UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL

THIAGO CANTOS LOPES

CYCLICAL SCHEDULING AND ASSEMBLY LINE BALANCING

TESE

CURITIBA
2021

THIAGO CANTOS LOPES

CYCLICAL SCHEDULING AND ASSEMBLY LINE BALANCING

PROGRAMAÇÃO CÍCLICA E BALANCEAMENTO DE LINHAS DE MONTAGEM

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial, da Universidade Tecnológica Federal do Paraná como requisito parcial à obtenção do título de Doutor em Ciências.

Área de Concentração: Engenharia de Automação e Sistemas.

Orientador: Prof. Dr. Leandro Magatão
(UTFPR, Brasil)
Coorientadora: Profa. Dra. Nadia Brauner
(Université Grenoble-Alpes, França)

CURITIBA

2021

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Curitiba

THIAGO CANTOS LOPES

CYCLICAL SCHEDULING AND ASSEMBLY LINE BALANCING

Trabalho de pesquisa de doutorado apresentado como requisito para obtenção do título de Doutor Em Ciências da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Engenharia De Automação E Sistemas .

Data de aprovação: 21 de Maio de 2021

Prof Cassius Tadeu Scarpin, Doutorado - Universidade Federal do Paraná (Ufpr)

Prof Flavio Neves Junior, Doutorado - Universidade Tecnológica Federal do Paraná

Prof.a Nadia Brauner, Doutorado - Université Grenoble Alpes

Prof Reinaldo Morabito Neto, Doutorado - Universidade Federal de São Carlos (Ufscar)

Prof Ricardo Luders, Doutorado - Universidade Tecnológica Federal do Paraná

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 21/05/2021.

ACKNOWLEDGEMENTS

This thesis closes a chapter in a journey in which I did not walk alone. I here acknowledge those that helped me go the distance in these doctoral years.

I thank my father Milton, my mother Solange, and my family for the love and support - and for patiently listening to me when I talked about mathematics. I thank my fiancée Ingrid for making my journey so especially meaningful and joyful that it is actually worth having. I thank my friends for adding spice to that journey. I thank my mentor, Leandro, for being a door opener, a guiding hand, and a fair (and prolific) critic. I thank my professors for the tools they gave me and my colleagues for the helping me learn how to use them. I thank my co-authors for bearing with my hopefully useful, but certainly too numerous, e-mails. I thank Nadia for trusting and welcoming me when I was a stranger in a foreign land. I thank my Country, for giving me more than I could ever ask to go far, and for also being a home to return to. And I thank God for the very generous blessing He gave and gives me in His mysterious ways...

Finally, I thank the financial support without which this most optimal journey would be unfeasible. This work has been funded by Fundação Araucária (Agreements 041/2017 FA – UTFPR – RENAULT) from 2017 to 2019, by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001, from 2019 to 2020, and the Graduate Program in Electrical and Computer Engineering (CPGEI) from 2020 to 2021 (Edital DIRPPG-CT 01/2020).

“All that is gold does not glitter,
Not all those who wander are lost;
The old that is strong does not wither,
Deep roots are not reached by the frost.

From the ashes a fire shall be woken,
A light from the shadows shall spring;
Renewed shall be blade that was broken,
The crownless again shall be king.”

The Fellowship of the Ring (Tolkien, J. R. R.).

RESUMO

LOPES, T. C.. **Programação Cíclica e Balanceameto de Linhas de Montagem**. 2021. 220 f. Tese – Doutorado em Ciências pelo Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI) – Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

Esta tese considera extensões do problema de balanceamento de linha de montagem ligados a *scheduling* cíclico, propondo modelos matemáticos e métodos de solução para resolver tais problemas. Cada um dos capítulos principais da presente é baseado em uma publicação específica pelo autor durante seu doutorado. As principais contribuições desta tese incluem: formulações matemáticas que descrevem o regime permanente de linhas de montagem de modelo misto operando sob sequências cíclicas; uma decomposição para resolver o problema de balanceamento quando sequenciamento e alocações de *buffer* são incorporados como variáveis de decisão; uma simheurística para balancear linhas de modelo misto quando a sequência de produtos é estocástica; correções para um artigo anterior (de outro autor) de balanceamento com estações multi-operadas; uma quebra de paradigma para linhas de estações multi-operadas, bem como uma decomposição eficiente e limites inferiores para o problema resultante; uma descrição dos custos de estoque interno associados a alocações fracionárias de tarefas no contexto de balanceamento dinâmico ou compartilhamento de tarefas; um método para resolver problemas multi-objetivos com fronteiras Pareto lineares por partes, juntamente a um estudo de caso que permite comparar diferentes sistemas de passo da linha no contexto de sequências cíclicas de produtos. Em vários capítulos, comparações a *benchmarks* da literatura demonstram a superioridade dos métodos e formulações propostas. Em outras, suas contribuições e descrições analíticas respondem perguntas ligadas a considerações gerenciais importantes tal qual o *bowl phenomenon*, custos de estoque interno, e a influência dos tipos de passo de linha para sua performance.

Palavras-chave: Balanceamento de linhas de montagem. Programação cíclica. Pesquisa Operacional. Programação linear inteira-mista.

ABSTRACT

LOPES, T. C.. **Cyclical Scheduling and Assembly Line Balancing**. 2021. 220 p. Thesis – Doctorate in Science by the Graduate Program in Electrical and Computer Engineering (CPGEI) – Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

This thesis addresses extensions of the assembly line balancing problem tied to cyclical scheduling and proposes mathematical models solution methods tackle such problems. Each of the thesis main chapter is based on a specific publication by the author throughout his doctorate. The main contributions include: mathematical formulations that can describe the steady state of mixed-model assembly lines when the product sequence is cyclical; a decomposition to solve the balancing problem when product sequencing and buffer allocations are also decision variables; a simheuristic to balance mixed-model lines when the product sequence is stochastic; corrections to a previous multi-manned balancing article (by another author); a paradigm shift for multi-manned lines that allow shorter line lengths, along with an efficient decomposition and lower bounds for the resulting problem; a description of internal storage costs tied to fractional task allocations in the context of dynamic line balancing or work-sharing; a method to solve multi-objective problems with part-wise linear Pareto fronts, and a case that allows comparing different line control systems in the context of cyclical product sequences. In several chapters, comparisons to literature benchmarks demonstrate the superiority of the proposed methods and formulations. In others, analytical descriptions and contributions answer research questions related to important managerial considerations such as the bowl phenomenon, internal storage costs, and the influence of line control types to line performance.

Keywords: Assembly line balancing. Cyclical scheduling. Operations research. Mixed-integer linear programming.

LIST OF ALGORITHMS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

The following list presents the parameters that interact with variables in the mathematical models contained in this document. For some expressions the number of elements in a set is used. For instance, $|T|$ states the number of tasks in the set T. Some letters have different meanings across chapters, this list indicates when that occurs for similarly named symbols.

PARAMETERS

A	Parameter matrix tied to integer variables, used in Chapter 10
A_s	Available space at stage s, used in Chapter 4
A_t	Worker assignment of task t, used in Chapter 7
B	Parameter matrix tied to continuous variables, used in Chapter 10
B_i	Boundaries between stages, used in Chapter 4
B_{max}	Number of available buffers, used in Chapter 5
B_s	Base processing time of station s in a task-sharing block, used in Chapter 9
$\mathbf{C}$	Right-hand-side vector tied to constraints, used in Chapter 10
C_m	Theoretical minimum cycle time for product model m
CT	Cycle time - is a parameter in Chapters 7 and 8
$\overline{CT}$	Cycle time of an incumbent solution
D_t	Duration of task t in single model problems, used in Chapters 7-9
$D_{t,m}$	Duration of task t for product model m
$\overline{D}_t$	Duration of task t summed across all products in the minimal part set
$\delta_s^{+/-}$	Positive/Negative difference between processing time and cycle time of station s
$E[*]$	Expected value of stochastic variable $*$, used in Chapter 6
ES_t	Earliest station to which task t can be assigned
F	Set of feasible task-station allocations $(t,\, s)$ tuples, used in Chapter 2
$F(z)$	Line length cost factor tied to fractional allocation z
$G_{t,s}$	Buffer cost tied to fractionally allocating task t to station s
H	A large number, used for Big-M conditional constraint relaxation
i	Positional index in Algorithms
J	Buffer capacity, used in Chapter 9
j_s	Correction factor for stage s, used to generalize Chapter 4's formulation
k_s	Parallelism degree at stage s, used in Chapter 4
K	Maximum denominator for fractional task allocations, used in Chapter 9
LS_t	Latest station to which task t can be assigned
M	Set of product models m

m_p	Product model of piece p
N_m	Number of products of model m in the minimal part set
O_m	Occupation or demand rate of model m
P	Set of product pieces p to be scheduled
$\overline{\pi}$	Value of dual-variable π
$Q_{t,s}$	Space requirement for task t in station s
R	Set of precedence relations (t_1, t_2): task t_1 precedes task t_2
R_m	Set of precedence relations (t_1, t_2) for product model m
S	Set of stations s, in Chapter 4 it states the set of stages instead
S_{sync}	Set of stations with synchronous product transfers
S_{async}	Set of stations with asynchronous product transfers
$\overleftarrow{s}$	station or buffer immediately preceding s, used in Chapter 5
S_b	Set of buffers, a subset of S
T	Set of tasks t
TAx_m	Theoretical minimum average processing time for product model m
V	Conveyor belt's velocity in paced lines
W	Set of workers w
w_{max}	Maximum number of workers per multi-manned station
ω_p	Time spent in a buffer by product p
$\Omega(\overline{\mathbf{x}})$	Feasible region given integer solution $\overline{\mathbf{x}}$
Ω_{seg}	Feasible region under a given segment in the criterion-space

Variables in the mathematical models are separated between integer and continuous ones. The key decision variable set in each chapter is usually represented by the letter x, followed by the appropriate subscripts. When the same starting letter has different meanings across chapters, this list present each in the order they appear.

BINARY AND INTEGER VARIABLES

$cf_{m_1,m_2,s}$	Set to 1 if model m_2 cyclically follows model m_1 at station s
$f_{m_1,m_2,s}$	Set to 1 if model m_2 follows model m_1 at station s
f_{t_1,t_2}	Set to 1 if task t_2 is performed after task t_2, used in Chapters 4 and 8
$x_{t,s}$	Set to 1 if task t is assigned to station s
$x_{t,s,m}$	Set to 1 it task t is assigned to station s for product model m, used in Chapter 4
$x_{w,s}$	Set to 1 if worker w is assigned to station s, used in Chapter 7
$x_{t,w}$	Set to 1 if task t is assigned to worker w, used in Chapter 8
$\mathbf{x}$	Set of all integer variables, used in Chapter 10
$y_{p,m}$	Set to 1 if piece p is set to product model m
$y_{m,n,s}$	Set to 1 model m is the n^{th} one to enter the station s, used in Chapter 4
y_s	Set to 1 if station s is open, used in Chapter 7
$y_{t,s}$	Set to 1 if task t is fractionally assigned to station s, used in Chapter 9
z_s	Set to 1 if a buffer is assigned at candidate position s
$z_{t,s}$	Set to 1 if task t is assigned to station s, used in Chapter 9

Similarly, continuous variables are listed bellow. These are mostly those tied to the scheduling aspect of the studied problems. These variables' meaning change less often between chapters. Nonetheless, these different purposes are also listed in the order they appear.

CONTINUOS VARIABLES

CT	Cycle time
$px_{m,s}$	Processing time of model m at station s
Px_s	Average processing time at station s
π	Dual-variable tied to a constraint
L	Line length
$Lmax_s$	Ending position of station s
$Lmin_s$	Starting position of station s
$Lend_{p,s}$	Position processing of product piece p at station s is completed
$Lin_{p,s}$	Entry position of product piece p at station s
$Tend_{p,s}$	Time processing of product piece p at station s is completed
$Tin_{p,s}$	Entry time of product piece/model p at station s
$Tout_{p,s}$	Departure time of product piece/model p at station s
$Tstart_t$	Starting position of task t
$Tstart_{p,s}$	Starting time of processing of product p at station s, used in Chapter 10
$Tx_{p,s}$	Processing time of product piece/model p at station s
W_{end_w}	Finishing position for worker w
W_{start_w}	Starting position for worker w
y	Set of all continuous variables, used in Chapter 10
Z_i	Mixed-integer linear objective i

CONTENTS

1 INTRODUCTION

1.1 THEMES AND METHODS

Assembly line balancing is a widely studied class of optimization problems tied to that manufacturing system. Boysen *et al.* (2008) present a particularly useful classification of such problems. At its most basic form, line balancing has a rather simple definition: to distribute tasks of deterministic durations to stations of equal capacity in a manner that respects precedence relations and optimizes a given goal function. Even that simpler problem is already NP-Hard (ÁLVAREZ-MIRANDA; PEREIRA, 2019). Many of the variants described by Boysen *et al.* (2008) focus on specific complications that arise by considering practical factors that force deviations from the most basic definition. These can include product diversity, task particularities, line features, station features, worker limitations, among others (BATTAÏA; DOLGUI, 2013).

Cyclical scheduling is any type of scheduling that repeats after a specific cycle time. Levner *et al.* (2010) present a classification of cyclical scheduling optimization problems. These are often tied to sequencing decisions regarding jobs of given processing time durations. In a general sense, assembly lines operate under cyclical schedules as each station is typically bound by the cycle time. This means that tasks being performed repeat throughout the line every cycle time. The simplest line balancing problems adopts a rather straightforward simplification: if the sum of processing times of all tasks assigned to a station is less or equal to the cycle time, then the cycle time is respected. While this consideration is valid for multiple classes of problems, this thesis considers opportunities tied to cases in which it is not and in which more adequate considerations are possible when explicitly considering cyclical scheduling when balancing the line. A natural example is mixed-model lines, i.e. lines shared between multiple products (BOYSEN *et al.*, 2008). In them, differences between products can help compensate for model-specific processing times that are higher than the cycle time, as shown in Chapter 2.

This thesis employs several Operational Research techniques: mixed-integer linear programming mathematical models, decompositions, simulations, heuristics, and lower bound algorithms. Readers can refer to references such as Hillier and Lieberman (2015) for an overview or introduction to these techniques, as the following chapters presuppose certain familiarity with them. Chapter 9 also includes a complexity proof by reduction. Readers can refer to Garey and Johnson (1979) for that field.

1.2 OBJECTIVES

This thesis aim is to study assembly line balancing problems in which cyclical scheduling plays a key role. Regarding these problem variants, this thesis aims specifically to:

1. Develop mathematical models to describe cyclical schedules in assembly lines;
2. Adapt these models to incorporate features such as parallel stations, and buffer allocation;
3. Describe the line control's role (i.e. conveyor belt system) and its practical implications;
4. Develop methods that allow efficient solution for problem variants.

