beta_{\mathcal{U}}$. Therefore, the assumption that such a s^* exists is false. For each $s \in \mathcal{U}$, it necessarily holds true that $\beta_{\mathcal{S}}(s) \geq \beta_{\mathcal{U}}(s)$. □

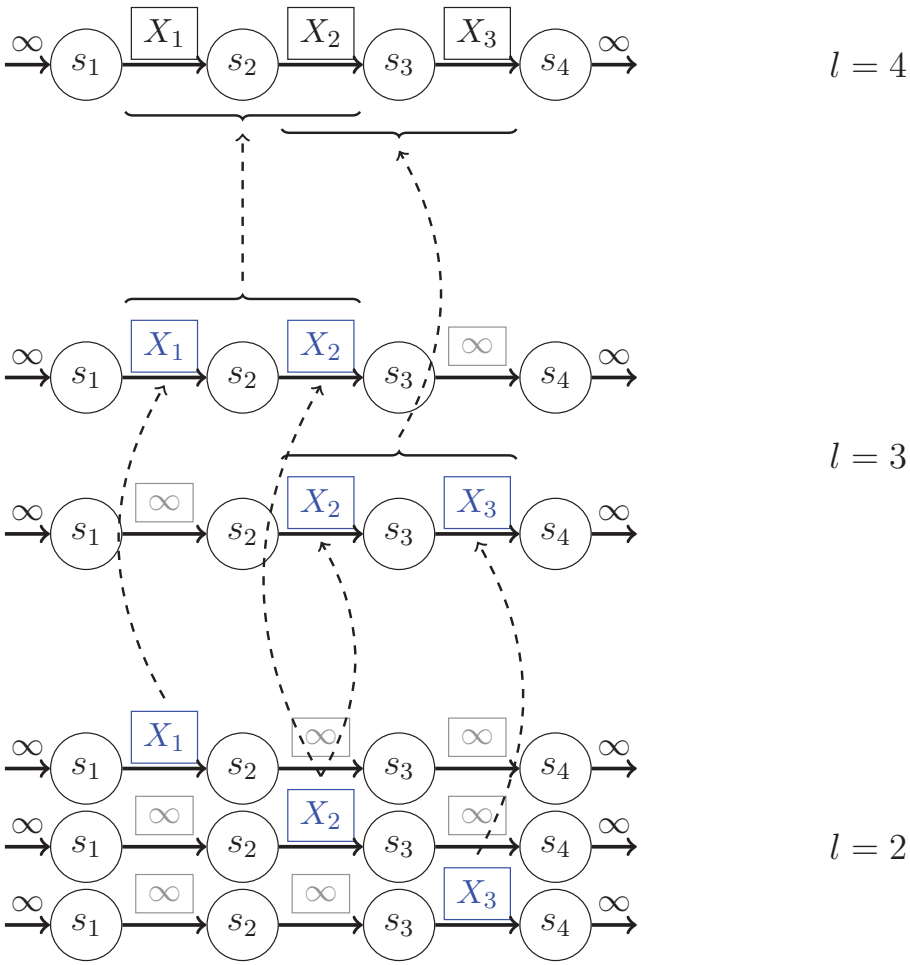


Figure 4.6: Obtaining lower bounds through the extended subsystems method for a line of length 4.

With this result comes a new possible algorithm: An instance of the BAP can be solved for the whole line where only l of the $S-1$ buffers are actually variable, the other buffers being fixed at their maximum values, until the augmented throughput TH^{+} reaches the goal TH^{*}. The thus obtained buffer sums are then, according to Theorem 18, lower bounds for the original formulation of the BAP. Once more, one may do this for increasing values of l. The process is depicted on Figure 4.6.

In comparison with the method described in Section 4.2.3, there are a few differences besides the correctness of the method when $W_0 > 0$. First, as the instances of the BAP are solved on complete lines instead of sub-lines, the needed computational effort can be expected to be greater here, despite the fact that the smaller amounts of variables make the subsystems far easier to solve than the original formulation. Furthermore, since the augmented throughput

is used as a measure of performance instead of the classical definition, the solving process is expected to terminate with lower buffer allocations, resulting in weaker bounds. This can be seen as a trade-off for the correctness of the algorithm.

Nonetheless, this new method also presents substantial advantages. First, the buffers that serve as variables no longer need to be consecutive: any combination of l servers may be used. This results in more bounds, and therefore potentially less combinations of buffers to be evaluated. Moreover, in contrast to Section 4.2.3, the extended subsystem method may be applied to obtain upper bounds; for this, the values $\beta_{\mathcal{U}}(s)$ are set to 0 instead of $\min\{B_s, W-1\}$ for $s \notin \mathcal{U}$. Upper bounds are not likely to be of great use when they apply to great amounts of variables, as the method progressively adds buffers to an initial low allocation until an optimum is found. However, individual lower bounds may still cut off large numbers of incumbent nodes. On top of that, they can replace the values of $\beta_{\mathcal{U}}(s)$, $s \notin \mathcal{U}$, if they are lower than the value $\min\{B_s, W-1\}$ during the computation of lower bounds, which may actually result in stronger bounds. The efficiencies of the different variations of this method are presented and compared in our numerical study in Section 4.5.

4.4 Using Global Lower Bounds

So far in Sections 4.3.2 and 4.3.3, the augmented throughput was employed as a way to generalize different techniques that would otherwise be permitted only in special cases. In this section, we show a different way to exploit this new definition. We present an alternative resolution approach that is based on the TH^+ measure. The performance of this new algorithm is compared with the classical approach in Section 4.5.

First, the following result flows directly from the monotonicity of the augmented throughput:

Theorem 19: If for a given positive integer k, each buffer allocation whose sum equals k has its augmented throughput strictly lower than TH^*, then $k+1$ is a lower bound on the total amount of buffer slots.

Proof. For each fastest no-wait schedule $\mathscr{S}$ whose buffer sum is no greater than k, there exists a fastest no-wait schedule $\mathscr{S}'$ whose buffer sum is exactly k such that:

$$\mathrm{TH}(\mathscr{S}) \leq \mathrm{TH}^{+}(\mathscr{S}) \leq \mathrm{TH}^{+}(\mathscr{S}')$$

Since by assumption it holds true that $\mathrm{TH}^{+}(\mathscr{S}') < \mathrm{TH}^{*}$, it follows that $\mathrm{TH}(\mathscr{S}) < \mathrm{TH}^{*}$. □

This property enables us to exploit an initial buffer allocation β_{ini} that is supposedly close to the optimum. Indeed, with just TH as a measure, it is impossible to determine whether there exists a buffer allocation β that is smaller than β_{ini} yet feasible. On the other hand, by computing the augmented throughput of each allocation of equal sum to β_{ini}, using Theorem 19, one may be certain that it is a global lower bound. An alternative exact solving algorithm can even be derived along the following steps:

1. Find an upper bound using a heuristic method
2. Iterate down from the upper bound and find the highest buffer sum such that each allocation has $\mathrm{TH}^{+} < \mathrm{TH}^{*}$ (global lower bound)
3. Iterate up from the global lower bound and find the lowest buffer allocation such that $\mathrm{TH} \geq \mathrm{TH}^{*}$

For the sake of clarity, this process is illustrated on Figure 4.7.

The literature on heuristic solving approaches is dense, and some references are given in Section 4.1. Through this work, for the sake of simplicity, we opt for an algorithm that was used by Levantesi et al. [63]. It is a straightforward approach that counts among the gradient methods; buffers are iteratively added to the zero allocation behind the station where it most improves the throughput. It terminates as soon as $\mathrm{TH} > \mathrm{TH}^{*}$, and a major advantage it presents is that the buffer distribution may be saved for each total amount of buffers. This will prove useful in step 2. Step 1 is carried out using Algorithm 2.

In Algorithm 2, it is implied in Line 4 that each buffer is successively increased, evaluated and decreased. The performances can then be compared, and the best server is chosen.

The second step, namely, obtaining a global lower bound, can then be performed by Algorithm 3.

In Algorithm 3, the two *While*-loops look similar; both determine whether there exists a distribution of sum i such that its augmented throughput is feasible.

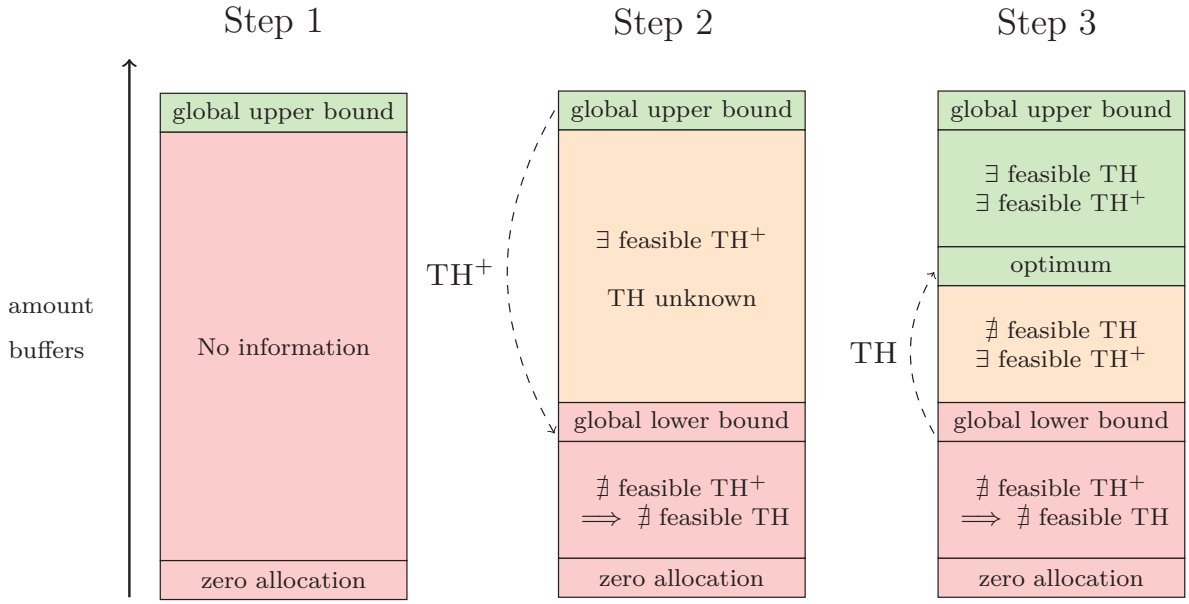


Figure 4.7: First, an upper bound is found. Then, TH^+ is used to iterate down to a lower bound. Finally, the optimum can be computed.

The difference is that in Line 6, no computation is done and a heuristic solution is used. On the other hand, in Line 11, an exponential amount of allocation is generated and evaluated. If the heuristic solutions are strong enough, only 1 BAP needs to be solved, which significantly saves time.

Finally, Algorithm 4 shows how the previously computed lower bound can be used in order to compute the optimum in a faster way. This corresponds to the classical solving approach.

Compared with the basic method described in Section 4.2, this new algorithm computes extra heuristic buffer allocations for increasing total buffer amounts.

Algorithm 2 Step 1: Find a heuristic feasible solution to the BAP and remember all intermediate allocations

1: $i \leftarrow 1$
2: $\beta \leftarrow \beta^0$
3: **while** $\mathrm{TH} < \mathrm{TH}^*$ **do**
4: Find buffer b where one extra buffer slot most increases TH
5: $\beta(b) \leftarrow \beta(b) + 1$
6: $h[i] \leftarrow \beta$
7: $i \leftarrow i + 1$
8: **return** h

Algorithm 3 Step 2: Compute a global lower bound using a table of heuristic solutions $h[k]$ of k buffers

1: $\beta \leftarrow h[n]$ where n is the highest index of h
2: $i \leftarrow \sum_k \beta(k)$
3: feasible $\leftarrow$ true
4: **while** feasible **do**
5: $\quad i \leftarrow i - 1$
6: $\quad \beta \leftarrow h[i]$
7: $\quad$ **if** β is TH^+-infeasible **then**
8: $\quad\quad$ feasible $\leftarrow$ false
9: feasible $\leftarrow$ true
10: **while** feasible **do**
11: $\quad \beta \leftarrow$ solution of BAP instance with exactly i buffers and TH^+ as measure
12: $\quad$ **if** BAP was TH^+-infeasible **then**
13: $\quad\quad$ feasible $\leftarrow$ false
14: $\quad i \leftarrow i - 1$
15: **return** $(i + 1)$

Also, in Step 2, BAP instances are iteratively solved for decreasing sums until a lower bound is found. This additional computational effort is compensated by the fact that total buffers that are lower than the bound computed in Step 2 no longer need to be evaluated. Hence, the approach using global bounds can be expected to perform better than the basic approach if the chosen heuristic performs well, and if TH^+ is not too different from TH.