1.3 RESEARCH OUTLINE AND SCOPE

This thesis compiles and presents the main contributions published by the author during his doctorate. Chapters 2-10 are adapted versions of the author's publication. These all consider assembly line balancing variants to which cyclical scheduling is applied in some capacity. The present chapter introduces both of these themes, and presents an overview of the contributions and how each chapter relates to the others. Chapters 2-10 are each based on a publication, meaning they are somewhat self-contained and present a "related works" section that contextualizes the chapter's contribution. Because these chapters address closely-related problem variants, a significant portion of their literature reviews is common. The author opted to maintain these context and related works sections rather than to combine them in a broader introduction for two reasons: first, to maintain the self-contained nature of each chapter; and second, to avoid an overly large literature review chapter that would probably fail to simultaneously contextualize all chapters. Each article that based this thesis had its own mathematical notation. These were adapted to present a more uniform tie between concepts and symbols throughout this document. However, some letters have chapter-specific meanings in mathematical models. The List of Symbols, present in this thesis preamble, detail these meanings to clarify this point. There are also some differences between chapters in terms of style, language, and structure. These reflect journal requirements and policies, but they also flow from the author's personal development throughout the doctorate. The publications that base this thesis are hereafter listed.

1.4 PUBLICATIONS

This thesis is mainly based on articles published in scientific journals and written by the author during his doctorate. These are hereafter listed in the order they appear in this thesis chapters:

1. *Lopes, T.C., Sikora, C.G.S., Michels, A.S., Magatão, L.*. Mixed-Model Assembly Line Balancing with Given Buffers and Product Sequence: Model, Formulation Comparisons and Case Study. **Annals of Operations Research**, v. 286, p. 475–500, 2020 - Base for Chapter 2;

2. *Lopes, T.C., Sikora, C.G.S., Michels, A.S., Molina, R.G., Magatão, L.*. Balancing and cyclically sequencing synchronous, asynchronous, and hybrid unpaced assembly lines. **International Journal of Production Economics**, v. 203, p. 216–224, 2018 - Base for Chapter 3;

3. *Lopes, T.C., Sikora, C.G.S., Michels, A.S., Magatão, L.*. Balancing and cyclical scheduling of asynchronous mixed-model assembly lines with parallel stations. **Journal of Manufacturing Systems**, v. 50, p. 193–200, 2019 - Base for Chapter 4;

4. *Lopes, T.C., Sikora, C.G.S., Michels, A.S., Magatão, L.*. An iterative decomposition for asynchronous mixed-model assembly lines: combining balancing, sequencing, and buffer allocation. **International Journal of Production Research**, v. 58, n. 2, p. 615–630, 2020 - Base for Chapter 5;

5. *Lopes, T.C., Michels, A.S., Lüders, R., Magatão, L.*. A simheuristic approach for throughput maximization of asynchronous buffered stochastic mixed-model assembly lines. **Computers and Operations Research**, v. 115, p. 104863, 2020 - Base for Chapter 6;

6. *Lopes, T.C., Michels, A.S., Magatão, L.*. A note to: A hybrid algorithm for allocating tasks, operators, and workstations in multi-manned assembly lines. **Journal of Manufacturing Systems**, v. 52, p. 205–208, 2019 - Base for Chapter 7;

7. *Lopes, T.C., Pastre, G.V., Michels, A.S., Magatão, L.*. Flexible multi-manned assembly line balancing problem: Model, heuristic procedure,and lower bounds for line length minimization. **Omega**, v. 95, p. 102063, 2020 - Base for Chapter 8;

8. *Lopes, T.C., Brauner, N., Magatão, L.*. Assembly Line Balancing with Fractional Task Allocations. **International Journal of Production Research**, 2021 *in press* - Base for Chapter 9.

Some papers presented by the author in scientific conferences also play a role in this thesis, both laying building blocks for some of the journal publications and as a base for one of the chapters:

1. *Lopes, T.C.*[1]*, Sikora, C.G.S., Magatão, L.*. Buffer and Cyclical Product Sequence Aware Assembly Line Balancing Problem: Model and Steady-State Balancing Case Study. **Annals of the XLVIII Brazilian Symposium of Operations Research**. Vitória-ES, Brazil, p. 3458–3469, 2016 - Base for Lopes *et al.* (2020b);

2. *Lopes, T.C.*[2]*, Sikora, C.G.S., Michels, A.S., Magatão, L.*. New Model for Simultaneous Balancing and Cyclical Sequencing of Asynchronous Mixed-Model Assembly Lines with Parallel Stations. **Annals of the XLIX Brazilian Symposium of Operations Research**. Blumenau-SC, Brazil, p. 3570–3581, 2017 - Base for Lopes *et al.* (2019);

3. *Lopes, T.C., Sikora, C.G.S., Michels, A.S., Magatão, L.*. A Matheuristic for Makespan Minimization of Asynchronous Stochastic Mixed-Model Assembly Lines. **Annals of the L Brazilian Symposium of Operations Research**. Rio de Janeiro-RJ, Brasil, p. 833771, 2018 - Base for Lopes *et al.* (2020b);

4. *Lopes, T.C., Sikora, C.G.S., Michels, A.S., Silva, A.C.L., Magatão, L.*. Modeling the Stochastic Steady-State of Mixed-Model Asynchronous Assembly Lines with Markov Chains **Proceeding of the XIX Latin-Iberoamerican Conference on Operations Research, CLAIO**. Lima-Peru, p. 183–189, 2019 - Related to the problem studied in Chapter 6.

5. *Lopes, T.C.*[3]*, Brauner, N., Magatão, L.*. Optimally solving multi-objective MILP problems with part-wise continuous Pareto fronts. **Proceedings of the XXI ROADEF**. Montpellier-France, 2020 - Base for Chapter 10.

[1] This paper was selected amongst the five best articles presented in the XLVIII SBPO conference (2016).
[2] This paper was selected amongst the five best articles presented in the XLIX SBPO conference (2017).
[3] This paper shared the first prize of best student article presented in the XXI ROADEF conference (2020).

Finally, during the years the research compiled in this thesis was developed, the author participated in several publications. Some of those are more related to his work during his Master's degree (LOPES *et al.*, 2017)[4], but others are more directly related to chapters in this thesis:

1. *Sikora, C.G.S., Lopes, T.C., Magatão, L.*. Traveling worker assembly line (re)balancing problem: Model, reduction techniques, and real case studies. **European Journal of Operational Research**. v. 259, p. 949-971, 2017;

2. *Sikora, C.G.S., Lopes, T.C., Schibelbain, D., Magatão, L.*. Integer based formulation for the simple assembly line balancing problem with multiple identical tasks. **Computers and Industrial Engineering**. v. 104, p. 134-144, 2017;

3. *Michels, A.S., Lopes, T.C., Sikora, C.G.S., Magatão, L.*. The Robotic Assembly Line Design (RALD) problem: Model and case studies with practical extensions. **Computers and Industrial Engineering**. v. 120, p. 320-333, 2018;

4. *Michels, A.S., Lopes, T.C., Sikora, C.G.S., Magatão, L.*. A Benders' decomposition algorithm with combinatorial cuts for the multi-manned assembly line balancing problem. **European Journal of Operational Research**. v. 278, p. 796-808, 2019 - Related to Chapters 7 and 8;

5. *Michels, A.S., Lopes, T.C., Magatão, L.*. An exact method with decomposition techniques and combinatorial Benders' cuts for the type-2 multi-manned assembly line balancing problem. **Operations Research Perspectives**. v. 7, p. 100163, 2020 - Related to Chapters 7 and 8.

1.5 THESIS STRUCTURE

Chapter 2 considers a balancing problem aimed at optimizing steady-state cycle time of an asynchronous line with given buffers and cyclical product sequence. Its main contribution is defining and validating a cyclical scheduling mathematical formulation that adequately represents such steady state. Chapter 3 incorporates sequencing decision variables to Chapter 2's formulation and demonstrates the importance of explicitly considering line pace for steady-state optimization. Chapter 4 expands Chapter 3's balancing-sequencing model to describe the cyclical scheduling

[4] This work also lead to the development of an international patent (Number: FR3062492A1)

of parallel stations. By doing so, it produces better solutions than a recent literature benchmark that studied the same problem. Chapter 5 presents a further extension of Chapter 3's formulation by also considering buffer allocation decision variables. It also marks a departure from previous chapters, as it is the first to produce a solution method (decomposition) other than simply solving the problem's mathematical programming model.

Chapter 6 furthers that departure by presenting a solution method that is entirely independent of mathematical models. Furthermore, it considers a stochastic problem variant in which the line must be balanced for a given buffer layout but under randomized product sequences. Naturally, this stochastic component limits the role cyclical scheduling plays for this specific chapter.

Chapters 7 and 8 present contributions for another class of assembly line balancing problems, tied to multi-manned stations. These chapters consider single-product lines, therefore, cyclical scheduling plays a role within stations rather than between them as in Chapters 2-5. Chapter 7 presents a shorter contribution that questions some results of a recent article. Chapter 8 presents a more flexible multi-manned line concept that allows shorter line lengths. It resonates Chapter 3's discussion on the importance of explicitly considering line pace and it demonstrates that a multi-objective dispute between cycle time and line length is possible for continuous paced lines.

Chapter 9 addresses a line balancing problem that allows fractional task allocations. The chapter analyses the relationship between internal storage costs and cycle time. By doing so, it corroborates Chapter 8's observation that multi-objective disputes can happen between line length and cycle time in paced lines and reinforces the interpretation that the former can act as continuously distributable buffers. Chapter 10 returns to the mixed-model context of Chapters 2-5 by addressing this multi-objective relationship with a case study with Chapter 2's data.

Lastly, Chapter 11 summarizes the contributions of this thesis and discusses directions for further works for each of the studied problem variants. Some of these reflect ongoing works, while others are suggestions.

2 BALANCING GIVEN PRODUCT SEQUENCE AND BUFFERS

This chapter consists of an adapted version of an article published in the journal Annals of Operations Research (LOPES *et al.*, 2020b), which in turn is an extended version of a conference paper (LOPES *et al.*, 2016) that was selected amongst the five best papers presented at the XLVIII Brazilian Symposium of Operations Research held in Vitória-ES in 2016. While the article was published in print in 2020, it was actually accepted for publication in 2017. Nonetheless it serves as a base formulation for the following chapters. It addresses a mixed-model line balancing problem variant in which the line pace is asynchronous and cyclical product sequence and buffer layouts are given.

2.1 CONTEXT AND RELATED WORKS

Assembly lines are product oriented layouts, in which products (or pieces) move through the workstations on the line in a sequential manner. In each workstation, a set of tasks is performed before the next workstation's tasks are allowed to begin. An important decision problem related to managing such lines is to balance the distribution of tasks amongst stations (SCHOLL; BECKER, 2006). Common goals for this problem include minimizing the number of workstations required for a given production rate (Type-1) and maximizing the production rate for a given number of workstations (Type-2). This chapter focuses on a Type-2 variant of such problems in which a cyclical product sequence and a set of finite buffers are given as parameters.

Assembly lines are often not dedicated exclusively to a single product, but rather shared by a set of products (SCHOLL, 1999). A common example is found in assembly lines of automotive factories with different vehicle models. Lines with more than one product model can be further classified as in two different types (BOYSEN *et al.*, 2008): if set-up times between models are significant and must be taken into account, significant size batches of each product go through the line in an alternated fashion, and the problem is called a multi-model assembly line; If those set-up times are small and negligible, models flow through the line in a more mixed fashion, such line is called a mixed-model assembly line. This chapter focuses on the later type.

Authors approach mixed-model balancing differently (BECKER; SCHOLL, 2006): some (THOMOPOULOS, 1970; MERENGO *et al.*, 1999) state that it is important to minimize differences between model processing times across stations (station smoothing, horizontal bal-

ancing), others (MERENGO *et al.*, 1999; MATANACHAI; YANO, 2001; PASTOR *et al.*, 2002) state that it is best to equalize average workloads between stations allowing more differences between models (vertical balancing). Bukchin (1998) presents comparisons between several performance measures for mixed-model balancing. These formulations are indirect goals, and were developed because measuring the line throughput directly is a challenge.

Assembly lines can be further classified in terms of the line flow control (BOYSEN *et al.*, 2008): paced lines (continuous) have a conveyor belt that moves all pieces constantly and together; In unpaced synchronous lines, all products move together, but discretely one station forward when all pieces have completed processing at their current stations; In unpaced asynchronous lines, pieces move discretely and almost independently: each piece can move when processing is completed at its current station and the next station is available. This chapter focuses on the later type of line control, which offers interesting possibilities and challenges linked to the unpaced flow of pieces in and out of stations and buffers: On the one hand, products might be blocked (when they are done processing, but the next station is occupied) and stations starved (when a piece leaves the station and the previous station has not completed processing). On the other hand, the independent flow of pieces can help compensate the differences between models processing times.

Blockages and starvations produce disturbances on the assembly line that must be taken into account. If the product sequence is cyclical, the system converges towards a steady-state. However, such state is not trivially determinable due to the aforementioned disturbances. Furthermore, if the line starts empty, transient-stage disturbances further complicate such determination. Figures 1(a) and 1(b) compare the steady-state cyclical behavior to the transient behavior on a two-model (flowing through the line on a 5-1 pattern) assembly line with seven stations and single unitary buffers between them[1] Notice that the replications of the piece set behave (six pieces, five of model 1 and one of model 2) identically in Figure 1(a), and differently in Figure 1(b): The behavior is stable on the cyclical schedule and unstable on the transient one, in particular on the first station and buffer: idle times occur at the first station due to downstream blockages, which are mainly originated at the second and third workstations.

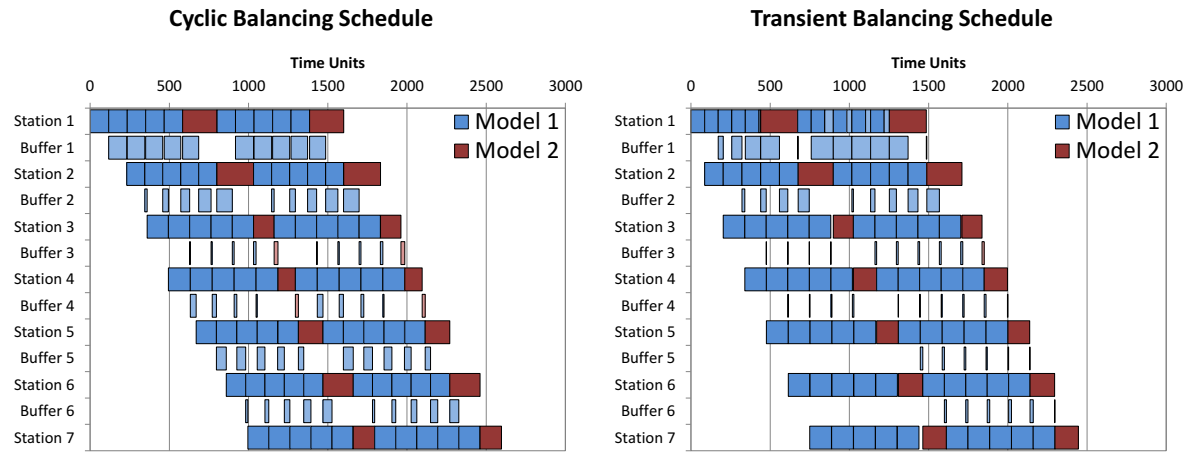
The optimization of a cyclical mixed-model flow-shop consists in combining traditional simpler problems, namely line balancing (first studied by Salveson (1955)), model sequencing

[1] The schedules illustrated by Figures 1(a) and 1(b) are generated by the optimization procedures discussed at Section 2.3.1. In these figures, darker colors represent processing times and lighter colors represent waiting times.