4.5 Numerical Study

4.5.1 Methodology

In this section, we aim to investigate the efficiency of the full line lower bounds and that of the global bounds based algorithm presented in Section 4.4. For this purpose, we solve the BAP in the warm-up case using three different methods:

- the classical approach without bounds
- the classical approach using adapted subsystem bounds

Algorithm 4 Step 3: Solve the BAP to optimality starting from a global lower bound k

```
1: i ← k
2: infeasible ← true
3: while infeasible do
4:     i ← i + 1
5:     β ← solution of BAP instance with exactly i buffers and TH as measure
6:     if BAP was TH-feasible then
7:         infeasible ← false
8: return β
```

- the global bound approach using adapted subsystem bounds

In order to dilute extreme values, 16 samples of processing times are randomly generated. We simulate an exponential distribution using Descriptive Sampling, a method based on a deterministic input. Saliby [77, 78] shows that descriptive sampling reduces variability, while Stolletz and Weiss [84] argue that particularly robust results are obtained for the BAP. Each setting is solved for every sample, and the mean total buffer and mean computation time are compared. All computations were carried out on a an Intel(R) Xeon(R) CPU E5-2690 with 2.9 GHz and 192 GB RAM. Implementation was done in Python 3.6, and the solving of the MILPs was performed by Gurobi 8.0 with default settings. Gurobi was restricted to run on a single core.

In this section, only the impact of bounds and the type of algorithm are considered. However, more parameters were tested during the numerical experiments. Each setting was run with and without an initial heuristic upper bound (using the procedure by Levantesi et al. [63], mentioned in Section 4.4). Moreover, each case was run with and without upper bounds, obtained by the same subsystem-based method as the lower bounds. In this study, all settings include the initial heuristic and none includes the upper bounds; the latter, unfortunately, turned out to greatly increase computation times instead of reducing them. For the sake of completeness, the full study can be found in Appendix A.

4.5.2 Impact of the Adapted Lower Bounds

Computation times are strongly correlated to the value of the optimal total buffer, which in turn is tied to the goal throughput. Hence, different values for

TH^*	Mean optimum	Minimum time	Mean time	Maximum time
4.5	6.0	62.50	90.57	289.80
5.0	11.1	266.35	333.37	425.38
5.5	19.0	884.91	1038.96	1215.06
6.0	35.8	3239.40	4156.89	5441.01

Table 4.5: Mean buffers and computation times of the classical method without lower bounds for different goal throughput values

TH^*	Mean optimum	Minimum time	Mean time	Maximum time
4.5	6.0	69.70	73.35	93.69
5.0	11.1	61.14	97.21	129.26
5.5	19.0	285.43	683.12	1002.40
6.0	35.8	1242.30	1977.30	3324.46

Table 4.6: Mean buffers and computation times of the classical method using full line lower bounds for different goal throughput values

the goal throughput are put to test. In contrast, the setting of the line itself is left constant, with $S = 5$ stations and $W = 10,000$ workpieces. Table 4.5 shows the results that were obtained for the classical approach without lower bounds, while the results using bounds are found in Table 4.6.

It can be read from those tables that the new lower bounds still account for substantial time savings. Since they are computed using the whole line rather than sub-lines, they cannot be expected to work as fast as proper subsystem lower bounds. Nonetheless, they do speed up the solution process by over 50% on average for $TH^* = 6.0$ and up to 70% for $TH^* = 5.0$. The minimum and maximum computation times are more unbalanced than the mean values, and while greater differences are observed, it should also be noted that the minimum time for $TH^* = 4.5$ is the only case where the no-bound case (62.50s) performed better than the lower bounds (69.70s).

4.5.3 Comparison with the Global Bounds Method

Now, we aim to compare the best results obtained for the classical approach with those of our new algorithm. The same setting with $S = 5$ and $W = 10,000$ is kept, and the values are computed for the same goal throughputs. Table 4.7

TH^*	Mean optimum	Minimum time	Mean time	Maximum time
4.5	6.0	39.66	62.64	68.86
5.0	11.1	100.25	108.25	131.72
5.5	19.0	227.29	293.04	372.57
6.0	35.8	1330.51	1807.51	2274.83

Table 4.7: Mean buffers and computation times of the global bound algorithm using full line lower bounds for different goal throughput values

shows the results that were obtained for the global bounds algorithm using the lower bounds from the previous subsection.

Compared to Table 4.6, computation times seem to be further decreased. Certainly, in the $TH^* = 5.0$ case, all three values are slightly higher than for the classical algorithm. Still, the new method seems faster for all other goal throughputs, and performs particularly well when $TH^* = 5.5$: 293.04s versus 683.12s average time, 372.57s versus 1002.40s maximum, as well as when $TH^* = 4.5$: 39.66s versus 69.70s minimum time, 62.64s versus 90.57s average. Hence, although the global bounds approach may not be the fastest method for every sample, it does prove to be a competitive alternative in general. For a more detailed analysis, see the results in Appendix A.

4.6 Conclusions and Outlook

In this chapter, a formulation is derived for the BAP with randomly generated samples of known processing times. We show that the existing literature insufficiently deals with problems that emerge from the use of a warm-up phase during performance evaluation. We compute exact solutions by applying combinatorial cuts and subsystem-based lower bounds that are known to significantly speed up the solution process. These techniques could be adapted to the warm-up case by defining a new monotonously increasing performance measure. Subsequently, a new solving approach based on this new measure is proposed. The numerical study reveals that the cuts and the lower bounds generalized to the warm-up case still lead to substantial time savings. Moreover, we show that the algorithm based on the monotonous evaluation measure performs significantly better than the classical approach.

This study points out the key role that is played by the definition of the throughput rate, and by the chosen performance measure. Details, such as warm-

up, can make the difference between an exact solution and an approximation. Further research should be directed towards improving the definitions for evaluation measures. While our augmented throughput has the advantage of being monotonously increasing, it may overtop the actual throughput by too much. A better heuristic, combined with a less overestimating measure, would further speed up the solution process.

Part III

Scheduling with Inventory Constraints

Chapter 5

Scheduling with Prefabrication

In this chapter, we consider a new class of applied single-machine scheduling problems, in which some jobs are needed to produce parts which are then used up by other jobs. This gives rise to a dynamic form of precedence constraints, where instead of jobs having fixed predecessors and successors, all jobs that produce a certain product may be chosen as predecessor for some job that consumes that same product. Importantly, only consuming jobs are considered for the objective function: these are the jobs which are important for the end-customer. We introduce the new model for single-stage scheduling and separate polynomially solvable and $\mathcal{NP}$-hard cases.

5.1 Introduction

Precedence constraints are very common additional constraints considered for scheduling problems. Traditionally, in scheduling literature there are two subtly different ways to introduce precedence constraints into a problem. One can either straightforwardly introduce a precedence graph between jobs, in which case a job cannot be started before all its predecessors in the graph have been finished.

Alternatively, one can consider jobs which consist of several operations, and have (possibly different) precedence graphs between the operations of each job. The latter kind of constraint is usually considered in scenarios where only the finish time of the final operation for each job is of concern (e.g. shop scheduling problems). Finishing times of preceding operations do not influence the value of the objective function. Of course, both approaches to precedence constraints can also be combined.

What both ways to express precedence have in common, is that the precedence graph is part of the input. Thus, predecessors and successors are clearly defined from the start and a solution algorithm only needs to ensure the constraints are met.

In practice, however, jobs may not possess a single possible predecessor. Instead, there may be a range of predecessors and successors, which we can pair together given some constraints. This happens, for example, in production environments, where an intermediate product is produced which can be built into several different end products. A company which produces two-dimensional wood cut-outs works in two steps: first, they cut the original wooden boards into a standard size; then they use the standard sized boards to produce the final cut-out form. At the time when the standard sized wooden board is produced, it is not yet determined which form will eventually be cut from it.

This gives rise to a new scheduling model, which we call scheduling with prefabrication. Here, some jobs produce (possibly different) products, while other jobs consume these products to eventually arrive at a final customer order. In a feasible schedule, a job can only be started if all necessary intermediate products are available in the necessary quantities. Producing jobs are not assigned to consuming jobs in advance, but instead are part of the scheduling process.

Note that scheduling with prefabrication has strong similarities with another model, namely scheduling with inventory constraints (see Briskorn et al. [21]). In that model, an intermediate product is produced by some jobs and consumed by others, just as in our model. The main difference is that in scheduling with inventory constraints, all jobs, producers and consumers, are considered equally for the objective, while in scheduling with prefabrication only consumers are considered. Comparing to the traditional models, scheduling with inventory constraints can be seen as the dynamic variant of precedences between jobs, while scheduling with prefabrication could be likened to the dynamic variant of precedences between operations, where only the final produced product is important for the objective function. We will later see that the two models yield different complexity results: some problems which are hard for inventory constraints are easy for prefabrication and reverse. Details on the link between the two scheduling models will be discussed in the later parts of this chapter.

This chapter is structured as follows. In Section 5.2 we formally introduce the scheduling model with prefabrication. We consider both the general setting and the particular case for single-machine problems and exactly one intermediate product. Section 5.3 presents a short review of the existing literature on

scheduling with inventory constraints. In Section 5.4, a few preliminary results are derived for the particular case of regular objectives. The main complexity results for the different variations of the problem can be found in Section 5.5 (lateness related objectives) and Section 5.6 (completion time related objectives). This classification is justified by the fact that results obtained for maximum lateness are used for the number of tardy jobs objective. Similarly, the sum of completion times is a particular case of the weighted sum of completion times. Overall, we start considering the easier problems, and progressively move to the harder ones. Eventually, the conclusions of this chapter are drawn in Section 5.7.

5.2 Problem Definition

In this section, we introduce scheduling problems with prefabrication formally.

In the most general setting, let $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ be a set of jobs which have to be processed on any of the parallel identical machines: $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. The processing time for a job $J_j \in \mathcal{J}$ is p_j. In addition, we are given a set of goods $g \in \mathcal{G}$. For each job $J_j \in \mathcal{J}$, the quantity $\delta_j^g \neq 0$ defines the amount of intermediate material which is produced (if $\delta_j^g > 0$) or consumed (if $\delta_j^g < 0$) by J_j.

Based on the δ-values we split the set of jobs $\mathcal{J}$ into two disjoint subsets for each type $g \in \mathcal{G}$ of goods:

- $\mathcal{J}^{g+} := \left\{ J_j \in \mathcal{J} \,:\, \delta_j^g > 0 \right\}$ (set of producers),
- $\mathcal{J}^{g-} := \left\{ J_j \in \mathcal{J} \,:\, \delta_j^g < 0 \right\}$ (set of consumers).

A schedule σ maps each job $J_j \in \mathcal{J}$ to a machine M_i and to an interval $[S_j(\sigma), C_j(\sigma)]$, where $S_j(\sigma)$ is the starting time and $C_j(\sigma) = S_j(\sigma) + p_j$ the completion time of job J_j in sequence σ. We also simply write S_j and C_j if the sequence is clear from the context. Note that a schedule can also be seen as a set of sequences σ_i, where each σ_i gives the order in which jobs are processed on machine M_i. We denote the k-th element of a sequence σ_i by $\sigma_i(k)$. As it is assumed that each job is processed exactly once and that the machines are identical, the following property holds:

$$\bigcup_{i=1}^{m} \bigcup_{k} \sigma_i(k) = \mathcal{J} \tag{5.1}$$

A schedule σ is feasible only if for each sequence i, $C_{\sigma_i(k)} \leq C_{\sigma_i(k+1)}\ \forall k \geq 1$ and the available amount of each good $g \in \mathcal{G}$ is always non-negative, i.e. at any time t,

$$\sum_{J_j \in \mathcal{J}^g(t)} \delta_j^g \geq 0, \tag{5.2}$$

where $\mathcal{J}^g(t) := \{J_j \in \mathcal{J}^{g+} : C_j \leq t\} \cup \{J_j \in \mathcal{J}^{g-} : S_j \leq t\}$.

Note that we only consider feasible instances, i.e.

$$\sum_{J_j \in \mathcal{J}} \delta_j^g \geq 0 \tag{5.3}$$

Since a schedule can always been seen as a succession of producing jobs and consuming jobs, we introduce the following terms.

Definition 12 (Producer block): In a schedule a set of consecutive jobs on a machine $m \in \mathcal{M}$ producing product $g \in \mathcal{G}$ is called a *producer block* of product g, $B^{\mathrm{p}}_{g,m}$.

Definition 13 (Consumer block): In a schedule a set of consecutive jobs on a machine $m \in \mathcal{M}$ consuming product $g \in \mathcal{G}$ is called a *consumer block* of product g, $B^{\mathrm{c}}_{g,m}$.

For the sake of completeness, we give an Mixed-Integer Program (MIP) formulation as an additional definition of our model. The objective function, the sum of completion times, may of course be replaced by other standard objectives. We opt for a disjunctive representation, that is, a schedule is given by the sequence of jobs that is carried out on each machine (the x_{ij} variables). Compared to rank-based and time-indexed models, this approach is often seen as more efficient. Additionally to the set of jobs $\mathcal{J}$, we define a set $\mathcal{J}^0$ of m fictive jobs J_{0i}. For each $J_{0i} \in \mathcal{J}^0$, it holds true that $\delta_i^g = 0\ \forall g \in G$ and $p_i = 0$.