(first modeled by Bard *et al.* (1992)), and buffers allocation (first studied by Koenigsberg (1959)). It has been repeatedly stated that, if possible, it is best to attempt to deal with these aspects simultaneously (SAWIK, 2000; BOYSEN *et al.*, 2008; BOYSEN *et al.*, 2009c) in order to achieve better final results. Nevertheless, these problems are often dealt with independently as some reviews for mainly balancing (BATTAÏA; DOLGUI, 2013), sequencing (BOYSEN *et al.*, 2009c), buffer allocation (DEMIR *et al.*, 2014), and cyclical scheduling (LEVNER *et al.*, 2010) suggest. Most authors focus separately on variations of each of this problem's aspects. Some examples: Scholl *et al.* (2009), Gurevsky *et al.* (2012), Kellegöz (2017) present variations of balancing problems, Leu *et al.* (1997), Heath *et al.* (2013), Golle *et al.* (2015) present variations of sequencing problems, and Karabati and Kouvelis (1994), Spinellis and Papadopoulos (2000), Gurgur (2013) present variations of buffer allocation problems. Karabati and Kouvelis (1994), in particular, reinforced the previously mentioned interconnection between these problems by showing that buffer allocation based exclusively on sequence-independent information often leads to suboptimal results.

Figure 1 – Cyclical and transient schedule comparisons

(a) Cyclical schedule representing steady-state **(b) Transient schedule representing initial state**



Source: Lopes *et al.* (2016)

Nonetheless, some authors combine these problems, mostly two at a time, either using decomposition strategies or (meta)heuristic procedures (BATTINI *et al.*, 2009; ÖZCAN *et al.*, 2010; TIACCI, 2015b). In general, however, most balancing models leave buffers and product sequences out of the picture, whereas in most sequencing models as well as in most cyclical flow-shop models, processing times are considered parameters. There are, however, some noteworthy exceptions: Sawik (2004) presents a formulation for balancing and scheduling pieces in an assembly system in which stations are not necessarily serial. Öztürk (2013) presented a model

for mixed-model assembly lines that linked balancing and sequencing, on (possibly buffered) asynchronous lines with a makespan minimization goal, but did not take the cyclical aspect into account. Sawik (2012) presents formulations that compare makespan results of different scheduling strategies. Öztürk *et al.* (2015) seek to optimize steady-state results, but also use a makespan minimization goal for a series of replications of the *minimal part set* (*MPS*). The minimal part set is defined as the smallest set of pieces that can be repeated indefinitely to reach a production target: if one needs to produce 5000 units of product A and 1000 units of product B, the *MPS* is five units of product A and one unit of product B. These makespan minimization approaches (SAWIK, 2012; ÖZTÜRK *et al.*, 2015) only represent the transient-state for the first n cycles and are not guaranteed to properly predict the steady-state behavior due to the aforementioned disturbances (blockages, starvations, and empty line start). The other previously mentioned approaches (station smoothing, horizontal balancing, and vertical balancing) are indirect performance measures, and are not guaranteed to optimize cyclical steady-state either.

Thus far, mixed-model balancing models still fail to properly represent steady-state. There are many different approaches which argue their specific (and different) goal functions also lead to efficiency maximization. This might not always be the case in a flow shop cyclical schedule, either due to aforementioned transient effects (as the line starting "empty"), or due to inherent disturbances associated with asynchronous lines (blockages and starvations). Moreover, a deeper investigation of single unitary buffer might offer interesting insights on steady-state behavior and convergence, as suggested by Figure 1(b).

In this chapter, goal function formulations used on mixed-model assembly line models and previously described in the literature are discussed and compared to the authors' proposed cyclical steady-state formulation (LOPES *et al.*, 2016). Buffer allocation and a cyclical product sequence are given as parameters, and the goal is to optimize the cyclical steady-state taking such aspects into account. Tests reported herein provide evidences that these features should be considered simultaneously. The problem at hand is a *buffer-and-cyclical-product-sequence-aware* mixed-model assembly line balancing problem. The presented model can be iteratively used to compare different options of layouts and cyclical sequences (some variations are tested in the case study). The model developed by the authors is presented and a case study tests such model with data from a real-world assembly line located on the outskirts of Curitiba-PR (Brazil). Section 2.2 presents the base formulation from Lopes *et al.* (2016), along with the extensions required to compare it to previous performance measures. Section 2.3 presents the results of

applying both the proposed formulation and different goal functions to the case study's data, insights drawn from such tests are discussed. Furthermore, a new dataset is presented and tested to verify the generality of such insights. Section 2.4 summarizes the main conclusions drawn from this chapter.

2.2 MATHEMATICAL MODELS

In order to describe the *buffer-and-cyclical-product-sequence-aware* balancing problem, a mathematical model developed with mixed-integer programming is employed. The proposed model is based on the following assumptions:

1. Tasks are indivisible and performed on the same stations for all product models.
2. There are precedence relations between some tasks, reflecting technological constraints.
3. Some of them might be restricted to a subset of stations, reflecting practical constraints (i.e. the problem is assignment bounded).
4. The product sequence and buffer layout are given, and the product sequence cycles indefinitely.
5. Transportation times between stations are small and can be ignored.
6. Products flow asynchronously and sequentially in the line between stations.
7. The goal is to balance the line in a way that maximizes its efficiency, minimizing the average *steady-state* cycle time.

The developed mathematical model is a variation of mixed-model assembly line balancing models (SCHOLL, 1999). Therefore, it contains a set of basic elements, such as tasks (T), precedence relations (R), stations (S), and models (M), i.e. product's type. The mathematical model uses as parameter the product sequence of the $|P|$ (product) pieces in the global set (batch). In other words, there is a fixed sequence of products that repeats cyclically. Some practical constraints mean that not every task t can be assigned at every station s. In order to describe these constraints, the set F is employed: F lists all tuples (t,s) that represent feasible task-station allocations. Each such tuple indicates that the task t can be assigned to the station s. The set F serves two purposes: First, it allows buffers to be represented as stations to which no task

can be assigned to. Second, it allows practical features to be incorporated into the model (e.g., some tasks were fixed to stations due to heavy equipment that could not be moved). Scheduling variables state the moment products enter (Tin) and depart each station ($Tout$) as well as the processing time (Tx) of each product at each station. The proposed metric for cycle time is denoted by CT_{PX}.

2.2.1 Proposed Formulation

The developed mathematical is presented by Expressions (1) to (9). It is based on a mixed-model balancing set up with assignment restrictions imposed to the x variable set.

$$\text{Minimize} \quad CT_{PX} \tag{1}$$

$$\sum_{(t,s) \in F} x_{t,s} = 1 \qquad \forall\, t \in T \tag{2}$$

$$\sum_{(t_1,s) \in F} s \cdot x_{t_1,s} \leq \sum_{(t_2,s) \in F} s \cdot x_{t_2,s} \qquad \forall\, (t_1, t_2) \in R \tag{3}$$

$$Tx_{p,s} = \sum_{(t,s) \in F} D_{t,m_p} \cdot x_{t,s} \qquad \forall\, p \in P, s \in S \tag{4}$$

$$Tout_{p,s} \geq Tin_{p,s} + Tx_{p,s} \qquad \forall\, p \in P, s \in S \tag{5}$$

$$Tin_{p,s} = Tout_{p,s-1} \qquad \forall\, p \in P, s \in S : s > 1 \tag{6}$$

$$Tin_{p,s} \geq Tout_{p-1,s} \qquad \forall\, p \in P, s \in S : p > 1 \tag{7}$$

$$|P| \cdot CT_{PX} + Tin_{1,s} \geq Tout_{|P|,s} \qquad \forall\, s \in S \tag{8}$$

$$|P| \cdot CT_{PX} \geq \sum_{(t,s) \in F,\, p \in P} D_{t,m_p} \cdot x_{t,s} \qquad \forall\, s \in S \tag{9}$$

Equation (2) establishes the assignment bounded occurrence constraint, while Inequality (3) models the precedence relations. Notice that F must be defined in a way that if a particular station s_b is a buffer, then no task can be assigned to it. In other words, the set F contains no tuple (t, s_b) with buffer stations $s_b \in S_b$. Equation (4) determines the processing time $Tx_{p,s}$ of each piece p in each station s, by combining the information of the piece p's model m_p with that of the duration parameters $D_{t,m}$ in time units of each task t for each model m. These processing times ($Tx_{p,s}$) bind together the model's last two variable sets, namely $Tin_{p,s}$ and $Tout_{p,s}$. These variables inform the time at which each piece p enters (Tin) and leaves ($Tout$) each station s. There are three logical constraints required to properly tie these variable sets. First, a piece can

only leave a station after it's been processed (stated by Inequality (5)). Second, when a piece leaves one station it enters the next one, as stated by Inequality (6). Notice that this constraint can be easily adapted to include movement times between stations, if such times are deemed relevant. Third, a piece may only enter one station after the previous piece has left said station as stated by Inequality (7). This constraint prevents two pieces from simultaneously occupying the same station. Notice that, while Constraints (5) and (6) tie each p^{th} piece's variables to its other variables, Constraint (7) tie each p^{th} piece's variables to its predecessor, the $(p-1)^{\text{th}}$ piece. In order to represent a steady-state operation, it is necessary to also tie the last piece ($|P|$) to the first one (1). This is done by the variable CT_{PX}, which states the proposed measure of steady-state mean cycle time. This measure is bounded by throughput time for each station s, as stated by Inequality (8). This bound is the adapted version of Inequality (7): it ties the end of one set of products to the beginning of the following set: just as the piece p is tied to the previous piece $p-1$, the piece 1 ties to the last piece (of a "previous" set) $|P|$, taking into account the shop's average steady-state cycle time: CT_{PX}. This metric takes scheduling variables explicitly into account. However, ignoring such variables, it is possible to imagine stations operating at a theoretical maximum efficiency (never waiting for a piece to enter or leave it) as a bound to steady-state behavior. Under this optimistic consideration, each station can be seen as a potential bottleneck that bounds the maximum average steady-state cycle time. In other words, even though the average processing time does not define the steady-state cycle time, it does inferiorly bound it, as stated by Inequality (9). Given that this bound for cycle time does not take scheduling variables into account, it can be referred to as "LB-CT". Being unaware of product sequences and buffers, this lower bound merely defines an inferior limit to steady-state, rather than being the key to steady-state behavior.

As stated by Tiacci (2015b), realistic line features (such as mixed-models) make throughput difficult to estimate directly. This leads to multiple authors employing an indirect performance measure (THOMOPOULOS, 1970; BUKCHIN, 1998). However, the problem's true goal is to maximize productivity (throughput), or equivalently, to minimize the average steady-state cycle time. Previously described indirect performance measures proposed by other authors are presented in Section 2.2.2, their results are discussed in Section 2.3.1.

In these following sections, the proposed formulation is referred to as PX, the realized value of steady-state cycle time as CT, and the proposed measure CT_{PX}. These notation differences are needed, as an actual comparison between CT and CT_{PX} is discussed in Section 2.3.1.

2.2.2 Benchmark Formulations and Performance Measures

As discussed in Section 2.1, different authors have employed different goal functions when approaching mixed-model balancing. Two variables are defined in terms of the balancing decision variable: The processing time ($px_{m,s}$) of each model m at each station s can be determined by adding the task times of the tasks performed on it, as stated by Equation 10.

$$px_{m,s} \;=\; \sum_{(t,s)\,\in\,F} D_{t,m} \cdot x_{t,s} \qquad \forall\, m \;\in\; M, s \;\in\; S \tag{10}$$

Second, weighted average processing time (Px_s) can be determined for each station s by taking into account the demand ratio of each model, as stated by Equation 11.

$$Px_s \;=\; \sum_{m\,\in\,M} \frac{n_m}{|P|} \cdot px_{m,s} \qquad \forall\, s \;\in\; S \tag{11}$$

Last, a parameter TAx_m is defined for each model m. It states the theoretical minimum average processing of each model. Such value is calculated in accordance to Equation 12.

$$TAx_m \;=\; \sum_{t\,\in\,T} \frac{D_{t,m}}{|S|} \qquad \forall\, m \;\in\; M \tag{12}$$

The first objective to be compared to the proposed formulation is the "Station Smoothing" measure. This objective was proposed by Thomopoulos (1970) and stated by Equation 13. The goal is to minimize the fluctuation of processing times at stations. This goal function is referred to as SX.

$$\text{Minimize} \quad SX \;=\; \sum_{m\,\in\,M} n_m \cdot \sum_{s\,\in\,S\setminus S_b} |TAx_m - px_{m,s}| \tag{13}$$

Subject to: (2)-(4) and (10)-(12)

The second objective is often called "Vertical Balancing" (hereafter referred to as VX) as proposed by Merengo *et al.* (1999) and stated by Equation 14. The idea behind this goal is to establish more balanced total workloads across stations, allowing more differences between models. This goal function minimizes the differences between the weighted average processing times.

$$\text{Minimize} \quad VX \;=\; \sum_{s\,\in\,S\setminus S_b} \left(\max_{k\,\in\,S\setminus S_b} Px_k - Px_s \right) \tag{14}$$

Subject to: (2)-(4) and (10)-(12)

Vertical Balancing (Equation 14) argues that the main relevant aspect of mixed-model balancing is the weighted average processing times at stations. Its rationale is: “If the average processing times are similar across stations, no workstation will behave as an important bottleneck”. Vertical balancing does not take the differences between models within a station directly into account.

The third objective is commonly called “Horizontal Balancing” (hereafter referred to as HX). The idea behind this goal is to establish more balanced workloads across models, allowing more differences between stations. This goal function minimizes the differences between processing times of each model and the largest processing time in each station. The formulation presented by Equation 15 was proposed by Merengo *et al.* (1999).

$$\text{Minimize} \quad HX = \sum_{s \in S \setminus S_b} \frac{\sum_{m \in M} (\max_{k \in M} px_{k,s} - px_{m,s}) \cdot n_m}{|P| \cdot \max_{m \in M} px_{m,s}} \tag{15}$$

Subject to: (2)-(4) and (10)-(12)

Horizontal Balancing (Equation 15) argues that the main relevant aspect is how model’s processing times differ in each workstation. Its rationale is: “If models all have similar processing times at each station, product sequence matters less and productivity is less dependent on it”. Horizontal balancing does not take weighted average processing times directly into account.

One should note that there are other many variants of horizontal balancing and vertical balancing proposed by other authors (MATANACHAI; YANO, 2001; PASTOR *et al.*, 2002). They are, however, similar in nature to the presented ones.

The fourth alternative goal function considered is the makespan minimization variant discussed in Section 2.3, proposed by Sawik (2012), Öztürk *et al.* (2015). This approach consists on taking two replications of the *MPS* into account and minimizing the completion of the last piece, as stated by Equation 16. To the best of the authors’ knowledge, prior to this work, this was the most sophisticated balancing goal that uses scheduling information.

$$\text{Minimize} \quad MX = Tout_{|P|,|S|} \tag{16}$$

Subject to: (2)-(7)

A last performance measure is hereafter indicated, a probabilistic estimate first presented by Dar-El *et al.* (1999). It consists on computing the probability of each model being the bottleneck at each station and multiplying this probability by the processing time of said model at

said station. First, the variable β compares each model at each station, as stated by Equation 17.

$$\beta_{m_1,s_1,m_2,s_2} = \begin{cases} 1 & \text{if } px_{m_1,s_1} > px_{m_2,s_2} \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

Next, the probability of each model m_1 at station s_1 having a higher load than station s_2 is computed. Let that probability be stated by $prob1_{m_1,s_1,s_2}$, as presented by Equation 18.

$$prob1_{m_1,s_1,s_2} = \sum_{m_2 \in M} \frac{n_{m_2} \cdot \beta_{m_1,s_1,m_2,s_2}}{|P|} \tag{18}$$

The product of these probabilities in every station represents the probability of model m being the line's bottleneck at station s. That probability ($prob2_{m,s}$) is stated by Equation 19.

$$prob2_{m,s} = \prod_{s_k \in S, s_k \neq s} prob1_{m,s,s_k} \tag{19}$$

At any given time, the probability of a model being in a station can be estimated by its occupation rate. This allows one to finally define the probabilistically expected cycle time ($P.E.$) as stated by Equation 20. Notice that this formulation assumes that processing times at stations are the root source of bottlenecks. This is further discussed in Section 2.3.

$$P.E. = \sum_{s \in S} \sum_{m \in M} \frac{n_m}{|P|} \cdot prob2_{m,s} \cdot px_{m,s} \tag{20}$$

This probabilistic estimate assumes a random sequence of products (DAR-EL *et al.*, 1999), in the problem at hand, the product sequence is known a priori. Furthermore, the non-linearity present at Equation 19 makes it difficult to incorporate this formulation to the main MILP model presented in Section 2.2. Therefore, the $P.E.$ measure is compared to the experienced steady-state results of each formulation.

2.3 EXPERIMENTS AND RESULTS

This chapter's experiments are divided in two parts: The first presents a case study in which the models are applied to a data from an automobile seat assembly line on the outskirts of Curitiba-PR (Brazil). The second consists of a generality study to verify whether the results observed with practical data can be replicated in other instances. That is done with a dataset constructed out of randomly generated SALBP-1 instances. Problem data (task processing times, precedence relations, and task-assignment possibilities), along with solution tuples generated by the proposed formulation, is available at the Lopes *et al.* (2020b)'s Supplementary Material for reproducibility purposes.

2.3.1 Practical Data Case Study

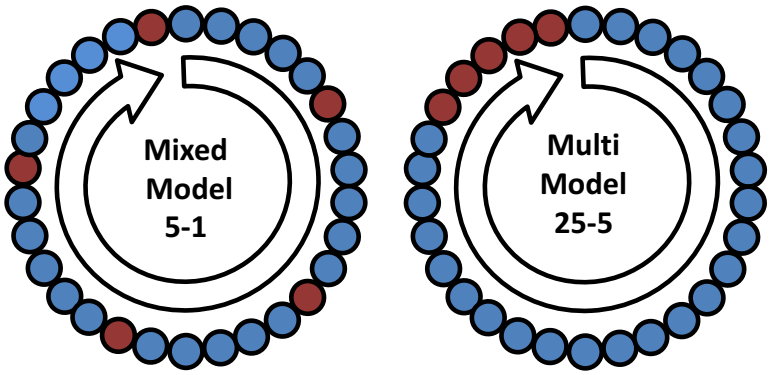
Instance Description

The case study is centered on a car seat assembly line with seven workstations that produced two product models: a simpler one and a more complex one with higher processing times in average. Demand rates are stable in a rate of 5 units of the simpler product model to 1 unit of the more complex one. Some tasks are fixed, reducing the problem's degrees of freedom (such fixed tasks are controlled by the F set). Due to internal reasons of the factory, such as a relatively constant internal demand for both products, the cyclical product sequences could be set in one of two configurations: Either the more complex product type is produced every six pieces (on a 5-1 pattern), or a batch of five units is produced every thirty pieces (on a 25-5 pattern). Intermediary configurations are regarded as confusing due to the loss of regularity and larger batches are deemed unpractical due to internal demands of the factory. The company also wanted to evaluate the impact of buffers: initially, the line did not have any internal storage. Company specialists suspected that a buffer between the second and third workstations would be very useful, but wanted to compare that it to installing buffers between every two stations.

The case study aimed at **establishing the best steady-state results as for given product sequences and buffer layouts**. By comparing multiple scenarios, one can verify the influence of sequencing and buffer allocation on solution quality. Two different product sequences with the same demand rate are considered: The first one (S1) is a "Mixed-Model" sequence, in that products alternate more frequently and, therefore, product interfaces are more relevant. The second one (S2) is a "Multi-Model" sequence, in that products alternate less often and interfaces are less relevant. Notice that despite the label "Multi-Model", set-up times between models are assumed to be negligible in both cases. Figure 2 illustrates the two cyclical sequences that are tested.

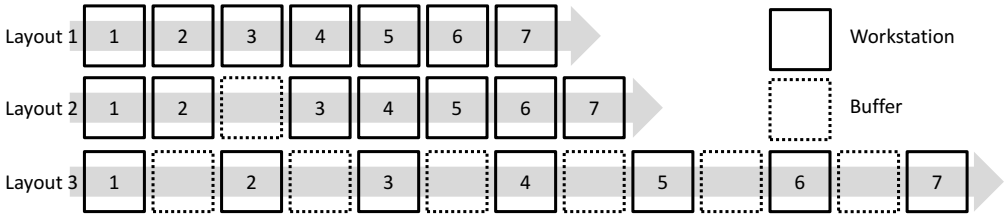
Three different buffer layouts are considered to test the influence of internal storage: The first one (L1) is a layout with no buffers between stations; The second one (L2), an intermediary layout with only one single buffer between the second and third stations; The last one (L3), a fully buffered line with one buffer between each station pair. The buffer layouts are illustrated by Figure 3. Notice that all buffers are single unitary buffers: each can only hold one piece at a time.

Figure 2 – "Mixed-Model" (S1) and "Multi-Model" (S2) sequences.



Source: Lopes *et al.* (2020b).

Figure 3 – Buffer layouts: L1, L2, and L3.



Source: Lopes *et al.* (2020b).

Proposed Formulation Results

The combination of the two sequences (S1, S2) with the three layouts (L1, L2, L3) generate six scenarios, one for each layout under each product sequence (e.g. S1L1, S2L1, S1L2, ...). For each of the generated scenarios, the proposed formulation (described in Section 2.2) is employed to generate a balancing solution that optimizes the steady-state. The optimal balancing solution generated for each scenario is applied to all other scenarios in order to verify how well solutions perform in configurations different than those they are designed to optimize. This gave rise to for a total of thirty-six cases. These are converted to model instances and solved to optimality using a universal solver (IBM ILOG CPLEX, available at *www.ibm.com*). Table 1 presents the realized values of steady-state cycle time for each one of the 36 combinations. Boldfaced values refer to steady-state behavior of each scenario with the optimal balancing obtained for that scenario. Numbers discussed in the text are highlighted in *italic* fonts.

Notice that for every scenario (line i) in Table 1, the best answer is the one obtained by the balancing designed to optimize it (column i). However, balancing solutions (column j) can behave both better and worse in scenarios other than the ones they are designed to optimize (line j): For instance, when the S1-L2 solution is applied to S1-L3, the realized cycle time (142.68) is

better than the one obtained by applying it to S1-L2 (143.87), but not as good as the one designed to S1-L3 applied to S1-L3 (133.48). Also, while the solutions with lowest steady-state cycle time also have low values of LB-CT (Inequality (9)), some cases with low LB-CT have high steady-state cycle times (for instance: S2-L3, with LB-CT of 135.48, applied to S1-L1 realized a stable CT of 168.45). This confirms an expected behavior: low values of LB-CT are **required** for low values of steady-state cycle time, but they are **not sufficient**.

Table 1 – Realized cycle time of each balancing solution applied to each scenario.

			Balancing from Sequence and Layout					
			Mixed-Model Seq (5-1): S1			Multi-Model Seq (25-5): S2		
			L1	L2	L3	L1	L2	L3
Applied to	S1	L1	**156.15**	166.33	*172.20*	*165.20*	*163.55*	*168.45*
		L2	155.28	***143.87***	152.52	155.78	155.78	152.35
		L3	153.20	*142.68*	***133.48***	140.53	140.53	135.48
	S2	L1	158.65	159.85	157.48	***149.02***	149.02	154.62
		L2	155.36	155.28	152.87	144.75	**144.75**	150.09
		L3	153.20	151.96	*146.14*	140.53	140.53	**135.48**
LB-CT			153.20	142.68	133.48	140.53	140.53	*135.48*

Solutions designed for buffered layouts tend to behave poorly when applied to un-buffered layouts: S1-L3 has the best steady-state behavior (133.48), however, when its balancing is applied to S1-L1, it displays the worst steady-state behavior (172.20). Furthermore, Table 1 shows that buffer layouts can influence the optimal sequencing decisions: For L1, the sequence that offers the best steady-state behavior is S2 (149.02), but for L2 and L3, the sequence S1 outperforms S2 (143.87 and 133.48).

Cases S2-L1 and S2-L2, in particular, generated very similar answers: When applied to all scenarios other than S1-L1 (in which S2-L2 (163.55) outperformed S2-L1 (165.2)), both generated equal answers. This demonstrates, that for some scenarios, there might exist **multiple optimal** balancing answers: two different balancing solutions provided the same steady-state cycle time for five out of six scenarios.

Table 2 presents the (model-station) processing times of all six solutions. Notice that the best solutions for the fully buffered cases (S1-L3 and S2-L3) display steady-state cycle times lower than the highest station-wise processing times of individual models. These cases also have very similar cycle times despite the different cyclical sequences: it may also be unintuitive how five pieces of model M2 (with much higher processing times in some stations than M1) can go through the line (Sequence S2) with only a minor impact on steady-state cycle time: How can single buffers compensate those differences? In order to further clarify how this occurs,

steady-state schedules are presented for thirty pieces of both the S1-L3 case (Figure 4(a)) and the S2-L3 case (Figure 4(b)).

Table 2 – Processing times for each model, at each station, for each layout and each cyclical sequence.

	Mixed-Model Sequence - S1								
	Layout 1			Layout 2			Layout 3		
Station	M_1	M_2	LB-CT	M_1	M_2	LB-CT	M_1	M_2	LB-CT
1	121.0	231.7	139.45	105.1	187.3	118.80	116.3	218.1	133.27
2	110.5	216.0	128.08	104.6	**255.5**	129.75	113.6	**231.8**	133.30
3	121.9	108.0	119.58	136.6	139.5	137.08	134.7	127.0	133.42
4	126.7	120.9	125.73	130.9	141.1	132.60	**138.2**	109.6	133.43
5	129.2	127.7	128.95	136.9	137.7	137.03	128.6	155.8	133.13
6	**139.8**	130.6	138.27	135.0	136.9	135.32	122.0	190.9	**133.48**
7	137.1	**233.7**	**153.20**	**137.1**	170.6	**142.68**	132.8	135.4	133.23

	Multi-Model Sequence - S2								
	Layout 1			Layout 2			Layout 3		
Station	M_1	M_2	LB-CT	M_1	M_2	LB-CT	M_1	M_2	LB-CT
1	123.5	**217.5**	139.17	123.5	**217.5**	139.17	117.7	**222.9**	135.23
2	125.3	216.0	140.42	125.3	216.0	140.42	118.8	216.0	135.00
3	126.3	175.7	134.53	126.3	165.8	132.88	123.7	127.6	124.35
4	127.7	77.3	119.30	127.7	87.2	120.95	126.4	169.6	133.60
5	**128.0**	144.0	130.67	**128.0**	144.0	130.67	132.9	141.7	134.37
6	127.8	132.9	128.65	127.8	132.9	128.65	**135.7**	132.9	135.23
7	127.6	205.2	**140.53**	127.6	205.2	**140.53**	131.0	157.9	**135.48**

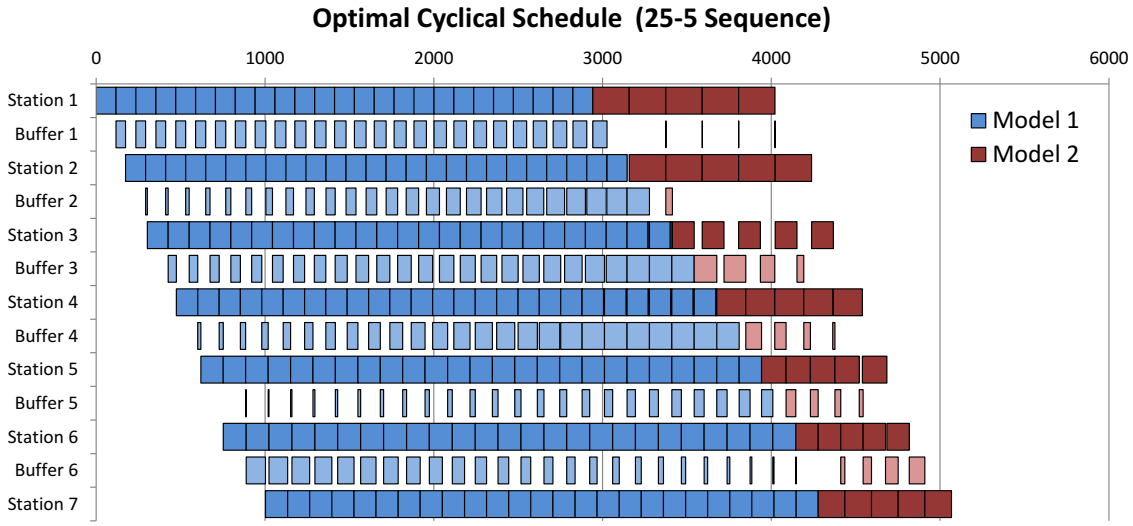
Notice that in Figure 4(a) the buffer utilization changes as the minimal part set goes through the line. Buffer 1 fills after the first piece is processed at Station 1 and stays full until the model two piece arrives. The second buffer is used initially for a small time, but this duration increases as the *MPS* passes through the line. Naturally, every time a full *MPS* passes through the system, the occupation returns to the initial configuration. This is required for the schedule to be representative of the steady-state behavior.