We assume that each job J_{0i} is processed on machine M_i at time *zero*.

$$\text{Minimize} \sum_{j \in \mathcal{J}} C_j \tag{5.4}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{J} \cup \mathcal{J}^0} x_{ij} = 1 \qquad \forall j \in \mathcal{J} \tag{5.5}$$

$$\sum_{i \in \mathcal{J}} x_{ji} \leq 1 \qquad \forall j \in \mathcal{J} \cup \mathcal{J}^0 \tag{5.6}$$

$$\sum_{i \in \mathcal{J} \cup \mathcal{J}^0} x_{ji} = 0 \qquad \forall j \in \mathcal{J}^0 \tag{5.7}$$

$$S_j + p_j = C_j \qquad \forall j \in \mathcal{J} \tag{5.8}$$

$$M \cdot (1 - x_{ij}) \geq C_i - S_j \qquad \forall i \in \mathcal{J},\ \forall j \in \mathcal{J} \tag{5.9}$$

$$S_j - S_i + \varepsilon \leq M \cdot y_{ij} \qquad \forall i \in \mathcal{J},\ \forall j \in \mathcal{J} \tag{5.10}$$

$$S_i - S_j - \varepsilon \leq M \cdot (1 - y_{ij}) \qquad \forall i \in \mathcal{J},\ \forall j \in \mathcal{J} \tag{5.11}$$

$$S_j - C_i + \varepsilon \leq M \cdot z_{ij} \qquad \forall i \in \mathcal{J},\ \forall j \in \mathcal{J} \tag{5.12}$$

$$C_i - S_j - \varepsilon \leq M \cdot (1 - z_{ij}) \qquad \forall i \in \mathcal{J},\ \forall j \in \mathcal{J} \tag{5.13}$$

$$\sum_{i \in \mathcal{J}^{g+}} z_{ij} \cdot \delta_i^g + \sum_{i \in \mathcal{J}^{g-}} y_{ij} \cdot \delta_i^g \geq 0 \qquad \forall g \in \mathcal{G},\ \forall j \in \mathcal{J} \tag{5.14}$$

S_j and C_j denote the starting and finishing dates of the jobs. The decision variables x_{ij} are used to indicate that job j immediately succeeds to job i, that is: both jobs are processed on the same machine and $C_i = S_j$. Additionally, in order to ensure that the cumulated material remains nonnegative (Constraint 5.14), the binary variables y_{ij} and z_{ij} are used. y_{ij} takes the value 1 exactly when job j starts processing no later than the *starting* time of job i, on any machine. In contrast, $z_{ij} = 1$ if and only if job j starts no later than the *finishing* time of job i. This distinction is needed as in our model, we consider that produced goods are added after the processing of a producer, while consumers remove the goods they require at the start of their processing. The correct setting of the y_{ij} and z_{ij} variables is guaranteed by Constraints 5.10 through 5.13. The ε-parameter is used to deal with the case where two consumers start exactly at the same time, or a consumer starts while a producer finishes. The value of ε is set to be smaller than the smallest unit of time; hence, it does not influence the inequality in case $S_j \neq S_i$ resp. $S_j \neq C_i$. In case of equality, however, it forces the y_{ij} and z_{ij} variables to take the value 1. The "Big-M" constant should be set at a value that is always larger than any subtraction of two starting or finishing dates; hence, $M = \sum_j p_j$ suffices here.

The size of the model doesn't seem excessively large; we introduced $\mathcal{O}(3n^2)$ variables and $\mathcal{O}(5n^2 + nk)$ constraints, where n denotes the cardinality of $\mathcal{J}$ and k the total amount of goods. Moreover, time indexing was avoided. Still, this model is tractable only for small instances in practice, mainly due to the impractical "Big-M" constraints. As it turns out, the theoretical complexity of this general formulation is high in most cases. In order to obtain more significant results, in the remaining part of this chapter, we consider problems with the following simplifications: it is assumed that $|\mathcal{M}| = |\mathcal{G}| = 1$. The previous MIP remains valid in this particular case, yet a simpler formulation can be derived on the basis of the model in Briskorn et al. [19].

$$\text{Minimize} \sum_i \sum_j x_{ij} C_{ij} \tag{5.15}$$

$$\text{s.t.} \quad x_{ij} + x_{ji} = 1 \qquad \forall i \in \mathcal{J},\ \forall j \in \mathcal{J},\ i \neq j \tag{5.16}$$

$$x_{ij} + x_{jk} + x_{ki} \leq 2 \qquad \forall (i,j,k) \in \mathcal{J}^3,\ i \neq j,\ j \neq k,\ k \neq i \tag{5.17}$$

$$\Delta_0^g + \sum_i \delta_i^g\, x_{ij} + \delta_j \geq 0 \qquad \forall j \in \mathcal{J},\ \forall g \in \mathcal{G} \tag{5.18}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in \mathcal{J},\ \forall j \in \mathcal{J} \tag{5.19}$$

In contrast to the multi-machine model, the decision variables x_{ij} indicate whether job j is scheduled after, but not necessarily directly after job i.

Furthermore, we introduce another simplification that may be applied to some instances. If in an instance $\mathcal{I}$ all producers are needed to schedule the consumers, i.e.

$$\sum_{J_j \in \mathcal{J} \setminus \{J_r\}} \delta_j^g < 0, \quad \forall J_r \in \mathcal{J} \tag{5.20}$$

we call $\mathcal{I}$ an instance that has the "no producer selection" property (NPS property).

Finally, as mentioned in Chapter 2, we use the usual three-field $\alpha \mid \beta \mid \gamma$-notation for scheduling problems (see e.g. [47]). For prefabrication or inventory constraints, we add the tokens "prfb" or "inv" respectively to the β-field. We write "prfb_1" when the amount of types of goods is restricted to one. In addition, the tokens "δ_1^+" and "δ_1^-" indicate that all producers or all consumers add or remove exactly 1 unit of resource, respectively, to or from the stack.

5.3 Related Research

Inventory constraints as defined above were described around the turn of the century by Laborie [62]. In his paper, the author studies scheduling problems with different types of cumulative and non-cumulative discrete resource constraints. His approach is extended to continuous resource consumption or production by Sourd and Rogerie [81]. Neumann and Schwindt [71], Neumann et al. [72] and Carlier et al. [25] consider inventory constraints for project scheduling problems. The authors minimize the total duration (makespan) of a project while keeping the inventory of a cumulative resource within specified bounds using a branch-and-bound algorithm.

Briskorn et al. [21], Boysen et al. [18] and Bazgosha et al. [8] apply inventory constraints to truck scheduling problems; machines represent warehouses, or transshipment stations, where vehicles, i.e. the jobs bring or pick up different types of goods. The authors provide complexity results [21] and develop heuristics [18, 8] and metaheuristics [8]. In their setting, material is added to or removed from the stack only by the starting or the completion of jobs. Moreover, all jobs, i.e. both producers and consumers, are taken into account in the objective function. Morsy and Pesch [68] compute approximated solutions for the truck scheduling problem while Briskorn and Leung [20] develop branch-and-bound algorithms.

Different variations were studied on the basis of [21]; Boysen et al. [17] and Drótos and Kis [38] consider the problem where all jobs produce material. Consumption of fixed amounts occurs at fixed times. Conversely, Györgyi and Kis [50, 51, 53] deal with instances where jobs only consume resources. Material is replenished at fixed times. Györgyi and Kis [52] show that some special cases of both variants can be reduced to a Knapsack problem.

To the best of our knowledge, problems with *prefabrication*, that is, inventory constraints where only consumers are taken into account in the objective, have not yet been addressed in literature. This seemingly small difference is motivated by applications in the manufacturing industry (see Introduction 5.1), and leads to a different mathematical problem. Consider, for instance, the following scheduling problems:

Example 2 (Problem 1 | $*$ | $C_{\max}$): Compute a schedule of minimum makespan in the single-machine case for jobs with either inventory or prefabrication constraints.

The problem described in Example 2 has a trivial solution when subject to inventory constraints in the sense of [21]: any schedule in which all producers are processed first, followed by all consumers (in any order), is feasible and induces no waiting time. Hence, it is optimal. In the case of prefabrication constraints, the same strategy leads to a feasible yet not necessarily optimal schedule. Indeed, less producers may suffice to produce the total amount that is required for the scheduling of all consumers. Dropping the superfluous producers would then reduce the total makespan. We show in Section 5.4 that the problem in Example 2 with prefabrication constraints is in fact $\mathcal{NP}$-hard.

Example 3 (Particular case of $1 \mid * \mid \sum_j C_j$): Assuming that:

$$\max_{J_j \in \mathcal{J}^+} p_j \leq \min_{J_j \in \mathcal{J}^-} p_j$$

Compute a schedule with minimum sum of completion times in the single-machine case.

The problem in Example 3 too admits a trivial solution when subject to inventory constraints. The schedule obtained by arranging the jobs in increasing processing times is obviously feasible and optimal. On the other hand, the same problem subject to prefabrication constraints can be shown to be $\mathcal{NP}$-hard (see Section 5.6).

Nonetheless, prefabrication and inventory constraints are related to each other in the case of weighted sum objective function. Prefabrication problems can then be seen as a particular case of inventory constrained problems where the weight of each producing job is set to zero.

5.4 Regular Objectives

First we discuss the influence of the NPS property on the complexity of the single-machine problem with prefabrication.

Let $\mathcal{H}^{\text{reg}}$ be the class of *regular* objectives, i.e. for each $h \in \mathcal{H}^{\text{reg}}$ it holds $h(\sigma) = h(C_{\sigma(1)}, \ldots, C_{\sigma(n)})$ and h is non-decreasing with respect to the completion times. Moreover, we consider sub-classes $U(\mathcal{H}^{\text{reg}}) \subset \mathcal{H}^{\text{reg}}$ with the following property:

Property 1: For each $V_1 \in \mathbb{N}$ there exists a regular objective $h \in U(\mathcal{H}^{\text{reg}})$ and $V_2 \in \mathbb{N}$ with $V_1 < V_2$ and $h(V_1) < h(V_2)$.

Examples for $U(\mathcal{H}^{\text{reg}})$ are $C_{\max}$, $\sum_j C_j$, $L_{\max}$ and $\sum_j U_j$.

Lemma 3: If the NPS property does not hold and $U(\mathcal{H}^{\text{reg}}) \subset \mathcal{H}^{\text{reg}}$ fulfills Property 1, $1 \mid \text{prfb}_1 \mid U(\mathcal{H}^{\text{reg}})$ is at least weakly $\mathcal{NP}$-hard.

Proof. We show weakly $\mathcal{NP}$-hardness via a reduction from the decision form of Knapsack:
Does there exists a subset of items $\mathcal{K} \subset \mathcal{A}$ such that $\sum_{a_i \in \mathcal{K}} v_i \geq V$ and $\sum_{a_i \in \mathcal{K}} w_i \leq W$?
Where:

- $a_i \in \mathcal{A}$, $i \in \mathbb{N}$, is a finite set of items,
- $w_i \in \mathbb{N}$ and $v_i \in \mathbb{N}$ are cost values and weights, respectively, for each $a_i \in \mathcal{A}$,
- $W, V \in \mathbb{N}$ are two fixed integers.

Let $\mathcal{D}^*$ be an instance of the decision form of $1 \mid \text{prfb}_1 \mid U(\mathcal{H}^{\text{reg}})$ with:

- set of producers $\mathcal{J}^+$: for each $a_i \in \mathcal{A}$ define J_i with $p_i = w_i$ and $\delta_i = v_i$,
- a single consumer J_c: $\delta_{J_c} = -V$, $p_{J_c} = W^* - W - 1$ and

$$W^* = \text{argmin}_{\{w \in \mathbb{N}: h^*(W) < h^*(w)\}} h^*(w)$$

  with $h^* \in U(\mathcal{H}^{\text{reg}})$ fulfilling Property 1.

$\mathcal{D}^*$ covers the decision form of Knapsack as the answer for this problem is "yes" if and only if there is a solution σ for $\mathcal{D}^*$ with $h^*(\sigma) = h^*(C_{J_c}) < h^*(W^*)$. Therefore, the decision form of $1 \mid \text{prfb}_1 \mid U(\mathcal{H}^{\text{reg}})$ is $\mathcal{NP}$-complete, and thus, $1 \mid \text{prfb}_1 \mid U(\mathcal{H}^{\text{reg}})$ at least weakly $\mathcal{NP}$-hard. □

Next, we analyse the structure of optimal solutions for the single-machine problem with prefabrication and regular objectives.