Figure 4(b) further clarifies how buffers reduce blockages and starvations caused by model differences (as shown by Table 2): buffers also display an **evolving pattern** of relative utilization. The main difference to the S1-L3 scenario is how incremental these changes are: the first five buffers in the S2-L3 case slowly build up the relative utilization as M1 pieces pass through the line. Buffer 6, however, displays the opposite behavior: the buffer decreases relative utilization as M1 pieces are produced and increases it as M2 pieces are produced.

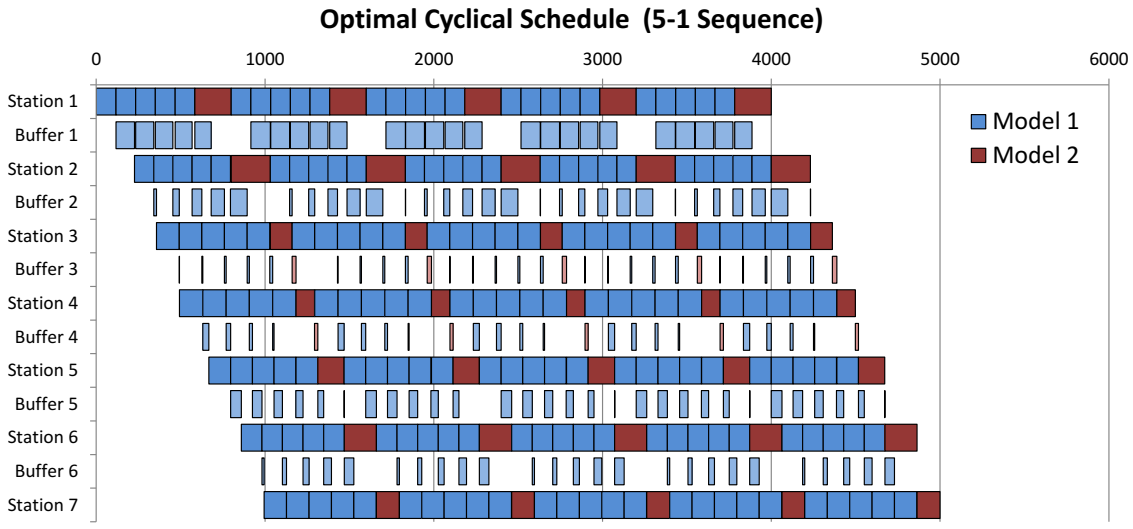
Notice that the third station on the S2-L3 case has significantly longer idle times than any station in the S1-L3 case. This is expected as its average processing time (124.35, indicated in Table 2) is much lower than the cycle time (135.48, indicated in Table 1), one could try to see

this as result of bad balancing. However, the fact that this is an optimal steady-state schedule for this scenario means that no other balancing can produce less unproductive times. For example, applying S1-L3 (that has a LB-CT of 133.48, indicated in Table 1) to S2-L3 leads to a steady-state cycle time of 146.14.

Figure 4 – Comparison between cyclical schedules 30 products
(a) Steady-state schedule under the "Multi-Model" cyclical sequence.



(b) Steady-state schedule under the "Mixed-Model" cyclical sequence.



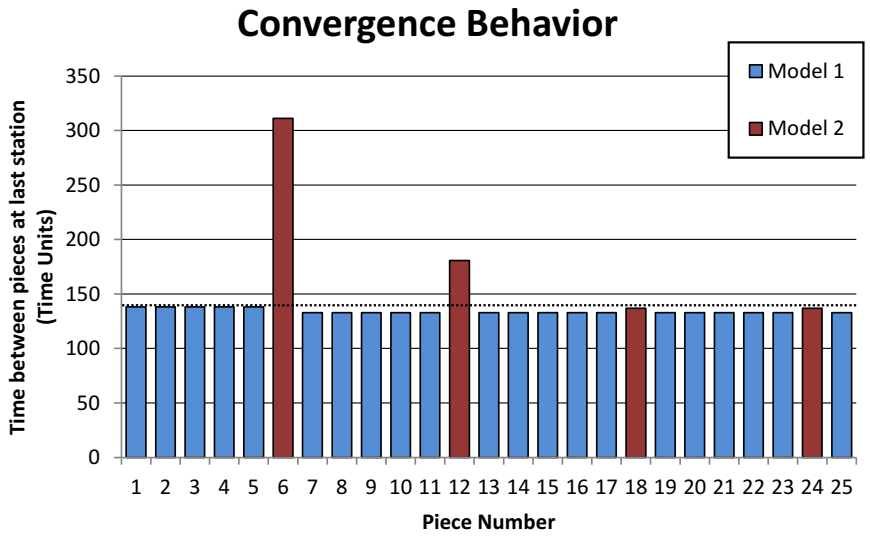
Source: Lopes *et al.* (2020b)

Lastly, by running the presented model, with Inequality (9), but without Inequality (8), a global value of LB-CT is achieved. This bound is 133.48 time units, which matched the value of the S1-L3 (Table 2) case. This means that a **finite number of buffers** allowed the system to reach the **maximum theoretical productivity**.

Result Validation via Simulation

The optimal steady-state cycle times predicted by the proposed model are validated using a simulation software (Simio, available at *www.simio.com*). The product sequence and processing times are deterministic and, therefore, the results are deterministic. As a result, a detailed statistical analysis of the data is not required, contrary to stochastic cases (ALEXANDER *et al.*, 2010). The predicted values for average steady-state cycle time matched perfectly with the steady-state conditions observed in the simulation (i.e. $CT = CT_{PX}$). Figure 5 shows the convergence behavior of the simulation for the S1-L3 case, reproducible with the input data shown in Table 2. Notice that model 2 takes three cycles to achieve steady-state, while model 1 takes only two. Model 1's initial cycle (138.2) time is not only higher than its final one (132.8) but also slightly higher than model 2's final one (136.9). Figure 5 demonstrates that the time between pieces at the last station (and, therefore, the cycle time) **gradually converges** towards steady-state (and, therefore, towards the steady-state cycle time). The simulation allows the verification of an interesting behavior: the initial cycle times of both models (138.2, and 311.1) are higher than the one verified at the steady-state for both models (132.8, and 136.9).

Figure 5 – Simulated convergence behavior for scenario S1-L3.



Source: Lopes *et al.* (2016).

These simulations confirmed that the difference between the realized cycle time and the value of LB-CT can be explained by the blockages and starvation disturbances, which happen cyclically due to the cyclical product sequence. Naturally, the simulation model required a warm-up period as the initial emptiness of the line meant that the first pieces took longer to go

through the system, as shown by Figure 5. Furthermore, as the first pieces passed, buffers are not yet able to compensate differences between models as well as they do in steady-state. This meant that the first couple of replications of the *MPS* are not necessarily representative of the steady-state, and that one cannot always measure model-wise steady-state cycle time based on the behavior of such early replications. Steady-state behavior (in this case time between pieces output) can be "better" than transient-state behavior, despite the naturally expected (and verified) higher *time in system* of later pieces. This "**empty line effect**" is a disturbance associated with the initial transient-state and is the key to explain the convergence behavior.

Comparisons to Other Formulations

In order to verify how well the different formulations behave, each goal function is implemented and their answers tested for steady-state behavior. Tests are performed for each buffer layout and each cyclical product sequence presented in the main case study described herein-above. These tests are subdivided in three groups: The first one refer to **scheduling unaware** literature formulations. The second, refer to the makespan minimization (**scheduling aware**) formulation described in the literature. The last group evaluates how well the literature described **probabilistic estimate** of cycle time behaves in regard to the **experienced cycle time** of the tested cases.

Scheduling Unaware Formulations

Out of the alternative goal functions presented in Section 2.2.2, three do not employ scheduling variables (e.g., $Tin_{p,s}$ or $Tout_{p,s}$), namely Station Smoothing (SX), Horizontal Balancing (HX) and Vertical Balancing (VX).

Three composed variants are also tested: First, $VX + SX$ is the goal function that originates from adding VX to SX. Second, $HX + CE$ employs the same goal function as HX and the additional constraint stated by Inequality 21, binding the processing time value of each model at each station to an upper bound value[2]. Last, $HX + CA$ is similar to $HX + CE$ but employs the additional constraint stated by Inequality 22, binding only the weighted average

[2] The value 250 time units is employed based on the optimal processing time from answers obtained by other formulations

processing time by an upper bound value[3].

$$px_{m,s} \leq MaxProcTime \qquad \forall\, s \in S, m \in M \tag{21}$$

$$Px_s \leq MaxWProcTime \qquad \forall\, s \in S \tag{22}$$

The main difficulty presented by Equation 15 is the non-linearity associated with the division by the variable $px_{m,s}$. In the case studies, an iterative process is employed: A trial solution is used, and the values of $\max_{m \in M} px_{m,s}$ are set as parameters for the next execution. Each variable $px_{m,s}$ is then limited by the value of $(\max_{m \in M} px_{m,s})$ multiplied by $(1+k)$. Where k is a parameter whose starting value is 0.2 and decreased (by 0.01) over the iterations. This led to solutions with progressively lower values of HX. Once $k = 0$ is reached, HX reached a local minimum value. This procedure is not guaranteed to generate optimal solutions in regard to HX, but did generate better results (lower values of HX) than the initial trial solution, justifying its use.

Table 3 – Realized cycle times obtained for each goal function.

Seq.	Layout	SX	VX	HX	SX+VX	HX+CE	HX+CA	MX	PX
S1	L1	167.28	172.20	364.47	170.23	171.68	177.62	156.73	156.15
	L2	154.70	154.55	364.47	154.70	171.68	158.37	143.87	143.87
	L3	145.00	134.57	364.47	141.33	170.27	158.37	138.93	133.48
S2	L1	154.35	159.55	364.47	151.42	179.83	187.48	149.02	149.02
	L2	149.95	154.94	364.47	147.01	174.15	182.38	144.75	144.75
	L3	146.31	148.41	364.47	142.64	170.27	170.79	136.38	135.48

Table 3 compares the observed steady-state cycle time of each formulation to those obtained by the proposed formulation and those obtained by the makespan minimization formulation, for a total of 48 cases. By comparing the base formulations (SX, VX and HX) it is rather clear that horizontal balancing is out-performed by both the station smoothing and the vertical balancing approaches. In the buffered cases (Layout 3 in particular), VX reached solutions comparable to the much more sophisticated scheduling aware formulations (MX and PX): VX applied to S1-L3 obtained a result (134.57) between PX's one (133.48) and MX's one (138.93).

Table 4 presents the processing times obtained on optimal solutions (or local minimum in HX's case). Notice how HX presents an interesting problem: Some stations have average

[3] The value 160 time units is employed based on observed cycle time values of other formulations

Table 4 – Model-station processing times of solutions obtained according to each formulation.

	Station Smoothing - SX			Vertical Balancing - VX			Horizontal Balancing - HX		
Station	M_1	M_2	LB-CT	M_1	M_2	LB-CT	M_1	M_2	LB-CT
1	126.0	184.6	135.77	119.9	200.4	133.32	95.2	95.2	95.2
2	126.4	**238.0**	**145.00**	110.5	**247.3**	133.3	**317.4**	**599.8**	**364.47**
3	**132.1**	160.9	136.90	134.2	129.2	133.37	41.9	42.1	41.93
4	123.7	129.2	124.62	**138.2**	109.6	133.43	29.4	29.3	29.38
5	126.6	146.3	129.88	128.6	155.8	133.13	94.9	94.7	94.87
6	123.8	144.7	127.28	122	190.9	**133.48**	150.1	149.6	150.02
7	127.6	164.9	133.82	132.8	135.4	133.23	157.3	157.9	157.4

	Composed - SX+VX			Composed - HX+CE			Composed - HX+CA		
Station	M_1	M_2	LB-CT	M_1	M_2	LB-CT	M_1	M_2	LB-CT
1	126.0	206.6	139.43	122.1	229	139.92	105.7	102.9	105.23
2	126.4	**216.0**	**141.33**	98.8	**248.1**	123.68	115.2	**374.2**	158.37
3	**132.1**	160.9	136.90	83.5	83.6	83.52	83.5	83.6	83.52
4	123.5	128.6	124.35	89.3	89.3	89.3	134.3	136	134.58
5	126.2	153.9	130.82	169.2	169.3	169.22	**151.6**	152.6	151.77
6	124.4	155.4	129.57	153.1	178.7	157.37	142.8	142.7	142.78
7	127.6	147.2	130.87	**170.2**	170.6	**170.27**	153.1	176.6	**157.02**

processing times substantially higher (over five times) than others. This is due to the fact that the goal function only measures the difference between models. HX **concentrates** processing times **differences** in a single station, which is the most loaded one. This is not a coincidence: Analyzing Equation 15 one can notice that differences between processing times count less if they occur in loaded stations. This means HX tends to intentionally load a station with very high processing time in one model and concentrate the differences between models as much as possible in there: Indeed, aside from that station, processing times are very similar across models. Vertical balancing (VX) presents solutions with better potential as the **average processing times** are **lower**, but the realization of said potential depends heavily on scheduling, product sequence, and buffer layout. In scenarios without buffers, VX produced steady-state behaviors very distant from optimal behavior. Station smoothing (SX) presents model-wise processing times that are roughly **stable across stations**, but worse values of LB-CT than VX. The SX solution realizes better values of cycle times than VX in four out of six cases. However, in the S1-L3 case, the better processing time distribution of VX outperforms SX substantially (134.57 versus 145).

In most cases the presented composed variants ($SX + VX$, $HX + CE$ and $HX + CA$) did outperform the base variants. However, the reason for the improvement in each case ought to be further discussed. The main problem with HX is its tendency to accumulate high processing time in one station. This means that HX is less capable of accumulating processing time in one

station when combined to either CE or CA. The $VX+SX$ combination, on the other hand, only outperforms both SX and VX in the S2 cases. In four cases $VX + SX$ outperforms SX and in other four it outperforms VX. In that sense, $VX + SX$ seems to combine the formulations strengths. One should note, however, that this is not enough to optimize steady-state as both MX and PX produce better steady-state results (than $VX + SX$) in all scenarios: this indicates that, in some cases **neither low average** processing times **nor smoothed processing times** across stations **are guaranteed to lead**, on their own, **to steady-state efficiency**.