Lemma 4: Consider $1 \mid \text{prfb}_1 \mid h^{\text{reg}}$. Then there always exists an optimal non-delay schedule in which the first job j^* of each consumer block B^{c} has a higher consumption than the total production quantity of any proper subset of the jobs of the preceding producer block B^{p}:

$$\delta_{j^*} > \sum_{j \in B^{\text{p}} \setminus \{j'\}} \delta_j \quad \forall\, j' \in B^{\text{p}}, \tag{5.21}$$

Proof. Assume there is an optimal schedule $\mathcal{S}^*$ with a producer/consumer block pair $B^{\mathrm{p}}/B^{\mathrm{c}}$ that does not fulfil the condition. Then there exists a $j' \in B^{\mathrm{p}}$ with

$$\delta_{j^*} \leq \sum_{j \in B^{\mathrm{p}} \setminus \{j'\}} \delta_j \,.$$

and we can define a schedule $\mathcal{S}^{**}$ in which B^{p} and j^* are rearranged as follows:

$$j \in B^{\mathrm{p}} \setminus \{j'\} \;\rightarrow\; j^* \;\rightarrow\; j'.$$

The objective value of $\mathcal{S}^{**}$ is as least as good as the one of $\mathcal{S}^*$ since C_{j^*} is reduced by $p_{j'}$, the completion times of the remaining consumers remain unchanged and h^{reg} is regular. □

5.5 Lateness Related Objective Functions

In this section, we give the complexity classes of different problems with prefabrication for the L_{max} and $\sum_j U_j$ objective functions. All the results presented in this section can also be found in Table B.1, in the appendix of this thesis. Note that in this whole section, we only consider problem instance with exactly one machine and one type of goods.

5.5.1 Maximum Lateness

Theorem 20: The following algorithm optimally solves $1 \mid \mathrm{prfb}_1, \delta_1^+, \delta_1^- \mid L_{\max}$ in time $\mathcal{O}(|\mathcal{J}| \cdot \log(|\mathcal{J}|))$:

1. Sort the consumers by increasing due date,
2. sort the producers by increasing processing time,
3. starting with a producer, schedule producers and consumers alternatingly; when no consumers are left (no NPS), schedule the remaining producers at a stretch.

Proof. Briskorn et al. [21] show that the consumers must be arranged following the Earliest Due Date (EDD-) rule. Let us now show that it is both feasible and optimal to order the producers by shortest processing times.

Let $\mathcal{S}$ be a feasible and optimal $1 \mid \text{prfb}_1, \delta_1^+, \delta_1^- \mid L_{\max}$ schedule where all consumers follow the EDD rule. Let us introduce the following notation: $C_j(\mathcal{S})$ the completion time of job j in schedule $\mathcal{S}$. We assume $\mathcal{S}$ is such that:

$$\exists\{J_1, J_2 \in \mathcal{J}^+\}\colon p_1 < p_2 \text{ and } C_1(\mathcal{S}) > C_2(\mathcal{S})$$

A new schedule $\mathcal{S}^*$ obtained by exchanging the positions of jobs J_1 and J_2 in schedule $\mathcal{S}$ is also feasible since $\delta_{J_1} = \delta_{J_2}$. Since no idle time is induced by the exchange of J_1 with J_2, one can state that:

$$\forall J_k \in \mathcal{J},\ C_k(\mathcal{S}) < C_2(\mathcal{S}) \text{ or } C_k(\mathcal{S}) > C_1(\mathcal{S}) : C_k(\mathcal{S}) = C_k(\mathcal{S}^*)$$

In other words, all jobs that were not scheduled between J_1 and J_2 are unaffected by the exchange, and thus identically scheduled in $\mathcal{S}$ and $\mathcal{S}^*$.

As a result, $C_2(\mathcal{S}^*) = C_1(\mathcal{S})$. However, since $p_1 < p_2$ it holds that $C_1(\mathcal{S}^*) < C_2(\mathcal{S})$. This induces that the lateness of the earliest consumer scheduled after J_2 in $\mathcal{S}$ has been reduced, whereas the lateness of the earliest consumer scheduled after J_1 in $\mathcal{S}$ is identical. Any other latenesses are unaffected. Therefore, $\mathcal{S}^*$ preserves feasibility and optimality of $\mathcal{S}$, and repeating similar exchanging steps leads to a solution where consumers follow the EDD rule and producers follow the SPT-rule. □

Although the following theorem with a similar proof was presented in Briskorn et al. [21], for the sake of completeness, its analogue for prefabrication problems is shown here.

Theorem 21: Problem $1|\text{prfb}_1|L_{\max}$ is strongly $\mathcal{NP}$-hard, even if the NPS-condition holds, all producers have efficiency $\frac{\delta_j}{p_j} = 1$ and all consumers have unit processing time and the same consumption δ^*.

Proof. We prove strong $\mathcal{NP}$-hardness via a reduction from the well known strongly $\mathcal{NP}$-complete problem 3-PARTITION. Recall that 3-PARTITION is defined as follows: given a target number B and $3m$ numbers $a_1, a_2, \ldots, a_{3m}$, with

$$\sum_{j=1}^{3m} a_j = mB$$

and $\frac{B}{4} < a_j < \frac{B}{2}$ for all $1 \leq j \leq 3m$, find a partition of the index set $\mathcal{I} = 1, 2, \ldots, 3m$ into m sets $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_m$, such that for all $1 \leq k \leq m$

$$\sum_{j \in \mathcal{I}_k} a_j = B$$

Given an instance of 3-PARTITION as described above, we define an instance of $1|\text{prfb}_1|L_{\max}$ with exactly one product g in the following way. First, for each number a_j of the 3-PARTITION instance, we have one producing job j with $p_j = \delta_j = a_j$. Furthermore, we add n additional consuming jobs $V_1, V_2, \ldots, V_m$ all with processing time $p_{V_k} = 1$ and consumption $\delta_{V_k} = \delta^* = -B$. Their due-dates are given by $d_{V_k} = kB + k$, for $1 \leq k \leq m$.

We show that there is a solution to the instance of 3-PARTITION if and only if there is a solution to the scheduling instance with maximum lateness $L_{\max} \leq 0$.

It is easy to see that a solution to the 3-PARTITION instance implies a solution to the scheduling instance. Indeed, we can obtain a schedule with $L_{\max} \leq 0$ by scheduling first all producing jobs j with $j \in \mathcal{I}_k$ (where $\mathcal{I}_k$ is given by the solution of 3-PARTITION), followed by the consuming job V_k. The final schedule is of the form

$$\mathcal{I}_1, V_1, \mathcal{I}_2, V_2, \mathcal{I}_3, V_3, \ldots, \mathcal{I}_m, V_m$$

where block $\mathcal{I}_k$ contains the producing jobs j with $j \in \mathcal{I}_k$, scheduled in any order. Clearly, the schedule is feasible, since in each block $\mathcal{I}_k$ a quantity of exactly B of product g is produced, which is then consumed fully by job V_k. Furthermore, the schedule has a maximum lateness of 0, since each block $\mathcal{I}_k$ takes exactly B time to process, followed by job V_k of length 1. Thus, job V_k finishes at time $kB + k = d_{V_k}$.

For the other direction, assume there is given a solution for the scheduling instance with maximum lateness $L_{\max} \leq 0$. Observe first that job V_1 cannot start before time B, since all producing jobs have efficiency 1 and it is not possible to produce the necessary quantity B of product g earlier than this. However, job V_1 also cannot be started later than time B as it would otherwise violate its own due-date. Thus, job V_1 starts exactly at time B.

Note that V_1 then consumes all of product g and thus job V_2 cannot be started before time $2B + 1$ (job V_1 finishes at time $B + 1$ and then it takes B time to produce the necessary quantity of product g). Again, job V_2 cannot be started any later than time $2B + 1$ as otherwise it would violate its own due-date. Thus, in the given solution to the scheduling instance, job V_2 starts exactly at time $2B + 1$. Continuing this argument inductively, we see that in the given solution job V_k has to start exactly at time $kB + (k - 1)$.

Therefore, the consuming jobs $V_1, V_2, \ldots, V_m$ split the given solution to the scheduling problem into m blocks, leaving gaps of size exactly B, into which the producing jobs have to be fit. Each gap of size B has to be filled completely, as otherwise not enough of product g would be produced and the given solution

would be infeasible. Denoting the set of producing jobs scheduled in the k-th gap by $\mathcal{I}_k$, this means that

$$\sum_{j \in \mathcal{I}_k} p_j = \sum_{j \in \mathcal{I}_k} \delta_j = \sum_{j \in \mathcal{I}_k} a_j = B$$

and therefore the index sets $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_m$ correspond to a solution of the original instance of 3-PARTITION. □

5.5.2 Number of Tardy Jobs

First, we may notice that some of the results of Subsection 5.5.1 also apply to the number of tardy jobs.

Corollary 4: It follows directly from Theorem 21 that any problem $1|\text{prfb}_1|\sum_j U_j$ is strongly $\mathcal{NP}$-hard, even if the NPS-condition holds, all producers have efficiency $\frac{\delta_j}{p_j} = 1$ and all consumers have unit processing time and the same consumption δ^*. □

The following observation, however, allows us to derive polynomial algorithms for other problem instances.

Observation 7 (Moore's algorithm): Moore's due date algorithm for the $1||\sum_j U_j$ problem (see Moore [67]) can be adapted to scheduling problems with prefabrication. We order the consumers following the EDD-rule. Next, we move some of these consumers to the end of the schedule, if late jobs cannot be prevented. Whenever we encounter the first late consumer, we move the consumer which most contributes to that lateness to the end of the schedule. We repeat this second step, until all consumers not moved to the end of the schedule in this way are on time. However, in order to apply this algorithm, two strategies must be defined:

- how to order the producing jobs
- how to chose a consumer to be moved

Observation 8 (Arranging producing jobs): Strategies for the producers can be identified by making the following observations:

- When δ_1^+ applies, then producers are arranged in SPT-order. If in any schedule σ there exist two jobs J_a and J_b such that $S_a(\sigma) < S_b(\sigma)$ while $p_a > p_b$, then swapping these jobs would result in a schedule σ' that is no worse than σ regarding the number of tardy jobs.

Problem $1\|\text{prfb}_1, *\|\sum U_j$	Ordering of producers	Strategy for consumers
$\delta_1^+, \quad \delta_1^-$	SPT-order	move largest p_j
$\delta_1^+, \qquad p_j^+ = \underline{p}$	SPT-order	move largest $p_j + \lvert\delta_j\rvert\,\underline{p}$
$\delta_1^+, \qquad\qquad p_j^- = \underline{p}$	SPT-order	move largest $\lvert\delta_j\rvert$
$\delta_1^-, \quad p_j^+ = \underline{p}$	non-increasing δ_j	move largest p_j
$p_j^+ = \underline{p}_1, \quad p_j^- = \underline{p}_2$	non-increasing δ_j	move largest $\lvert\delta_j\rvert$

Table 5.1: Strategies for the ordering of producers and the moving of consumers when applying Moore's algorithm

- similarly, if $p_j^+ = \underline{p}$ applies, then the producers are sorted by non-increasing δ_j.

The orderings defined in Observation 8 are presented in Table 5.1 for different instances of prefabrication problems. The table also shows which strategy to apply for the consumers. The properties that are given in the third column correspond to the consumer that should be moved to the end of the schedule.

Moore's algorithm combined with the strategies defined in Table 5.1 can be shown to compute optimal solutions for the prefabrication problems listed in the table. An overview of the complexity results for the weighted sum of completion times objective is given by Figure 5.2.

5.6 Total Completion Time Related Objective Functions

In this section, we give the complexity classes of different problems with prefabrication for the $\sum_j C_j$ and $\sum_j w_j C_j$ objective functions. While the sum of weighted completion times is the more general setting, it is also harder to obtain meaningful results for this particular objective. All the results presented in this section can also be found in Table B.2, in the appendix of this thesis. Note that in this whole section, we only consider problem instance with exactly one machine and one type of goods.

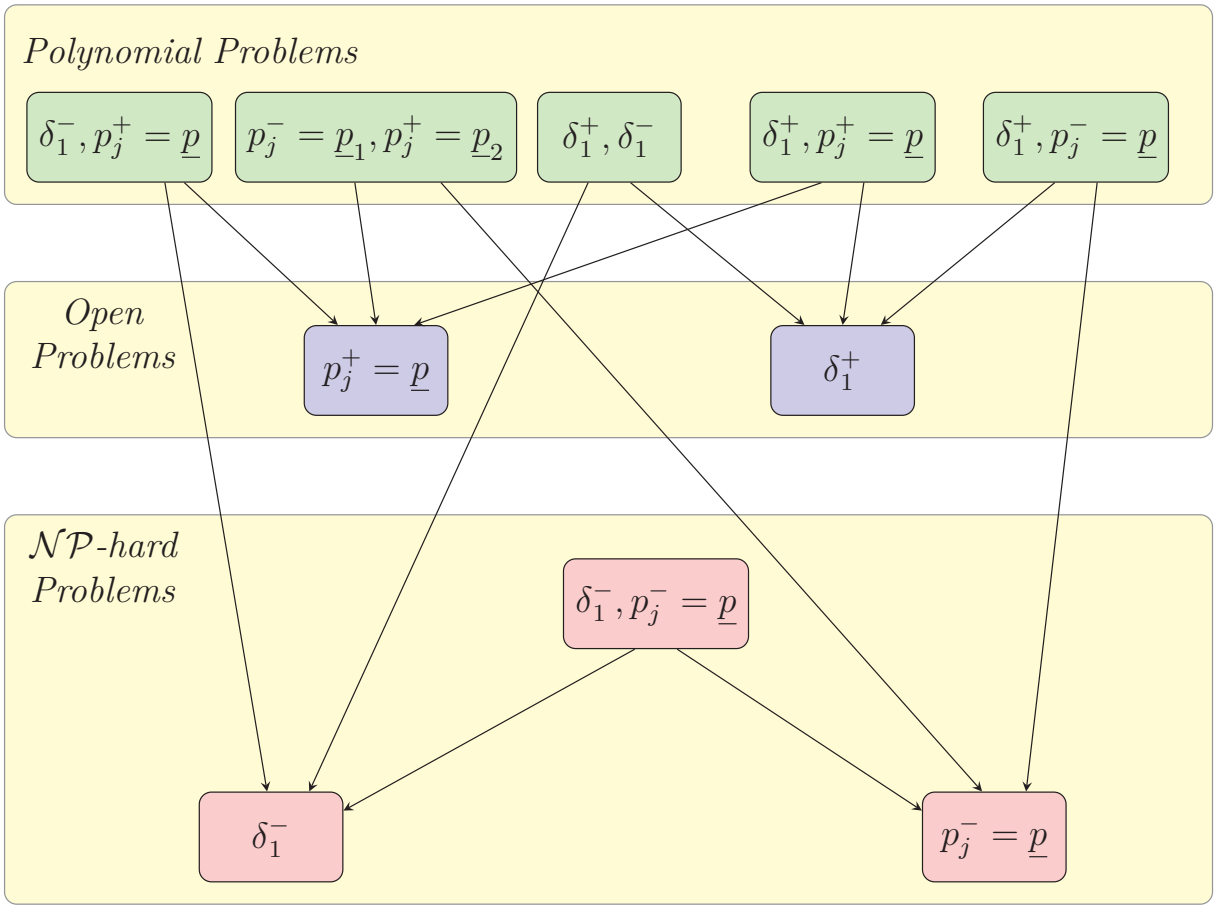


Figure 5.1: Complexity classes of the $1 \mid \text{prfb}_1, * \mid \sum_j U_j$ prefabrication problems

5.6.1 Sum of Completion Times

Theorem 22: Problem $1|\text{prfb}_1|\sum_j C_j$ is strongly $\mathcal{NP}$-hard, even if the NPS-condition holds, all producers have efficiency $\frac{\delta_j}{p_j} = 1$ and all consumers have unit processing time and the same consumption $\delta^* < 0$.

It remains strongly $\mathcal{NP}$-hard even if instead all consumers have the same processing time p^- and $p^- > p_j$ for any $J_j \in \mathcal{J}^+$.

Proof. We show strong $\mathcal{NP}$-hardness via a reduction from 3-PARTITION. The construction has some similarity with the proof of strong $\mathcal{NP}$-hardness of minimizing the maximum lateness in [21].

Recall that 3-PARTITION is defined as follows: given a target number B and $n = 3m$ numbers $a_1, a_2, \ldots, a_{3m}$, with

$$\sum_{j=1}^{3m} a_j = mB$$

and $\frac{B}{4} < a_j < \frac{B}{2}$ for all $1 \leq j \leq 3m$, find a partition of the index set $\mathcal{I} = 1, 2, \ldots, 3m$ into m sets $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_m$, such that for all $1 \leq k \leq m$

$$\sum_{j \in \mathcal{I}_k} a_j = B$$

Given an instance of 3-PARTITION, we define an instance of $1|\text{prfb}_1|\sum_j C_j$ with exactly one product g in the following way. For each number a_j of the 3-PARTITION instance, we have one producing job J_j with $p_j = \delta_j = a_j$. Furthermore, we add m additional consuming jobs $V_1, V_2, \ldots, V_m$ all with processing time $p_{V_k} = 1$ and consumption $\delta_{V_k} = \delta^* = B$

We show that there is a solution to the instance of 3-PARTITION, if and only if there is a solution to the instance of $1|\text{prfb}_1|\sum_j C_j$ with

$$\sum_j C_j \leq \frac{m(m+1)}{2}(B+1)$$

Suppose first there is a solution to the instance of 3-PARTITION. Then we can obtain a schedule with $\sum_j C_j \leq mB + \frac{m(m+1)}{2}$ by scheduling in an alternating fashion first all producing jobs j with $j \in \mathcal{I}_k$ (where $\mathcal{I}_k$ is given by the solution of 3-PARTITION), followed by the consuming job V_k. The final schedule is of the form

$$\mathcal{I}_1, V_1, \mathcal{I}_2, V_2, \mathcal{I}_3, V_3, \ldots, \mathcal{I}_m, V_m$$

where block $\mathcal{I}_k$ contains the producing jobs J_j with $j \in \mathcal{I}_k$, scheduled in any order. Clearly, the schedule is feasible, since in any block $\mathcal{I}_k$ a quantity of exactly B of product g is produced, which is then fully consumed by job V_k. Furthermore, note that job V_k finishes after exactly $kB + k = k(B+1)$ time, leading to a total sum of completion times of

$$\sum_j C_j = \sum_{k=1}^{m} k(B+1) = (B+1)\sum_{k=1}^{m} k = \frac{m(m+1)}{2}(B+1)$$

For the opposite direction, note that the order in which we schedule the consumers $V_1, V_2, \ldots, V_m$ in a feasible schedule is not important: the positions of two consumers can be swapped arbitrarily without changing the feasibility or the the objective value of a given schedule. Thus from now on we assume that the consumers are scheduled in order of their numbering.

Then, the earliest time at which consumer V_k can be scheduled is $kB + k - 1$. Indeed, it takes at least kB time to produce kB units of product g, as B units are needed for V_k and each consumer $V_1, V_2, \ldots, V_{k-1}$ scheduled before it, and additionally $k - 1$ time to schedule consumers $V_1, V_2, \ldots, V_{k-1}$ themselves. Thus,

$$\sum_{k=1}^{m} k(B+1) = (B+1)\sum_{k=1}^{m} k = \frac{m(m+1)}{2}(B+1)$$

is a lower bound for the total completion time of a feasible schedule. Moreover, the lower bound can only be reached if each consumer V_k is scheduled exactly at time $kB + k - 1$. In such a schedule, note that after each consumer V_k is scheduled, the available amount of product g is reduced to 0.

Therefore, the consuming jobs $V_1, V_2, \ldots, V_m$ split the given solution to the scheduling problem into m blocks, leaving gaps of size exactly B, into which the producing jobs have to be fit. Each gap of size B has to be filled completely, as otherwise not enough of product g would be produced and the given solution would be infeasible. Denoting the set of producing jobs scheduled in the k-th gap by $\mathcal{I}_k$, this means that

$$\sum_{j \in \mathcal{I}_k} p_j = \sum_{j \in \mathcal{I}_k} \delta_j = \sum_{j \in \mathcal{I}_k} a_j = B$$

and therefore the index sets $\mathcal{I}_1, \mathcal{I}_2, \ldots, \mathcal{I}_m$ correspond to a solution of the original instance of 3-PARTITION.

The second statement can be proven using an analogous reduction, but with consumers having common processing time B instead of 1. □

Observe that our result implies that for the problem considered in [21] it is strongly $\mathcal{NP}$-hard to minimize the total weighted completion time, even if all producers have the same efficiency and zero weight and all consumers are identical, with unit weights and processing times.

Furthermore, as observed previously in Section 5.3, minimizing the total completion time when all producers have smaller processing time then any consumer

remains strongly $\mathcal{NP}$-hard for our problem (due to the second statement of Theorem 22), while it can be solved by scheduling jobs in order of their processing times for the problem studied in [21].

Lemma 5: Any $1 \mid \text{prfb}_1, \delta_1^- \mid \sum_j C_j$ optimization problem has an optimal solution such that all consumers are scheduled following the Shortest Processing Time (SPT-) rule.

Proof. Assume there is an optimal solution σ in which the consumers are not sorted according to the SPT-rule. Then there exists a pair of consumers J_{j^*} and $J_{j^{**}}$ in σ, separated at most by one producer $J_{j'}$ (with processing time $p_{j'}$) and no other consumer, with $p_{j^*} > p_{j^{**}}$. Exchanging J_{j^*} and $J_{j^{**}}$ leads to a solution σ^* which is feasible as well. In σ^* the completion times of all consumers but J_{j^*} and $J_{j^{**}}$ remain unchanged. For J_{j^*} and $J_{j^{**}}$ it holds:

$$\begin{aligned} C^*_{j^*} &= C_{j^*} + p_{j'} + p_{j^{**}} \\ C^*_{j^{**}} &= C_{j^{**}} - p_{j'} - p_{j^*} \\ \Rightarrow\ C^*_{j^*} + C^*_{j^{**}} &= C_{j^*} + C_{j^{**}} + \underbrace{p_{j^{**}} - p_{j^*}}_{<0} \end{aligned}$$

□

Unfortunately, as the following example illustrates, this is does not hold true for the $1 \mid \text{prfb}_1 \mid \sum_j C_j$ problem.

Example 4 (Counterexample): *Producing jobs:*

	Length	Amount
P_1	2	2
P_2	3	3

Consuming jobs:

	Length	Amount
C_1	1	1
C_2	2	3
C_3	3	1

Then the best solution respecting the SPT-rule is

$$P_1 \to C_1 \to P_2 \to C_2 \to C_3$$

with objective value 22. However, the optimal solution is

$$P_1 \to C_1 \to C_3 \to P_2 \to C_2$$

with objective value 20.

Theorem 23: Any $1 \mid \mathrm{prfb}_1, \delta_1^- \mid \sum_j C_j$ optimization problem has an optimal solution such that all consumers are scheduled following the SPT-rule and all producers are scheduled with non-increasing efficiency δ_j / p_j. The complexity of this solution approach is that of a sorting algorithm, namely, $\mathcal{O}(n \log n)$.

Proof. Assume there exists an optimal schedule σ with consumers sorted according to the SPT-rule and in which the producers are not sorted with non-increasing efficiency. Then there are two producers J_{j^*} and $J_{j^{**}}$ with

$$\frac{\delta_{j^*}}{p_{j^*}} < \frac{\delta_{j^{**}}}{p_{j^{**}}},$$

and J_{j^*} is scheduled before $J_{j^{**}}$ in σ and no other producers are scheduled between them. Since $p_{j^*}, p_{j^{**}} > 0$ it holds:

$$\delta_{j^*} \cdot p_{j^{**}} - \delta_{j^{**}} \cdot p_{j^*} < 0. \tag{5.22}$$

We consider two cases:

- $\delta_{j^{**}} \geq \delta_{j^*}$:

  Let σ' be a schedule that differs from σ as follows:

  - Schedule $J_{j^{**}}$ in the position of J_{j^*}.
  - After $J_{j^{**}}$ schedule the δ_{j^*} consumers that are assigned to J_{j^*} in σ (in the same order as in σ).
  - Then schedule the next $\delta_{j^{**}} - \delta_{j^*}$ which are assigned to $J_{j^{**}}$ in σ (in the same order as in σ).
  - Then schedule J_{j^*}.

All other jobs are scheduled as in σ. Note that the completion times of these jobs remain unchanged.

The total completion time of the consumers assigned to J_{j^*} in σ is changed by $\delta_{j^*} \cdot (p_{j^{**}} - p_{j^*})$ whereas the total completion time of the $\delta_{j^{**}} - \delta_{j^*}$ subsequent jobs is changed by $p_{j^*} \cdot (\delta_{j^*} - \delta_{j^{**}})$. Therefore, the total completion times of σ and σ' differ by

$$\delta_{j^*} \cdot (p_{j^{**}} - p_{j^*}) + p_{j^*} \cdot (\delta_{j^*} - \delta_{j^{**}}) = \delta_{j^*} \cdot p_{j^{**}} - \delta_{j^{**}} \cdot p_{j^*} < 0.$$

- $\delta_{j^{**}} < \delta_{j^*}$:

  Let σ'' be a schedule that differs from σ as follows:

  - Schedule $J_{j^{**}}$ in the position of J_{j^*}.
  - After $J_{j^{**}}$ schedule the first $\delta_{j^{**}}$ consumers that are assigned to J_{j^*} in σ (in the same order as in σ).
  - Then schedule J_{j^*}.
  - Then schedule the remaining $\delta_{j^*} - \delta_{j^{**}}$ which are assigned to J_{j^*} in σ (in the same order as in σ).

  All other jobs are scheduled as in σ. Note that the completion times of these jobs remain unchanged.

  The total completion time of the consumers assigned to $J_{j^{**}}$ in σ'' is changed by $\delta_{j^{**}} \cdot (p_{j^{**}} - p_{j^*})$ whereas the total completion time of the $\delta_{j^*} - \delta_{j^{**}}$ subsequent jobs is changed by $p_{j^{**}} \cdot (\delta_{j^*} - \delta_{j^{**}})$. Therefore, the total completion times of σ and σ'' differ by

$$\delta_{j^{**}} \cdot (p_{j^{**}} - p_{j^*}) + p_{j^{**}} \cdot (\delta_{j^*} - \delta_{j^{**}}) = \delta_{j^*} \cdot p_{j^{**}} - \delta_{j^{**}} \cdot p_{j^*} < 0.$$

As a result, in both cases schedule σ cannot be optimal which contradicts our assumption. □

Theorem 24: Problem $1|\text{prfb}_1, p_j^+ = \underline{p}_1, p_j^- = \underline{p}_2|\sum_j C_j$ can be solved to optimality by ordering both producers and consumers in non-increasing δ_j order.