In all scenarios, the **scheduling unaware** formulations are **dominated by *PX***, indicating a weakness of these formulations: Not being able to explicitly take into account the product sequence and buffers.

Goal Function Comparison and MPS Replications

As discussed in Sections 2.1 and 2.2, makespan minimization (SAWIK, 2012; ÖZTÜRK *et al.*, 2015) for some *MPSs* is not necessarily the same as steady-state optimization. This is confirmed by Table 3, which shows that for three cases (S1-L1, S1-L3, and S2-L3) the realized cyclical behavior is worse when applying the makespan minimization goal (for two *MPS* replications) than when applying the proposed formulation. It is expected, however, that as more replications are taken into account these two goals are likely to **converge** towards the same results. In order to test that hypothesis, the scenario with most significant differences between the results of each formulation (S1-L3) is tested for up to forty *MPS* replications. The results are summarized by Table 5.

Table 5 shows the realized values of both steady-state cycle time (CT) and makespan (MX), when minimizing both PX and MX. For one replication, 1611.7 is the minimal makespan. PX obtains 1752.6, meaning the initial behavior is better when minimizing MX. However, the realized values of CT when minimizing makespan are higher than those obtained by the proposed formulation (144.28 versus 133.48), meaning the stable behavior is worse when minimizing MX.

These tests show that the proposed formulation produces better steady-state results, but longer makespan for the n first replications. However, as the number n of considered replications is increased, the difference in both makespan and steady-state behavior between formulations decrease. Notice, however, that **the proposed formulation achieves the best steady-state behavior without the need to consider multiple replications** of the piece set.

This confirms the previously mentioned hypothesis, but does not explain the results on its own: Why do makespan minimization formulations produce different results for different numbers of replications?

Table 5 – Comparison between proposed formulation and makespan minimization for S1-L3 scenario

Goal:	Makespan Minimization (MX)			Proposed Formulation (PX)		
Values:	MX	CT	CPU Time	MX	CT	CPU Time
No. Rep.						
1	1611.7	144.28	5”	1752.6	133.48	6”
2	2445.3	138.93	6”	2596.8	133.48	8”
3	3277.7	137.37	7”	3397.7	133.48	9”
4	4101.1	137.07	9”	4198.6	133.48	9”
5	4923.0	136.62	12”	4999.5	133.48	11”
10	8954.1	133.62	30”	9002.0	133.48	18”
20	16971	133.62	2’54”	17011	133.48	48”
40	33005	133.62	7’23”	33029	133.48	1’20”

Figure 1(a) and Figure 1(b) in Section 2.1 offer further insights as they present the cyclical schedules for optimal answers of the proposed formulation and the makespan minimization formulation, respectively. As expected, the makespan is smaller on Figure 1(b), but the steady-state result is worse (about 3% higher average cycle time, as presented by Table 5). Notice how the replications of the *MPS* do not display the same behavior if the makespan minimization formulation is employed.

The probable explanation for this behavior difference lies in the transient-state disturbances, or “empty line effect” (also verified in the simulation model as the key to the convergence behavior): just as in simulation models, the line is started “empty”, and this can be exploited by the makespan minimization goal. This gain might, however, come at a cost for the steady-state behavior: makespan minimization generates a dispute between exploiting the “empty line effect” and optimizing how well the pieces flow through the line when such effect does not exist. This is also concluded by Öztürk *et al.* (2015), who discussed the advantages of considering multiple product replications. The proposed formulation (PX), however, is **insensitive** to the empty line effect, and achieves steady-state optimization without having to subject itself to multiple replications of the piece set.

In industrial practical cases, the number of pieces to be produced might be on the order of thousands (5000 pieces of model 1, 1000 pieces of model 2). If that is the case, not only the behavior of the initial pieces is negligible, but it might also be unpractical to optimize makespan for so many pieces simply due the large size of the problem, measured in number of variables

and constraints: As the number of replications increased, so did the required CPU time to solve both models (see Table 5). In that case the best goal is clearly to optimize cyclical behavior, which can be performed by the proposed model with one *MPS* replication, regardless of the total number of pieces to be produced. Notice that, as shown by Table 5, even with forty *MPS* replications the makespan minimization formulation does not reach the optimal steady-state behavior. This reinforces the contribution of the proposed formulation.

If the number of produced pieces is smaller, one could argue that it's interesting to take advantage over the "empty line effect" to reduce the makespan. However, this effect can often be unreliable: Factories usually continue in one day the work with pieces from the previous day and, therefore, pieces that are already in the line make the "empty line effect" (observed in both the simulation model and in the makespan minimization model) inexistent as the previous day's pieces tend to slow down the first replications of the current workpieces. If that is the case, even for smaller total number of produced pieces it might be best to optimize steady-state behavior instead of the makespan, even if one could take the total number of replications into account.

Probabilistic Estimate

This section discusses the probabilistic estimate of cycle time $P.E.$ introduced by Dar-El *et al.* (1999). The values of $P.E.$ are calculated for each solution of the proposed method and also of those obtained by the scheduling unaware formulations. Then, P.E. values are compared to the realized cycle time those solutions offered. Table 1 and Table 3 present the values of six steady-state results for each of the 12 solutions. Such minimum and maximum values of realized cycle time are compared to the probabilistic estimate of that each solution in Table 6, which summarizes results from 72 cases.

Table 6 – Comparison between highest and lowest realized CT and corresponding probabilistic estimate.

Solution	Min	Max	P.E.	Solution (PX)	Min	Max	P.E.
SX	145.00	**167.28**	165.60	S1-L1	153.20	158.65	177.44
VX	134.57	172.20	172.50	S1-L2	142.68	166.33	170.20
HX	364.47	364.47	364.47	S1-L3	133.48	**172.20**	172.07
SX+VX	141.33	**170.23**	164.77	S2-L1	140.53	165.20	167.00
HX+CE	170.27	179.83	192.37	S2-L2	140.53	163.55	168.67
HX+CA	158.37	187.48	193.21	S2-L3	135.48	**168.45**	158.20

Notice how, for almost every case, the estimate is conservative: higher than the realized cycle time. This could lead one to suppose that it can be used as an upper bound for steady-state

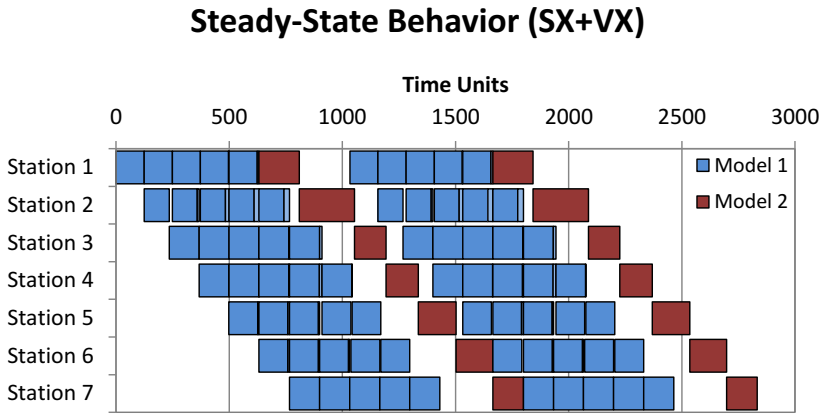
solution quality. However, there are problems with that line argument: First, this estimate is based on random product sequence; Second, the measure is insensitive to buffers capacity of assisting scheduling and, therefore, the correspondence between realized cycle time and the probabilistic estimate is not high. Lastly, in some cases ($SX + VX$, S1-L1, for instance) the estimate is better than the realized result. All these "pathological results" occurred for the same scenario: S1-L1. In order to clarify why this happened, one of these "pathological" cases ($SX + VX$) is compared to a "healthy" one (S1-L1) in, respectively, Figure 6(a) and Figure 6(b). Some insights on the reasons why this occurred: In Figure 6(a), the bottleneck station is the sixth one, even though the processing times on both models are not large. The station is a bottleneck due to starvations that are **propagated** and generate idle times between pieces, this can be seen as a scheduling bottleneck: Almost ironically, the sixth station blocks the previous stations because it is starved by them. In Figure 6(b), the differences between processing times absorb the blockages and prevent them from propagating idle times. Notice that the replications of the piece set are closer to each other in Figure 6(b) than in Figure 6(a): Blockages propagate through stations in Figure 6(a) and generate worse steady-state behavior.

This implies on a rather unintuitive conclusion: In some specific cases, differences between models is a **desirable feature**. Naturally, the benefit is not the difference itself, but rather what it allows when scheduling comes into play. This is naturally easier when buffers are added. Furthermore, Figure 6(a) offers a clear example on how and why a station without high processing times can be the system's bottleneck.

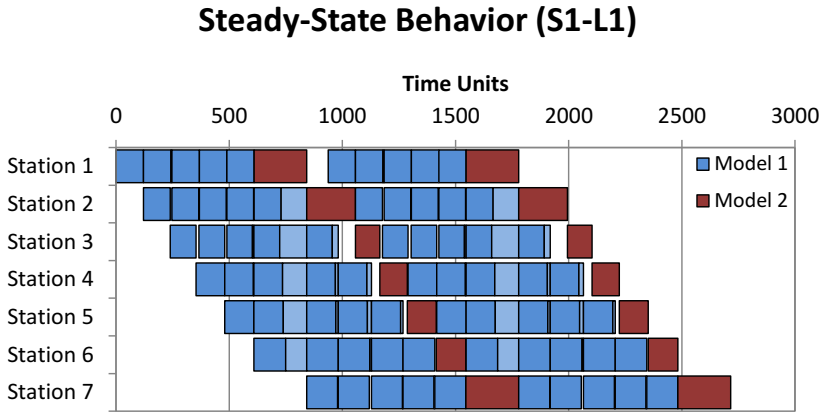
2.3.2 Generality Case Study

The results described in Section 2.3.1 refer to variations of a single instance: One set of data vectors describing tasks, models, and demand rates is tested with two different product sequences and three buffer layouts. It is necessary to verify whether or not these results hold in general. Furthermore, task-balancing often occurs before model sequencing (BOYSEN *et al.*, 2009b) contrary to the case study, in which the product sequence is known as a parameter. Therefore, if the procedure described by Boysen *et al.* (2009b) is adopted, product sequences are optimized for a certain balancing solution. This poses the question: Can the proposed formulation lead to production rate improvements for a product sequence that is itself optimized for a previous balancing solution? Given the relative proximity between the results obtained by the proposed formulation (PX) and the makespan minimization ones (MX), one can also ask: How do these

Figure 6 – Comparison between cyclical schedules generated by different formulations
(a) $VX + SX$ formulation: full schedule repeats every 1021.4 time units.



(b) PX formulation: full schedule repeats every 940.2 time units.



Source: Lopes *et al.* (2020b)

sequence-aware formulations behave in other instances?

Data Generation

In order to answer the questions herein above mentioned and to verify the generality of the results presented in the previous sections, new tests are performed on a larger dataset. The SALBP instances from the dataset presented by Otto *et al.* (2013) are used to generate mixed-model ones, as described hereafter: The SALBP instances are designed for the type-1 variation of the problem, with a cycle time of 1000 time units. The instances vary in size, task time distributions, precedence graph structure, and ordering strength. Otto *et al.* (2013) present three types of task time distributions: peak in the bottom, peak in the middle, and bimodal. The first two generate instances with a normal distribution of task times centered on either a low or medium value (relative to 1000 time units). The later generates task times that are most likely

to be small, but have a small probability of being much larger. This type of task distribution is chosen to generate mixed-model instances due to the following reason: If the task times of multiple SALBP instances are converted into task times of different product models, most tasks will have similar small processing times, and the tasks that have higher processing times will differ between models. This emulates a rather interesting mixed-model characteristic.

In order to restrict the comparisons to optimal solutions of each formulation, only the small (all of which had 20 tasks) cases from Otto *et al.* (2013) are used. No filter is applied to the structure of the precedence diagrams or to its ordering strength. A total 175 bimodal and small SALBP instances are used. These are grouped five-by-five sequentially, generating 35 groups of task processing times. For each group, five task-properties data vectors are generated: one for the precedence diagram of each of the SALBP instances. For all instances, the demand rates are presumed to be equal, 20% for each product, with an *MPS* of $(1,1,1,1,1)$. This leads to a total of 175 mixed-model data vectors.

In order to optimize the production rate, the number of stations must be given. A specific value is chosen (seven workstations), based on the main case study. Three buffer layouts are considered: *Empty*, with no buffers; *Half*, with single unitary buffers before every even station (3 buffers in total); and *Full*, with single unitary buffers between every station (six buffers in total). This leads to a total of 525 individual *buffered mixed-model instances*.

The cyclical product sequence could be arbitrarily chosen. But one goal of this dataset is to verify if the proposed formulation can lead to production rate improvements for a product sequence that is itself optimized for a previous balancing solution. Therefore, for each *buffered mixed-model instance* two specific cyclical sequences are produced: the ones that lead to the best production rates for the SX and VX formulations. This generates a total of 1050 *sequence-aware instances* for the formulations MX and PX. Lopes *et al.* (2020b)'s Supplementary Material provides additional details about the data generation.

Generality Study Results

Each of the 525 *buffered mixed-model instances* is solved for two sequence unaware goal functions (SX, VX)[4], generating an optimal balancing solution according to said goal. The best possible cyclical sequence is then defined for the balancing solution, this can be done rather

[4] Horizontal balancing HX is not tested due to its poor performance on the practical case study and to its non-linearities that prevent optimal solutions from being found.

quickly by comparing the 24 possible alternatives[5]. This sequence solution (the best possible one for either SX or VX) is then used to generate a sequence-aware instance that is solved by both proposed model (PX) and the makespan minimization model (MX). The cycle time obtained by each formulation (PX, MX, SX, VX) can then be compared to every other formulation. These results are summarized by Table 7.

Table 7 – Generality study results: comparison to benchmarks.