Proof. Assume that there is an optimal solution in which consumer j is scheduled before consumer i but $\delta_i > \delta_j$. Then exchanging these two jobs leads to a feasible schedule as well (because of $\delta_i > \delta_j$, at the beginning of job i (old schedule) there is enough quantity of the product to execute job i and job j). This schedule has the same objective as the old one as $p_j^+ = \underline{p}_2$. However, it maybe in addition be possible to schedule job i before one or more preceding producers since $\delta_i > \delta_j$. □

5.6.2 Sum of Weighted Completion Times

Theorem 25: For $1 \mid \text{prfb}_1, \delta_1^+, p_j^+ = \underline{p} \mid \sum w_j C_j$ an optimal solution can be computed in $\mathcal{O}(n \log n)$ time. We achieve this by ordering the consumers in decreasing order of $w_j/(p_j + |\delta_j| \cdot \underline{p})$.

Proof. The producing jobs are identical, hence a solution is solely determined by a sequence of consumers.

Since each producer has its produced quantity $\delta_j = 1$, the time cost of a consumer $J_j \in \mathcal{J}^-$ is given explicitly; namely the sum of its own processing time p_j and the amount of time taken to produce its consumed resources, that is, $-\delta_j \cdot \underline{p}$.

Hence, if we replace the processing time of each consumer $J_j \in \mathcal{J}^-$ by $p_j - \delta_j \cdot \underline{p}$, the problem $1 \mid \text{prfb}_1, \delta_1^+, p_j^+ = \underline{p}_j \mid \sum w_j C_j$ reduces to the standard $1 \mid\mid \sum w_j C_j$ scheduling problem. The latter can be solved (by Smith's rule, see [80]) in $\mathcal{O}(n \log n)$ time. □

Corollary 5: It follows directly from Theorem 22 that both prefabrication problems $1 \mid \text{prfb}_1 \mid \sum w_j C_j$ and $1 \mid \text{prfb}_1, p_j^- = \underline{p} \mid \sum w_j C_j$ are $\mathcal{NP}$-hard. □

An overview of the complexity results for the weighted sum of completion times objective is given by Figure 5.2.

5.7 Conclusions and Outlook

In this chapter, we describe a new class of scheduling problems which we refer to as problems with prefabrication. These are motivated by applications from the manufacturing industry where products cannot be produced directly;

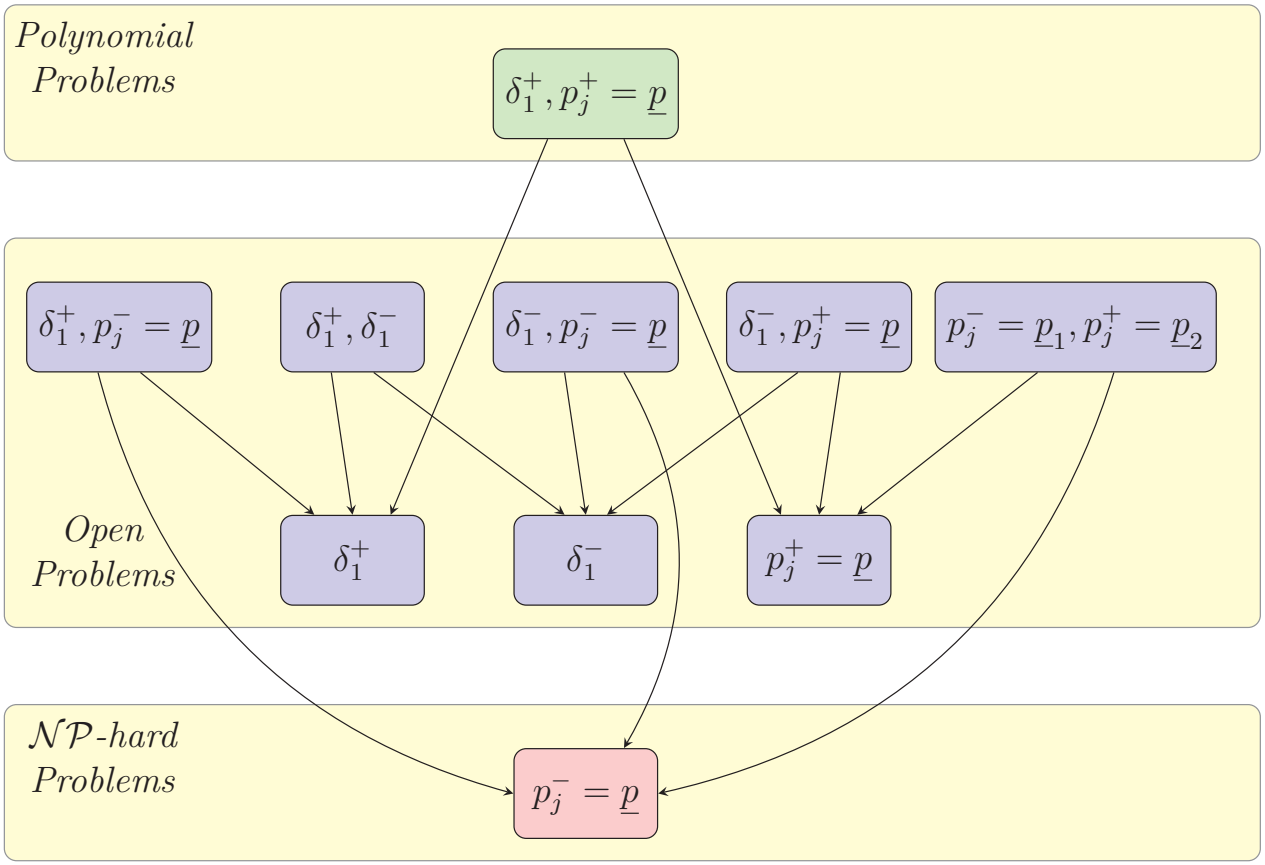


Figure 5.2: Complexity classes of the $1 \mid \text{prfb}_1, * \mid \sum_j w_j C_j$ optimization problems

instead, they require other incomplete sub-products to be manufactured first. These sub-products have no other finality than to be serve as a raw material for the final product. Therefore, in our setting, we distinguish producing and consuming jobs. Only the consumers, which consume the intermediate material, but output the final product, are taken into account in the objective function.

We show that when the number of types of goods, or the number of machines, is unrestricted, then the problem is generally hard to solve. We therefore focus on the single-machine single-material case. Furthermore, we show that problems with prefabrication are mostly hard as well when it is not assumed that the sum of the goods produced by all producers equals the sum of the goods consumed by all consumers (NPS-condition). Hence, we further restrict our study to problems where the NPS-condition holds.

We determine the complexity classes for different instances of problems with prefabrication, particularly with the maximum lateness, number of tardy jobs

and (weighted) sum of completion times objective functions. We consider the particular cases where jobs have identical processing times, as well as cases where the produced or consumed amounts are equal to 1. Our study suggest that it is easier to obtain results for instances with lateness-related objectives. Moreover, polynomial algorithms exist for a few cases. On the other hand, it seems much harder to obtain complexity results on total completion time-related objectives. Globally, even though our study focuses on very simplified and restrictive problems, scheduling with prefabrication proves to be hard. Polynomial algorithms could be found for very few problems with the weighted sum of completion times, for instance. Interestingly, for any type of objective function, restricting the producing jobs seems to be more likely to result in an easier problem than restricting the consumers (see Appendix B).

Since it is clear that complexities are often $\mathcal{NP}$-hard, we suggest that further research be directed towards solution approaches. Scheduling problems with inventory constraints are generally well-solved by constraint programming approaches. Nonetheless, efficient approximations or heuristics could be derived for instances with prefabrication. In addition, as soon as efficient solution approaches are found, the very restrictive setting of our study should be generalized. Instances with more than one type of goods are more realistic, in particular instances where the goods form a chain; each job consumes (exactly one) type of good g while producing type $g + 1$.

Part IV

Closing

Chapter 6

Conclusions and Further Research

In this thesis, we introduced new models for particular production planning problems, and derived algorithms to solve them.

In particular, we consider the buffer allocation problem (BAP) in a flow line with $\mathcal{S}$ stations and W workpieces. In Chapter 3, the aim is to compute an exact solution for a known sample of processing times $p \in \mathbb{R}^{\mathcal{S}\times W}$ that would also be robust against uncertainty. While some authors [88] did achieve a form of robustness by solving the BAP for a large amount of samples, to the best of our knowledge, robust solutions in the sense of Soyster [82] had not yet been computed before. We chose and justified the concept of Γ-robustness; it limits over-conservativeness and allows the user to chose the required level of robustness. We solved the problem in two different ways: by a generative-evaluative method, where the evaluation is reduced to a longest path problem in a graph, as well as by a direct method, obtained by dualizing the longest path formulation. It is observed that the decomposed flux method performs better when the level and amplitude of the uncertainty are small, while the direct dual method becomes more competitive when the uncertainty is greater.

We believe that our methodology yields potential for applications in the steel industry. Höhn [56], Frasch [40] and Frasch et al. [41] describe a flowshop scheduling environment, similar to that of the BAP, yet with a different objective function. In their setting, the amount of interruptions in the processing of the last station should be minimized. The amount of these *strand interruptions* is commonly small, yet each of them incurs a great cost. Hence, a robust formulation for this problem, following our approach for the BAP, seems particularly suitable. Besides alternative objective functions, we believe that future research should be conducted towards different job characteristics; for instance, when the overtaking of workpieces is permitted. Also, the problem

could be extended to more complicated flow line structures, for instance, with parallel machines at some stages.

However, the numerical experiments conducted in Chapter 3 showed that computational tractability is a major concern of our approach for the BAP. Therefore, a few ways to reduce the graph size in the decomposed approach were described. Furthermore, Chapter 4 addresses the problem of computation speed by investigating better cuts and stronger bounds on the objective function for the decomposed method without uncertainty. It is shown that the bounds and cuts that were previously used in literature cannot be applied to the case where a warm-up phase is carried out. Therefore, we defined a new measure of performance for buffer allocations, TH^+, that permits the use of those bounds and cuts in the warm-up case. Moreover, we derived a new algorithm for the nominal BAP that is based on global bounds for the TH^+-measure. This new method seems to perform better than the classical decomposed algorithm for most of the tested problem instances. Its efficiency mainly depends on two factors: the heuristic method that is chosen and the performance measure that is employed.

Therefore, this new algorithm directly provides two axes in which future research could be directed; computing fast heuristic solutions for the BAP, and finding "good" performance measures. This would further speed up nominal and robust buffer allocation. We understand the word "good" in the sense that a measure m should be monotonously increasing with the total sum of buffers, be greater than the classical throughput TH while minimizing the distance $m - \mathrm{TH}$. Heuristic solutions serve as a starting point for the iterative procedure. Due to the combinatorial nature of the BAP, a slightly overestimated initial heuristic solution increases the amount of iterations in a disproportioned way; hence, a good heuristic potentially yields great time savings.

Another problem setting that arises from manufacturing industry is that of inventory constrained scheduling. It was addressed in Part III of this thesis, the aim being to determine the complexity classes of different settings. Here, we considered a classical single-stage scheduling problem with one machine. The jobs are divided into a set $\mathcal{J}^+$ of producers that generate $\delta_j > 0$ units of material, and a set $\mathcal{J}^-$ of consumers that produce $\delta_j < 0$ units of the same material. While each consumer J_j is given a due date d_j and a completion time C_j, producers are not taken into account in the objective function. We motivated our model with applications from the production industry, and showed that it results in a different problem than what can be found in literature. This new setting was shown to be $\mathcal{NP}$-hard in most cases, if not further restricted. We

proved, however, that when the total produced amount over all jobs equals zero, then polynomial algorithms can be derived for different cases. We called this restriction the "no producer selection", or NPS-condition. We considered the weighted sum of tardy jobs $\sum_j w_j U_j$ and the weighted sum of completion times $\sum_j w_j C_j$ as objective functions, and reviewed the complexity class of the resulting scheduling problems for different job characteristics.

It results from our classification that this new type of problems is generally hard. Even when restricted to one type of resource, to one machine and when subject to the NPS-condition, many problem settings remain $\mathcal{NP}$-hard. Our listing is not exhaustive, and further complexity results could be investigated. Nonetheless, we believe that future efforts should rather be directed towards solving approaches. We have shown that some problem instances can be reduced to a knapsack problem, for which efficient, even if not polynomial algorithms are known to exist. For the remaining hard problems, it might be possible to derive approximations or heuristics.

Bibliography

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The design and analysis of computer algorithms*. Addison-Wesley Publishing Company Inc., Reading, Massachusetts, 1974.

[2] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. PHI Englewood Cliffs NJ, 1993.

[3] R. K. Ahuja et al. "Faster algorithms for the shortest path problem". In: *Journal of the ACM (JACM)* 37.2 (1990), pp. 213–223.

[4] J. M. Alden et al. "General Motors increases its production throughput". In: *Interfaces* 36.1 (2006), pp. 6–25.

[5] A. Alfieri and A. Matta. "Mathematical programming formulations for approximate simulation of multistage production systems". In: *European Journal of Operational Research* 219.3 (2012), pp. 773–783.

[6] A. Alfieri, A. Matta, and E. Pastore. "A column generation algorithm for the Buffer Allocation Problem approximated by the Time Buffer concept". In: *IFAC-PapersOnLine* 49.12 (2016), pp. 739–744.

[7] M. Amiri and A. Mohtashami. "Buffer allocation in unreliable production lines based on design of experiments, simulation, and genetic algorithm". In: *The International Journal of Advanced Manufacturing Technology* 62.1-4 (2012), pp. 371–383.

[8] A. Bazgosha, M. Ranjbar, and N. Jamili. "Scheduling of loading and unloading operations in a multi stations transshipment terminal with release date and inventory constraints". In: *Computers & Industrial Engineering* 106 (2017), pp. 20–31.

[9] R. Bellman. "On a routing problem". In: *Quarterly of Applied Mathematics* 16.1 (1958), pp. 87–90.

[10] A. Ben-Tal, L. El Ghaoui, and A. Nemirovski. *Robust optimization*. Vol. 28. Princeton University Press, 2009.

[11] A. Ben-Tal and A. Nemirovski. "Robust solutions of uncertain linear programs". In: *Operations Research Letters* 25.1 (1999), pp. 1–13.

[12] A. Ben-Tal and A. Nemirovski. "Robust solutions of linear programming problems contaminated with uncertain data". In: *Mathematical Programming* 88.3 (2000), pp. 411–424.

[13] J. F. Benders. "Partitioning procedures for solving mixed-variables programming problems". In: *Numerische mathematik* 4.1 (1962), pp. 238–252.

[14] D. Bertsimas and M. Sim. "Robust discrete optimization and network flows". In: *Mathematical Programming* 98.1-3 (2003), pp. 49–71. ISSN: 0025-5610. DOI: 10.1007/s10107-003-0396-4.

[15] D. Bertsimas and M. Sim. "The price of robustness". In: *Operations Research* 52.1 (2004), pp. 35–53.

[16] H.-G. Beyer and B. Sendhoff. "Robust optimization–a comprehensive survey". In: *Computer methods in applied mechanics and engineering* 196.33-34 (2007), pp. 3190–3218.

[17] N. Boysen, S. Bock, and M. Fliedner. "Scheduling of inventory releasing jobs to satisfy time-varying demand: an analysis of complexity". In: *Journal of Scheduling* 16.2 (2013), pp. 185–198.

[18] N. Boysen, D. Briskorn, and M. Tschöke. "Truck scheduling in cross-docking terminals with fixed outbound departures". In: *OR Spectrum* 35.2 (2013), pp. 479–504.

[19] D. Briskorn, F. Jaehn, and E. Pesch. "Exact algorithms for inventory constrained scheduling on a single machine". In: *Journal of scheduling* 16.1 (2013), pp. 105–115.

[20] D. Briskorn and J. Leung. "Branch and bound algorithms for minimizing maximum lateness of trucks at a transshipment terminal". In: *Optimization Online* 2677 (2010).

[21] D. Briskorn et al. "Complexity of single machine scheduling subject to nonnegative inventory constraints". In: *European Journal of Operational Research* 207.2 (2010), pp. 605–619.

[22] P. Brucker. *Scheduling algorithms*. Springer, 2007.

[23] M. H. Burman. "New results in flow line analysis". PhD thesis. Massachusetts Institute of Technology, 1995.

[24] M. Burman, S. B. Gershwin, and C. Suyematsu. "Hewlett-Packard uses operations research to improve the design of a printer production line". In: *Interfaces* 28.1 (1998), pp. 24–36.

[25] J. Carlier, A. Moukrim, and H. Xu. "The project scheduling problem with production and consumption of resources: A list-scheduling based algorithm". In: *Discrete Applied Mathematics* 157.17 (2009), pp. 3631–3642.

[26] W. K. Chan et al. "Properties of discrete event systems from their mathematical programming representations". In: *Simulation Conference, 2003. Proceedings of the 2003 Winter*. Vol. 1. IEEE. 2003, pp. 496–502.

[27] W. K. Chan and L. Schruben. "Optimization models of discrete-event system dynamics". In: *Operations Research* 56.5 (2008), pp. 1218–1237.

[28] G. Codato and M. Fischetti. "Combinatorial Benders' Cuts for Mixed-Integer Linear Programming". In: *Operations Research* 54.4 (2006), pp. 756–766. ISSN: 0030-364X. DOI: 10.1287/opre.1060.0286.

[29] M. Colledani et al. "Analytical methods to support continuous improvements at Scania". In: *International Journal of Production Research* 48.7 (2010), pp. 1913–1945.

[30] R. Conway et al. "The role of work-in-process inventory in serial production lines". In: *Operations Research* 36.2 (1988), pp. 229–241. ISSN: 0030-364X. DOI: 10.1287/opre.36.2.229.

[31] A. Costa et al. "A parallel tabu search for solving the primal buffer allocation problem in serial production systems". In: *Computers & Operations Research* 64 (2015), pp. 97–112.

[32] Y. Dallery, R. David, and X.-L. Xie. "Approximate analysis of transfer lines with unreliable machines and finite buffers". In: *IEEE Transactions on Automatic Control* 34.9 (1989), pp. 943–953.

[33] Y. Dallery and S. B. Gershwin. "Manufacturing flow line systems: a review of models and analytical results". In: *Queueing systems* 12.1-2 (1992), pp. 3–94.

[34] G. B. Dantzig. "Linear programming under uncertainty". In: *Manamgement Science* 1 (1955), pp. 197–206.

[35] L. Demir, S. Tunali, and D. T. Eliiyi. "The state of the art on buffer allocation problem: A comprehensive survey". In: *Journal of Intelligent Manufacturing* 25.3 (2014), pp. 371–392. ISSN: 0956-5515. DOI: 10.1007/s10845-012-0687-9.

[36] L. Demir, S. Tunal, and D. T. Eliiyi. "An adaptive tabu search approach for buffer allocation problem in unreliable non-homogenous production lines". In: *Computers & Operations Research* 39.7 (2012), pp. 1477–1486.

[37] E. W. Dijkstra. “A note on two problems in connexion with graphs”. In: *Numerische mathematik* 1.1 (1959), pp. 269–271.

[38] M. Drótos and T. Kis. “Scheduling of inventory releasing jobs to minimize a regular objective function of delivery times”. In: *Journal of Scheduling* 16.3 (2013), pp. 337–346.

[39] T. Fliedner and J. Liesiö. “Adjustable robustness for multi-attribute project portfolio selection”. In: *European Journal of Operational Research* 252.3 (2016), pp. 931–946.

[40] J. V. Frasch. “Algorithms and complexity for steel production scheduling”. MA thesis. Technical University of Kaiserslautern, 2009.

[41] J. V. Frasch, S. O. Krumke, and S. Westphal. “MIP formulations for flowshop scheduling with limited buffers”. In: *International Conference on Theory and Practice of Algorithms in (Computer) Systems*. Springer. 2011, pp. 127–138.

[42] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness*. Vol. 1. WH Freeman and Company, New York, 1979.

[43] A. M. Geoffrion. “Generalized benders decomposition”. In: *Journal of Optimization Theory and Applications* 10.4 (1972), pp. 237–260.

[44] S. B. Gershwin. “An efficient decomposition method for the approximate evaluation of tandem queues with finite storage space and blocking”. In: *Operations Research* 35.2 (1987), pp. 291–305.

[45] S. B. Gershwin and J. E. Schor. “Efficient algorithms for buffer space allocation”. In: *Annals of Operations Research* 93.1-4 (2000), pp. 117–144.

[46] M. Goerigk and A. Schöbel. “Algorithm engineering in robust optimization”. In: *Algorithm engineering*. Springer, 2016, pp. 245–279.

[47] R. L. Graham et al. “Optimization and approximation in deterministic sequencing and scheduling: a survey”. In: *Annals of discrete mathematics*. Vol. 5. Elsevier, 1979, pp. 287–326.

[48] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer-Verlag, Berlin Heidelberg, 1988.

[49] G. Gürkan. “Simulation optimization of buffer allocations in production lines with unreliable machines”. In: *Annals of Operations Research* 93.1-4 (2000), pp. 177–216.

[50] P. Györgyi and T. Kis. "Approximation schemes for single machine scheduling with non-renewable resource constraints". In: *Journal of Scheduling* 17.2 (2014), pp. 135–144.

[51] P. Györgyi and T. Kis. "Approximability of scheduling problems with resource consuming jobs". In: *Annals of Operations Research* 235.1 (2015), pp. 319–336.

[52] P. Györgyi and T. Kis. "Reductions between scheduling problems with non-renewable resources and knapsack problems". In: *Theoretical Computer Science* 565 (2015), pp. 63–76.

[53] P. Györgyi and T. Kis. "Approximation schemes for parallel machine scheduling with non-renewable resources". In: *European Journal of Operational Research* 258.1 (2017), pp. 113–123.

[54] F. Harary. *Graph theory*. Addison-Wesley Publishing Company, Inc., 1972.

[55] S. Helber et al. "Using linear programming to analyze and optimize stochastic flow lines". In: *Annals of Operations Research* 182.1 (2011), pp. 193–211.

[56] W. Höhn. "Flowshop-scheduling in der Stahlindustrie". MA thesis. Technical University of Berlin, 2007.

[57] J. R. Jackson. "Scheduling a production line to minimize maximum tardiness". In: *Management Science Research Project* (1955).

[58] A. R. Kan. *Machine scheduling problems: classification, complexity and computations*. Martinus Nijhoff, The Hague, 1976.

[59] A. M. Koster and M. Kutschka. "Network design under demand uncertainties: A case study on the abilene and GEANT network data". In: *Photonic Networks, 12. ITG Symposium*. VDE. 2011, pp. 1–8.

[60] A. M. Koster, M. Kutschka, and C. Raack. "Robust network design: Formulations, valid inequalities, and computations". In: *Networks* 61.2 (2013), pp. 128–149.

[61] S. O. Krumke and H. Noltemeier. *Graphentheoretische Konzepte und Algorithmen*. Springer-Verlag, 2009.

[62] P. Laborie. "Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results". In: *Artificial Intelligence* 143.2 (2003), pp. 151–188.

[63] R. Levantesi, A. Matta, and T. Tolio. "A new algorithm for buffer allocation in production lines". In: *Proceedings of the Third Aegean International Conference on Design and Analysis of Manufacturing Systems*. 2001, pp. 19–22.

[64] J. MacGregor Smith. "Joint optimisation of buffers and network population for closed finite queueing systems". In: *International Journal of Production Research* 54.17 (2016), pp. 5111–5135.

[65] J. MacGregor Smith and F. Cruz. "The buffer allocation problem for general finite buffer queueing networks". In: *IIE Transactions* 37.4 (2005), pp. 343–365.

[66] A. Matta. "Simulation optimization with mathematical programming representation of discrete event systems". In: *Simulation Conference, 2008. WSC 2008. Winter*. IEEE. 2008, pp. 1393–1400.

[67] J. M. Moore. "An n job, one machine sequencing algorithm for minimizing the number of late jobs". In: *Management Science* 15.1 (1968), pp. 102–109.

[68] E. Morsy and E. Pesch. "Approximation algorithms for inventory constrained scheduling on a single machine". In: *Journal of Scheduling* 18.6 (2015), pp. 645–653.

[69] N. Nahas. "Buffer allocation and preventive maintenance optimization in unreliable production lines". In: *Journal of Intelligent Manufacturing* 28.1 (2017), pp. 85–93.

[70] G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*. John Wiley & Sons, 1999.

[71] K. Neumann and C. Schwindt. "Project scheduling with inventory constraints". In: *Mathematical Methods of Operations Research* 56.3 (2003), pp. 513–533.

[72] K. Neumann, C. Schwindt, and N. Trautmann. "Scheduling of continuous and discontinuous material flows with intermediate storage restrictions". In: *European Journal of Operational Research* 165.2 (2005), pp. 495–509.

[73] J. Olhager. "Evolution of operations planning and control: from production to supply chains". In: *International Journal of Production Research* 51.23-24 (2013), pp. 6836–6843.

[74] T. Park. "A two-phase heuristic algorithm for determining buffer sizes of production lines". In: *International Journal of Production Research* 31.3 (1993), pp. 613–631.

[75] M. Pinedo. *Scheduling: Theory, algorithms, and systems*. 3rd. New York: Springer, 2008. ISBN: 9781461419860.

[76] S. G. Powell and D. F. Pyke. "Allocation of buffers to serial production lines with bottlenecks". In: *IIE Transactions* 28.1 (1996), pp. 18–29.

[77] E. Saliby. "Descriptive sampling: a better approach to Monte Carlo simulation". In: *Journal of the Operational Research Society* 41.12 (1990), pp. 1133–1142. (Visited on 07/12/2016).