	Buffers	SX	VX	MX	PX
Avg. Ratio to PX	Empty	1.047	1.095	1.032	1
	Half	1.06	1.069	1.048	1
	Full	1.098	1.002	1.066	1
Max. Ratio to PX	All Layouts	1.22	1.25	1.18	1
Min. Ratio to PX	All Layouts	1	1	1	1
Average CT	Empty	830.16	866.13	817.15	791.97
	Half	779.98	785.49	770.85	735.55
	Full	759.8	691.86	736.85	691.08
No. CT=LB-CT	All Layouts	1	151	11	263

Table 7 presents the average values of cycle time for each formulation (SX, VX, MX, PX) and the (minimum, average, and maximum) ratios between those values and the CT value achieved by the proposed formulation (PX). Notice that the global minimum value for all cases is 1, meaning that *the proposed formulation is not outperformed in any instance*. These results showed that the other methods behave differently depending on the buffer layouts: the SX goal leads to better results than the VX goal in the *Empty* and *Half* buffer layouts, but is outperformed by VX in the *Full* Buffer layout. In this layout, VX is very close to PX, with a 0.2% average difference in realized cycle times, meaning that this is a good alternative when assembly lines have many buffers (e.g. in the Electronic Industry). However, it generated in average the worst results for the other layouts. The makespan minimization alternative (MX) generated good results (second only to PX) for the buffer layouts *Empty* and *Half*, but showed a large difference to the proposed formulation in the *Full* buffered scenario. This mimics the results obtained on Section 2.3.1, suggesting that when **more buffers** are present there is a **stronger dispute between** the optimization of **steady-state** behavior and **transient-state** behavior. These results also corroborate the observed interconnections between balancing and buffer allocation. Furthermore, a comparison between the realized values of cycle time and the minimal theoretical

[5] In an *MPS* of five units (1,1,1,1,1), by fixating the first model in the first position, one can generate $4!$ combinations of cyclical sequences. Each sequence requires a quick simulation of a few replications of the *MPS* going through the line. Alternatively, linear relaxations of the proposed model can be used.

one (LB-CT) shows that, in 263 cases, a finite number of buffers allows the system to reach the maximum theoretical productivity.

The Probabilistic Estimate (P.E.) discussed in Section 2.2.2 is also compared to the realized cycle time of each formulation (SX, VX, MX, PX). The minimum, average and maximum ratios are presented by Table 8. Notice that the estimate repeats the conservative overall behavior that had been previously verified: In average for all methods the estimate is higher than the realized cycle time, and remains insensitive to buffers capacity of assisting scheduling. For some instances, however, the P.E. is up to 6% optimistic, repeating the inconsistencies that had been previously observed in Section 2.3.1.

Table 8 – Generality study results: ratios to probabilistic estimate

Buffer	Empty			Half			Full			All Layouts		
Method	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.	Max.	Avg.	Min.
SX	1.16	1.07	**0.94**	1.23	1.14	1.03	1.38	1.17	1.03	1.38	1.12	**0.94**
VX	1.24	1.14	**0.98**	1.39	1.25	1.14	1.63	1.42	1.24	1.63	1.27	**0.98**
MX	1.31	1.17	1.06	1.42	1.24	1.1	1.52	1.31	1.13	1.52	1.24	1.06
PX	1.4	1.2	1.07	1.48	1.3	1.14	1.65	1.41	1.26	1.65	1.3	1.07

A detailed table with all output information of each of the 1050 individual executions (525 for SX, 525 for VX, each followed by MX and PX) is available at the Lopes *et al.* (2020b)'s Supplementary Material, along with the data vectors. The results reported above summarize that information, answering the aforementioned generality questions: What is **observed** in the practical case study is **verified** anew in the combinatorial instances. In the majority of cases (82%), **it is possible to improve** the productivity of the line by using a sequence aware formulation (by in average 5%), **even when that sequence is optimal for a previous balancing solution**. This reinforces the main contribution of this chapter: although sequencing is usually performed after balancing, when a cyclical sequence is known, the presented balancing formulation (PX) can offer better steady-state production rates.

2.4 CONCLUSIONS

This chapter presents a model for balancing an asynchronous assembly line with given buffers and cyclical product sequence is presented. The model is applied to data from a car seat assembly line on the outskirts of Curitiba-PR (Brazil) in a practical case study. The proposed formulation accurately predicts steady-state configurations with only one replication of the product set, and is validated by a simulation model. The optimized result is aware of the buffer

layout and product sequence (problem's scenario). Answers obtained with each tested scenario are applied to all other scenarios, allowing one to state that a triple inter-dependency of optimal solutions exists connecting balancing, sequencing and buffer allocation (Table 1).

Furthermore, comparisons between scenarios (Figure 4(a) and Figure 4(b)) allowed to verify that buffers can aid scheduling mixed-model assembly lines whether product models change often or not: The relative utilization of buffers can slowly increase or decrease as rather large minimal part sets (up to 30 pieces tested) go through the line.

The steady-state results achieved by the proposed formulation are compared to those reached by previous literature described goal functions: station smoothing, vertical balancing, horizontal balancing, and makespan minimization. Variants of previous formulations are also tested. The case study has shown that optimizing a flow-shop with makespan minimization goal for a finite number of replications is not equivalent to optimizing for steady-state efficiency (Table 5 and Figure 1(b)). Moreover, the makespan minimization formulation's answer converges towards the proposed formulation's answer as the number of considered replications increases.

A probabilistic estimate of cycle time (DAR-EL *et al.*, 1999) is tested, showing conservative values for most scenarios. In some scenarios, however, the estimate is optimistic. A closer investigation on the solution provided in such scenarios suggested (counter-intuitively) that differences between models can be a desirable feature, by helping to compensate via scheduling other differences between models on other stations. This case showed that the probabilistic estimate is unable to take into account scheduling-generated bottlenecks: Stations can propagate the differences between models and lead to steady-state bottlenecks in stations without high processing times on the individual models (Figure 6(a)).

A combinatorial study with 1050 individual instances is conducted. The optimal answers of each instance in accordance to each formulation is analyzed, and compared to the probabilistic estimate. With these additional tests, the results verified on the practical case study are reinforced: First, balancing, sequencing and buffer allocation problems are interconnected. Second, makespan minimization of a finite number of replications does not necessarily optimize steady-state. Third, a finite number of buffers might allow an assembly line to reach maximum theoretical efficiency. Last, the proposed formulation generated solutions with steady-state cycle times in average 5% better than alternative formulations (station smoothing, vertical balancing and makespan minimization), and it is not outperformed in a single any instance.

3 INCORPORATING SEQUENCING AND LINE CONTROL

This chapter consists of an adapted version of an article published in the International Journal of Production Economics (LOPES *et al.*, 2018). In it, the formulation presented in Chapter 2 is extended to incorporate mixed-model sequencing decisions, as well as the distinction between synchronous and asynchronous product transfers.

3.1 CONTEXT

Assembly lines are product oriented layouts commonly used in industry for large-scale production of similar product models (SCHOLL, 1999). These lines are associated to many optimization problems. Some have been studied for a long time such as workload balancing (SALVESON, 1955), model sequencing and scheduling (THOMOPOULOS, 1967). Others were more recently introduced such as material supply scheduling (STERNATZ, 2014).

There are various types of assembly lines, which can be classified according to multiple criteria (BATTAÏA; DOLGUI, 2013), such as product diversity. This chapter focuses on mixed-model assembly lines, in which products flow through the line without set-up times required between models. These lines can be further classified in terms of line pace (BOYSEN *et al.*, 2007; BOYSEN *et al.*, 2009b): in paced lines, the conveyor moves all pieces continuously, while in unpaced lines movement is discrete. This chapter focuses on unpaced lines, which are usually classified as synchronous or asynchronous. Because synchronous lines move forward all pieces together, they are cheaper and simpler, yet more restrictive. On the other hand, asynchronous ones allow independent movements for pieces and, consequently, they are more flexible and expensive. Hybrid lines (partly synchronous and partly asynchronous) can be reasonable intermediaries between the two (KOUVELIS; KARABATI, 1999). To the best of the author's knowledge, however, hybrid lines have not been deeply studied in the literature.

Line pace is particularly relevant when mixed-model sequencing (and therefore scheduling) is considered (BOYSEN *et al.*, 2009c). Some relevant scheduling problems in mixed-model unpaced assembly lines are starvations and blockages (BOYSEN *et al.*, 2008). When producing at large scale with relatively stable demands, it is common to seek a cyclical schedule for the line (LEVNER *et al.*, 2010). Such schedules are often based on the Minimal Part Set (MPS) concept (BARD *et al.*, 1992), which is the smallest set of products that will reach the target demand if

produced multiple times.

General and cyclic synchronous scheduling has been studied for quite some time (KARABATI *et al.*, 1992; KARABATI; KOUVELIS, 1996; KOUVELIS; KARABATI, 1999), including variants such as: lines with stochastic processing times (DOERR *et al.*, 2000; CHIANG *et al.*, 2012; URBAN; CHIANG, 2016), order release in synchronous manufacturing (RIEZEBOS, 2010; RIEZEBOS, 2011), and flowshops problems with resources and set-up constraints (WALDHERR; KNUST, 2017). Asynchronous lines have also been studied for long: initially as a "formulation irregularity" (JOHNSON, 1983), but later as a research subject on its own with both stochastic (NAKADE *et al.*, 1997) and deterministic variants (SAWIK, 2000; SAWIK, 2002; SAWIK, 2004; SAWIK, 2012; ÖZTÜRK *et al.*, 2013).

When combining degrees of freedom, such as balancing, sequencing, and scheduling, authors often employ decompositions (KARABATI; SAYIN, 2003; BATTINI *et al.*, 2009) or meta-heuristic procedures (ÖZCAN *et al.*, 2010). Such is the case because, while the combined problem can allow better solutions, complexity and difficulty also tend to increase substantially. Furthermore, works that do combine these degrees of freedom (e.g. Karabati and Sayin (2003)) often operate some simplifications or use indirect performance measures for the throughput (TIACCI, 2015a).

A review of recent works (Section 3.2) shows that, while balancing and scheduling problems are sometimes addressed simultaneously, cyclical steady-state optimization of unpaced lines is still little explored. Most works on mixed-model balancing impose a common cycle time for all product models rather than explicitly considering how balancing and model sequencing interact in terms of blockages and starvations in the steady-state. This chapter extends Chapter 2's formulation to allow simultaneous balancing, sequencing, and cyclical scheduling of unpaced (synchronous, asynchronous, and hybrid) lines.

The remaining of this chapter is organized as follows: Section 3.2 presents a review of recent related works. Section 3.3 presents the context, concepts, and hypotheses of the studied problem. Section 3.4 presents the new MILP model. Section 3.5 presents the results of the comparison between the new formulation and previously described ones. Section 3.6 discusses the obtained results, drawing insights from differences of solutions and computational times found in the tested formulations, as well as its implications. Lastly, the main conclusions drawn from this chapter are summarized in Section 3.7.

3.2 RELATED WORKS

There are many research topics tied to mixed-model assembly lines. This review presents recent works and compares them in regard to how they measure steady-state performance (cycle time), whether they incorporate balancing, sequencing or both, and what are the goal functions.

Alghazi and Kurz (2018) introduced mathematical models that incorporate practical ergonomic and zoning constraints. In this formulation, the goal is to minimize the cycle time, measured as the weighted average of processing times across product models. Such measure can be optimistic depending on the line pace, buffer layout and product sequence, as shown in Chapter 2. A much more common approach is to impose the cycle time as a limit for all models. For instance, Akpinar and Baykasoglu (2014a), Akpinar *et al.* (2017) present formulations and Benders cuts for mixed-model balancing with set-up times between tasks. In sequence dependent formulations, tasks are sequenced rather than product models and the task-wise scheduling of each model must respect a given cycle time. Delice *et al.* (2017), Roshani and Nezami (2017) presented formulations and meta-heuristics for another class of problems in which this task-scheduling is necessary, the multi-manned mixed-model assembly lines. In these lines, workstations have multiple workers or machines in order to reduce line length and, as a consequence, it is necessary to sequence tasks for each model. Cycle time is respected by the task-scheduling of every product model and the goal is to minimize the number of workers and stations (line length). Kucukkoc and Zhang (2014), Kucukkoc and Zhang (2015), Kucukkoc and Zhang (2016) present further variants and meta-heuristics for balancing-sequencing problems with parallel assembly lines. Due to possible differences in cycle times in each line, the minimal common multiple (MCM) of the cycle times is used to define a production cycle. Task times are weighted by the MCM divisors of the assembly line they are processed at and the total weighted processing time of each operator is bounded by the MCM of cycle times. These works have goal functions tied to the minimization of the number of workstations, line length and smoothness indexes of workload distribution.

The cycle time limit for all models is often used as a restriction by authors whose formulations use multiple secondary objectives: Zhong (2017)'s balancing formulation optimizes the number of stations, horizontal and vertical balancing; Saif *et al.* (2014)'s balancing-sequencing one also optimizes vertical and horizontal balancing, and the total flow time of the product models (without considering blockages or starvations).

Stochastic variants, in which task times are random variables were also discussed with two main strategies being recently employed: Dong *et al.* (2018)'s balancing formulation presents a chance-constraint for cycle time that requires the cycle time for each model to be respected with a given probability; Tiacci (2015a), Tiacci (2015b), Tiacci (2017) employs direct simulation-based performance measure for balancing and balancing with buffer allocation problems. Furthermore, he argues that some common performance measures (workload smoothing (THOMOPOULOS, 1970), vertical and horizontal balancing (BECKER; SCHOLL, 2006), weighted average processing times (ALGHAZI; KURZ, 2018), and demanding a cycle time for all models Karabati and Sayin (2003)) are not goals in themselves, but rather supposed means to achieve a high and stable throughput.

Some authors, however, do employ direct performance measures that take product sequence into account: Sawik (2002), Sawik (2004) presents balancing-sequencing models with a makespan minimization goal. A similar approach was employed by Li *et al.* (2017), who present a balancing-sequencing and robot selection formulation to minimize makespan. These models, however, do not produce cyclical schedules, meaning that extending such formulations for large numbers of products tend to be difficult. Sawik (2012), Öztürk *et al.* (2015) present balancing-sequencing formulations with makespan minimization goals and cyclical constraints (between minimal part sets) in order to better approximate steady-state optimization.

Chapter 2 presents a balancing formulation for asynchronous lines with a steady-state performance measure (realized cycle time) that explicitly takes into account a given product sequence. Such performance measure is shown to be accurate and lead to better results than common surrogates, such as vertical balancing, horizontal balancing, and workload smoothing. However, it does not incorporate sequencing decisions and is restricted to asynchronous lines. This chapter extends such formulation to incorporate cyclical sequencing and to be applicable to synchronous and hybrid lines as well.

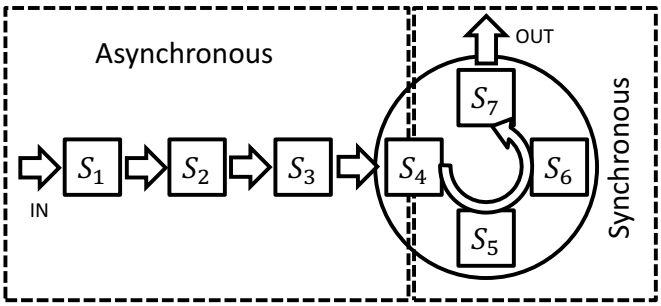
3.3 PROBLEM STATEMENT

The efficiency of unpaced mixed-model assembly lines is heavily influenced by the planning of such lines, including the task allocation, product sequence, and also how pieces are transported between each station. In respect to the transport of workpieces, these lines can either be synchronous, asynchronous or a combination of both (hybrid lines).