[78] E. Saliby. "Understanding the Variability of Simulation Results: An Empirical Study". In: *Journal of the Operational Research Society* 41.4 (1990), pp. 319–327. ISSN: 0160-5682. DOI: 10.1057/jors.1990.53.

[79] A. Schrijver. *Theory of linear and integer programming*. John Wiley & Sons, 1986.

[80] W. E. Smith. "Various optimizers for single-stage production". In: *Naval Research Logistics Quarterly* 3.1-2 (1956), pp. 59–66.

[81] F. Sourd and J. Rogerie. "Continuous filling and emptying of storage systems in constraint-based scheduling". In: *European Journal of Operational Research* 165.2 (2005), pp. 510–524.

[82] A. L. Soyster. "Convex programming with set-inclusive constraints and applications to inexact linear programming". In: *Operations Research* 21.5 (1973), pp. 1154–1157.

[83] D. D. Spinellis and C. T. Papadopoulos. "A simulated annealing approach for buffer allocation in reliable production lines". In: *Annals of Operations Research* 93.1-4 (2000), pp. 373–384.

[84] R. Stolletz and S. Weiss. "Buffer allocation using exact linear programming formulations and sampling approaches". In: *IFAC Proceedings Volumes* 46.9 (2013), pp. 1435–1440. ISSN: 14746670. DOI: 10.3182/20130619-3-RU-3018.00461.

[85] M. Thorup. "Integer priority queues with decrease key in constant time and the single source shortest paths problem". In: *Journal of Computer and System Sciences* 69.3 (2004), pp. 330–353.

[86] S. Weiss, A. Matta, and R. Stolletz. "Optimization of buffer allocations in flow lines with limited supply". In: *IISE Transactions* 50.3 (2018), pp. 191–202.

[87] S. Weiss, J. A. Schwarz, and R. Stolletz. "The buffer allocation problem in production lines: Formulations, solution methods, and instances". In: *IISE Transactions* 51.5 (2019), pp. 456–485.

[88] S. Weiss and R. Stolletz. "Buffer allocation in stochastic flow lines via sample-based optimization with initial bounds". In: *OR Spectrum* 37.4 (2015), pp. 869–902. ISSN: 0171-6468. DOI: 10.1007/s00291-015-0393-z.

[89] Y. Yin, K. E. Stecke, and D. Li. "The evolution of production systems from Industry 2.0 through Industry 4.0". In: *International Journal of Production Research* 56.1-2 (2018), pp. 848–861.

[90] M. Zhang, A. Matta, and G. Pedrielli. "Discrete event optimization: Workstation and buffer allocation problem in manufacturing flow lines". In: *Proceedings of the 2016 Winter Simulation Conference*. IEEE Press. 2016, pp. 2879–2890.

Appendix A

Full Numerical Study of the Nominal BAP with Initial Bounds

In this appendix, we provide the all the numerical results for the nominal BAP, some of which were left out of Chapter 4. The parameters that are being compared are: the type of algorithm (classical or global bounds method), the use of an initial heuristic solution, the use of subsystems lower bounds and the use of initial upper bounds. Unfortunately, unlike lower bounds, the subsystems upper bounds turned out to greatly increase the required computational effort in most cases. These tests are run for four different values of the goal throughput TH^*. Indeed, the latter strongly impacts the value of the optimal buffer sum, which in turn, determines computation times.

A setting is defined by an algorithm type and the presence of an initial heuristic, upper bounds and lower bounds. The algorithm type is indicated by "THN" for the classical method using the normal definition of the throughput, or by "THP" for the global bounds method using the augmented throughput. The presence of upper bounds, lower bounds and a heuristic initial solution are denoted by the tokens "UB", "LB" and "H" respectively (in that order). Their absence is shown with an "N". Note that the THP-algorithm is always carried out with the "H"-token, as computing an initial heuristic is part of the algorithm.

Surprisingly, the results in Table A.1 show that computing an initial heuristic solution slightly worsens the computation times. Although a heuristic always improves computation speed, this seems to be balanced out by the time taken to compute that heuristic, for this "small" instance (with only 6 buffers). From the direct comparison of both algorithms, it results that THP clearly performs better. As for the upper bounds, they multiply computation times by roughly 4. Obviously, spending the time to compute them is not worth the gains they account for.

Table A.1: Mean buffers and computation times for the BAP with $TH^* = 4.5$

Setting				Mean optimum	Min time	Mean time	Max time
THN	N	N	N	6.0	48.92	86.72	284.06
THN	N	N	H	6.0	62.50	90.57	289.80
THN	N	LB	H	6.0	69.70	73.35	93.69
THN	UB	LB	H	6.0	236.92	276.46	382.05
THP	N	N	H	6.0	35.57	59.20	71.66
THP	N	LB	H	6.0	39.66	62.64	68.86
THP	UB	LB	H	6.0	161.50	217.89	353.31

Table A.2: Mean buffers and computation times for the BAP with $TH^* = 5.0$

Setting				Mean optimum	Min time	Mean time	Max time
THN	N	N	N	11.08	371.61	416.68	488.64
THN	N	N	H	11.08	266.35	333.37	425.38
THN	N	LB	H	11.08	61.14	97.21	129.26
THN	UB	LB	H	11.08	278.94	436.06	594.62
THP	N	N	H	11.08	296.42	315.62	379.69
THP	N	LB	H	11.08	100.25	108.25	131.72
THP	UB	LB	H	11.08	344.80	492.49	605.04

Contrarily to the results obtained with $TH^* = 4.5$, Table A.2 shows that the initial heuristic now substantially speeds up computations. Also, it can be noted that THP now performs only slightly better than THN without lower bounds, and even less good than THN when lower bounds are used. The upper bounds, while still being very counterproductive, yield computations times that are comparable with the setting without any bounds at all.

Table A.3: Mean buffers and computation times for the BAP with $TH^* = 5.5$

Setting				Mean optimum	Min time	Mean time	Max time
THN	N	N	N	19.0	596.96	674.86	785.47
THN	N	N	H	19.0	884.91	1038.96	1215.06
THN	N	LB	H	19.0	285.43	683.12	1002.40
THN	UB	LB	H	19.0	3231.43	4282.84	4941.34
THP	N	N	H	19.0	773.28	906.47	1037.85
THP	N	LB	H	19.0	227.29	293.04	372.57
THP	UB	LB	H	19.0	2935.81	3057.32	3202.38

The results for a goal throughput $TH^* = 5.5$ presented in Table A.3, while showing much higher values, resemble the case with $TH^* = 4.5$ more than with $TH^* = 5.0$. Again, the heuristic solution significantly slows computation, while the global bounds approach THP performs up to a factor 2 better than the classical THN. Once more, the upper bounds are outperformed by any other setting.

Table A.4: Mean buffers and computation times for the BAP with $TH^* = 6.0$

Setting				Mean optimum	Min time	Mean time	Max time
THN	N	N	N	35.78	3242.10	3975.18	4628.03
THN	N	N	H	35.78	3239.40	4156.89	5441.01
THN	N	LB	H	35.78	1242.30	1977.30	3324.46
THN	UB	LB	H	35.78	4138.18	4624.21	5207.06
THP	N	N	H	35.78	4725.04	5446.85	6558.62
THP	N	LB	H	35.78	1330.51	1807.51	2274.83
THP	UB	LB	H	35.78	4233.30	4595.20	5097.59

Table A.4 presents the results for the largest instance of the BAP, where $TH^* = 6.0$. The differences between all settings seem a bit smoother than for the other values of TH^*. Computing an initial heuristic solution still worsens computation speed, but not as extremely as for $TH^* = 5.5$. The global bounds algorithm THP remains the better choice in presence of a heuristic and lower bounds, yet is slightly outperformed when no lower bounds are used. Interestingly, the use of upper bounds (combined with lower ones) makes the global bounds algorithm faster than with no bounds at all.

Last, when considering the amplitudes between the minimum and maximum times, these seem to be significantly smaller, at all values of TH^*, for the THP algorithm than for THN. This is especially true when both a heuristic and lower bounds are computed.

Appendix B

Complexity Tables for Scheduling with Prefabrication

Problem $1\|\text{prfb}_1, *\|\sum_j U_j$	Complexity	Explanation
δ_1^+	open (also for $L_{\max}$)	
$\delta_1^+, \quad \delta_1^-$	$O(n \log n)$	Table 5.1
$\delta_1^+, \quad p_j^+ = \underline{p}$	$O(n \log n)$	Table 5.1
$\delta_1^+, \quad p_j^- = \underline{p}$	$O(n \log n)$	Table 5.1
δ_1^-	str. $\mathcal{NP}$-hard (also for $L_{\max}$)	Corollary 4
$\delta_1^-, \quad p_j^+ = \underline{p}$	$O(n \log n)$	Table 5.1
$\delta_1^-, \quad p_j^- = \underline{p}$	str. $\mathcal{NP}$-hard (also for $L_{\max}$)	Corollary 4
$p_j^+ = \underline{p}$	open (also for $L_{\max}$)	
$p_j^+ = \underline{p}_1, \quad p_j^- = \underline{p}_2$	$O(n \log n)$	Table 5.1
$p_j^- = \underline{p}$	str. $\mathcal{NP}$-hard (also for $L_{\max}$)	Corollary 4
–	str. $\mathcal{NP}$-hard (also for $L_{\max}$)	Corollary 4

Table B.1: Complexity results for $\sum_j U_j$ and related objectives

Problem $1\|\text{prfb}_1, *\|\sum_j w_j C_j$	Complexity	Explanation
δ_1^+	open (also for $\sum_j C_j$)	
$\delta_1^+, \quad \delta_1^-$	open (also for $\sum_j C_j$)	
$\delta_1^+, \qquad p_j^+ = \underline{p}$	$\mathcal{O}(n \log n)$	Theorem 25
$\delta_1^+, \qquad p_j^- = \underline{p}$	open ($\mathcal{O}(n \log n)$ for $\sum_j C_j$)	Sort by non-increasing δ_j for $\sum_j C_j$
δ_1^-	open ($\mathcal{O}(n \log n)$ for $\sum_j C_j$)	Theorem 23 for $\sum_j C_j$
$\delta_1^-, \quad p_j^+ = \underline{p}$	open ($\mathcal{O}(n \log n)$ for $\sum_j C_j$)	Theorem 23 for $\sum_j C_j$
$\delta_1^-, \qquad p_j^- = \underline{p}$	open ($\mathcal{O}(n \log n)$ for $\sum_j C_j$)	Theorem 23 for $\sum_j C_j$
$p_j^+ = \underline{p}$	open (also for $\sum_j C_j$)	
$p_j^+ = \underline{p}_1, \quad p_j^- = \underline{p}_2$	open ($\mathcal{O}(n \log n)$ for $\sum_j C_j$)	Theorem 24 for $\sum_j C_j$
$p_j^- = \underline{p}$	str. $\mathcal{NP}$-hard (also for $\sum_j C_j$)	Corollary 5
–	str. $\mathcal{NP}$-hard (also for $\sum_j C_j$)	Corollary 5

Table B.2: Complexity results for $\sum_j w_j C_j$ and related objectives

Appendix C

Scientific Career

Oct. 2016 - Sep. 2019	**PhD study** in Mathematics, funded by a scholarship of the Fraunhofer ITWM *Technische Universität Kaiserslautern*, Germany
May 2016 - Sep. 2016	**Project Studies in Advanced Technology (ProSAT)** *Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM*, Kaiserslautern, Germany
Sep. 2010 - Apr. 2016	**Diplôme d'Ingénieur (Master's Degree)** in mathematical engineering *Institut National des Sciences Appliquées (INSA)*, Rouen, France

Appendix D

Wissenschaftlicher Werdegang

Okt. 2016 - Sep. 2019	**Promotionsstudium** in Mathematik, mit einem Stipendium vom Fraunhofer ITWM *Technische Universität Kaiserslautern*, Deutschland
Mai 2016 - Sep. 2016	**Project Studies in Advanced Technology (ProSAT)** *Fraunhofer-Institut für Techno- und Wirtschaftsmathematik ITWM*, Kaiserslautern, Deutschland
Sep. 2010 - Apr. 2016	**Diplôme d'Ingénieur (Master-Studium)** in Mathematik *Institut National des Sciences Appliquées (INSA)*, Rouen, Frankreich

Scheduling problems play an important role in the area of production planning. However, due to e.g. uncertainties, real-world applications may induce additional constraints, and lead to intractable models. In the literature, approximated solutions are often computed. This thesis aims to derive exact yet tractable algorithms for different scheduling problems under robustness or inventory constraints.

First, we consider the notoriously NP-hard Buffer Allocation Problem (BAP) in flow lines. In its classical approach, it assumes that the processing times of jobs are known in advance. Realistically, this is not the case. Therefore, we present a model for the BAP with additional robustness constraints. We compute exact solutions and demonstrate the tractability of our method.

Next, we lay focus on inventory-constrained scheduling. In this setting, jobs are assumed to add or remove a given amount of material from a common stack. We identify a new class of such problems, where the objective function only depends on the consuming jobs. We provide complexity results and algorithms for variations of the problem with different objective functions and constraints.

ISBN 978-3-8396-1609-3

9 783839 616093

FRAUNHOFER VERLAG