A station s has a synchronous pace ($s \in S_{\text{sync}}$) when the departure of the current

product model must coincide with the arrival of the next one. A station s has an asynchronous pace ($s \in S_{\text{async}}$) when the arrival of the next product model can happen after the departure of the current one.

Figure 7 – Hybrid unpaced line layout ($S_{\text{async}} = \{1,2,3,4\}$, $S_{\text{sync}} = \{5,6,7\}$)



Source: Lopes *et al.* (2018)

A hybrid line example is illustrated by Figure 7, in which product flow is asynchronous in stations 1 to 4 and synchronous[1] in stations 5 to 7 due to a circular rotating platform. In this case, approximating the line to an asynchronous one might lead to unfeasible schedules due to the synchronous part. Similarly, approximating the line to a purely synchronous one might eliminate good feasible schedules that take advantage of asynchronous possibilities from the first part of the line.

Synchronous transfer constraints are more restrictive than asynchronous ones and hybrid lines are expected to be intermediaries. An illustrative numerical example is hereby presented to demonstrate these differences: Consider an assembly line with four workstations and three product models with equal demand rates (MPS=$[1,1,1]$). The instance contains four tasks with no precedence relations between them and processing times given by Table 9.

Table 9 – Numerical Example Data - Processing Times (in Time Units, TU)

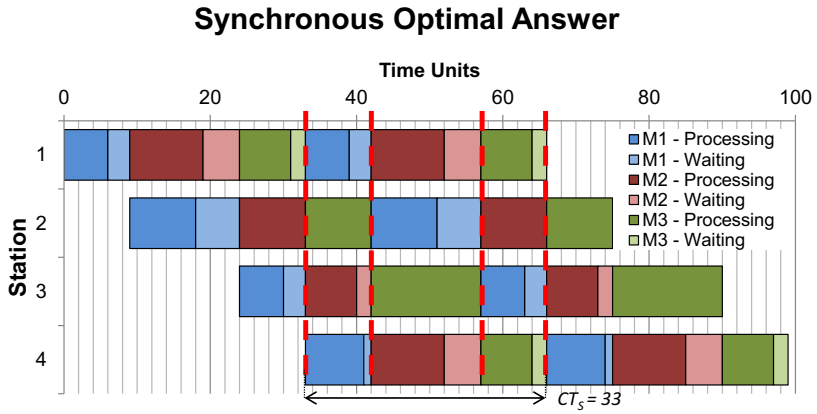
Model	M1	M2	M3
Task 1	6	7	15
Task 2	6	10	7
Task 3	8	10	7
Task 4	9	9	9

The optimal answers for synchronous and asynchronous lines are illustrated by Gantt schedules in Figure 8(a) and Figure 8(b), respectively. In these figures, two replications of the MPS are represented to ease the cyclical behavior visualization. Lighter colors represent

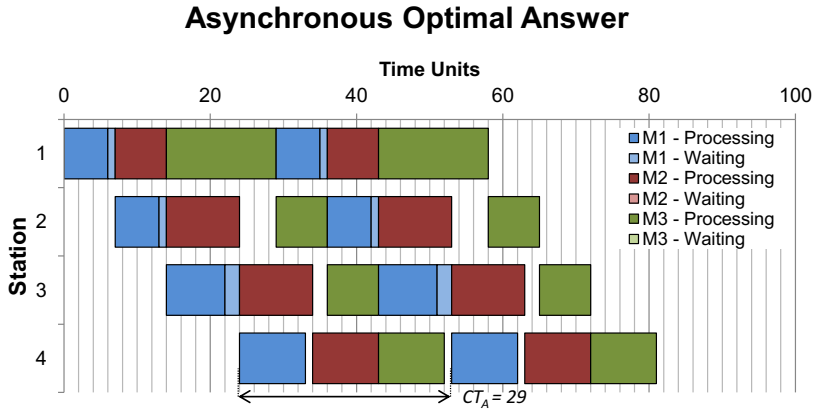
[1] Notice that station 4 is the interface between the synchronous and the asynchronous part of the line. This means that entry times are not bound by synchronism constraints, but departure times are.

Figure 8 – Comparison between line controls

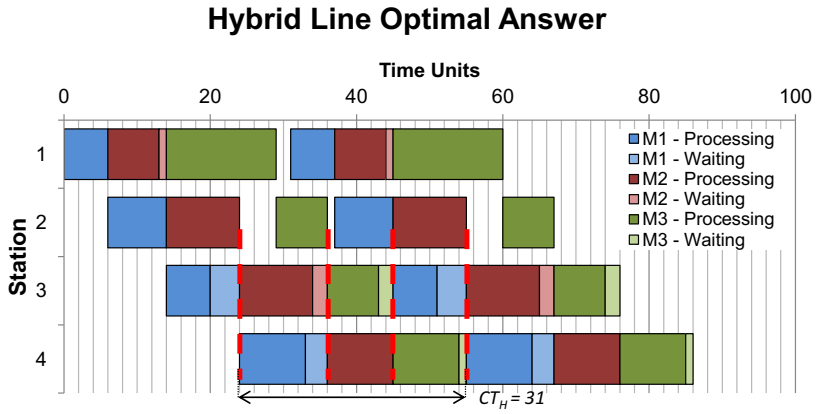
(a) Cyclical Schedule for a Synchronous Line ($S_{\text{sync}} = \{1,2,3,4\}$).



(b) Cyclical Schedule for an Asynchronous ($S_{\text{async}} = \{1,2,3,4\}$) Line.



(c) Cyclical Schedule for a Hybrid Line ($S_{\text{async}} = \{1,2\}$, $S_{\text{sync}} = \{3,4\}$).



Source: Lopes *et al.* (2018)

blockages and blank spaces are starvations; both can be considered idle times. On the cyclical schedule, red dashed lines mark the synchronous transfers of products in Figure 8(a).

If the synchronism constraints are applied to stations three and four, and the asynchronism possibilities are kept at stations one and two, a hybrid line is defined. Figure 8(c) presents the optimal answer for that line. Notice that synchronous transfers are forced on a subset of

stations. Once again, two replications of the MPS are represented in Figure 8(c) and red dashed lines represent the synchronous transfers of products.

Comparing the optimal solutions of each formulation, one can see that the synchronous line has the highest value of cycle time (CT_S=33), the asynchronous one has the lowest one (CT_A=29), and the hybrid line has an intermediary value (CT_H=31). This occurs because, on hybrid lines, the synchronism constraints occur for only a subset of stations. This illustrates a class of problems that cannot be described (and therefore optimized) by previous literature mathematical models.

Furthermore, there are three degrees of freedom that ought to be simultaneously solved in order to achieve optimal solutions: balancing, cyclical sequencing and scheduling. The problem presented herein is based on the following assumptions:

1. Tasks are indivisible and must be performed in the same stations for all product models;
2. There are precedence relationships between some tasks, characterizing technological constraints;
3. The MPS is known, i.e. product demand is known;
4. Transfer times between stations are neglected;
5. Line Control is a station-wise parameter: pieces move to the next station either on a synchronous or an asynchronous pace.

Consequently, a new approach is required to optimally combine the balancing and (cyclical) sequencing degrees of freedom and maximize the line's efficiency by minimizing the average steady-state cycle time as the goal. Section 3.4 seeks to present a mixed-integer linear programming (MILP) formulation that allows to describe and optimize unpaced (synchronous, asynchronous, or hybrid) lines.

3.4 MATHEMATICAL MODEL

In this section, the notation is introduced and the problems presented. The goal is to maximize throughput of an assembly line with $|S|$ stations: $S = \{s_1,s_2,...,s_{|S|}\}$. Each task t_i from the set $T = \{t_1,t_2,...,t_{|T|}\}$ must be assigned to one station. A partial ordering is imposed on task allocation, with the set of precedence relations R with vectors (t_p,t_s). Each of such vector

states that the preceding task t_p must be performed before the succeeding task t_s. The minimal part set contains $|P|$ pieces (jobs), each of them represents a different product model[2]. The processing time of each task t for each model m is given by the matrix $D_{t,m}$ and is not dependent on the station that the task is assigned to. The proposed mixed-integer linear programming formulation for synchronous and asynchronous lines is hereafter presented. The Synchronous Balancing and Cyclical Sequencing Problem (SBCS-P) is defined with Expressions (23) to (33). Its Asynchronous counterpart (ABCS-P) is defined with Expressions (23) to (31) and (34) to (35).

$$\text{Minimize} \quad CT \tag{23}$$

Subject to:

$$\sum_{s \in S} x_{t,s} = 1 \qquad \forall\, t \in T \tag{24}$$

$$\sum_{s \in S} s \cdot x_{t_1,s} \leq \sum_{s \in S} s \cdot x_{t_2,s} \qquad \forall\, (t_1,t_2) \in R \tag{25}$$

$$\sum_{p \in P} y_{p,m} = N_m \qquad \forall\, m \in P \tag{26}$$

$$\sum_{m \in P} y_{p,m} = 1 \qquad \forall\, p \in P \tag{27}$$

$$y_{1,1} = 1 \tag{28}$$

$$Tx_{p,s} \geq \sum_{t \in T} D_{t,m} \cdot x_{t,s} - H \cdot (1 - y_{p,m}) \qquad \forall\, m, p \in P, s \in S \tag{29}$$

$$Tout_{p,s} \geq Tin_{p,s} + Tx_{p,s} \qquad \forall\, p \in P, s \in S \tag{30}$$

$$Tin_{p,s} = Tout_{p,s-1} \qquad \forall\, p \in P, s \in S : s > 1 \tag{31}$$

$$Tin_{p,s} = Tout_{p-1,s} \qquad \forall\, p \in P : p > 1, s \in S_{\text{sync}} \tag{32}$$

$$Tin_{1,s} + CT = Tout_{|P|,s} \qquad \forall\, s \in S_{\text{sync}} \tag{33}$$

$$Tin_{p,s} \geq Tout_{p-1,s} \qquad \forall\, p \in P : p > 1, s \in S_{\text{async}} \tag{34}$$

$$Tin_{1,s} + CT \geq Tout_{|P|,s} \qquad \forall\, s \in S_{\text{async}} \tag{35}$$

The key binary decisions in these problems are: the allocation of tasks to stations, controlled by the binary variable set x (balancing); and the definition of a cyclical model sequence for the pieces, controlled by the binary variable set y (sequencing). The binaries $x_{t,s}$ are set to one when the task t is assigned at the station s. The binaries $y_{p,m}$ when the model m is the

[2] If the MPS contains multiple instances of the same product model, these can be treated as different products with equal processing times.

p^{th} piece in the sequence. Given these key operational decisions are made, scheduling auxiliary variables are employed to measure solution quality. The key idea behind them is to represent steady-state by analyzing a single MPS. The variables sets Tin, Tx, and $Tout$ represent entry, processing, and departure times, respectively, of each piece at (or from) each station.

In a hybrid line, scheduling rules of stations are affected by the transfer system. Some have synchronous transfers (S_{sync}), meaning a piece must enter when the previous piece departs. Other stations have asynchronous transfers (S_{async}), meaning a piece can enter after the previous piece departs. Naturally, these sets combined make into the full set of stations: $S = S_{\text{sync}} \cup S_{\text{async}}$. The time between the entry of the first product and the departure of the last one limits the cycle time for the whole MPS. Such cycle time is represented by the performance measure variable CT[3], whose minimization is the problem's goal.

Constraints (24) to (27) establish that all tasks must be performed, all precedence relations must be respected, all models (jobs) must be produced according to their demand N_m and each piece must be assigned to a unique product model. Constraint (28) provides a symmetry break - valid due to the cyclical nature of the product sequences. Constraint (29) determines the processing time at each piece at each station. Constraints (30) to (31) establish that pieces can only leave a station when processing is complete, and require that pieces enter the next station when they depart from the current one. Finally, the unpaced blockage feature is introduced with Equations (32) and (33) describing the synchronous flow of pieces: all pieces must move one station forward simultaneously. Furthermore, the time between the entry of the first piece and the departure of the last piece is the same at all stations: the cycle time of an MPS is directly tied to the line's productivity. Lastly, Inequalities (34) and (35) describe the asynchronous flow of pieces. The main difference to the synchronous one is that products do not need to change stations simultaneously: a piece can only enter a station after the previous one departs from it, but does not have to do it immediately. Therefore, the last two constraints for the asynchronous problem are inequalities ($\geq$) while those of synchronous lines are equations ($=$), meaning that asynchronous lines are indeed less restrictive.

A particular strength of this formulation is its capacity to represent hybrid unpaced lines: Suppose that some stations of an unpaced line are asynchronous, but the latter ones are synchronous, as illustrated by Figure 7. By using the appropriate constraints for each set of

[3] Also referred to as CT_S, CT_A, and CT_H to differentiate synchronous, asynchronous, and hybrid lines, respectively.

stations[4], it is possible to represent such hybrid lines.

3.5 RESULTS

In this section, the model presented in Section 3.4 is tested on a dataset with 140 instances. The dataset is made available in the Lopes *et al.* (2018)'s supporting information. The balancing data vectors are based on Otto *et al.* (2013)'s SALBP dataset. This dataset contained instances of various sizes, probabilistic distributions for the task times, and precedence diagram structures. In this chapter, tests were conducted in small and medium instances (20 and 50 tasks). In Otto's dataset, task times followed one of three distributions: peak in the bottom, peak in the middle, and bimodal distributions; the latter was chosen to emulate differences between product models. Instances were generated using all precedence relation structures described by Otto *et al.* (2013). In order to better represent the results, instances are gathered based on their Order Strength (OS). The OS is a measure of how restricted is the task allocation due to precedence relations. The range of the OS goes from 0 to 1, where 0 represents an instance without any precedence relation, and 1 results in an instance with a fixed sequence for the task assignments.

In order to better represent differences between models, instance generation was exclusively based on the bimodal data vectors presented by Otto *et al.* (2013). There were grouped five by five in order to generate mixed-model instances. Each instance of Otto's dataset is considered a model in the proposed dataset. The processing times of tasks of different data vectors were converted in processing times of different models. The precedence relations are defined as the ones of the first data vector in each group of five data vectors. This leads to two sets of 35 mixed-model task property instances. The instances were grouped in accordance to their order strength (OS). Each resulting set of instances contains 15 instances with OS = 0.2, 15 instances with OS = 0.6, and 5 instances with OS = 0.9.

Two sequencing problem sizes were considered, first a small minimal part set with one piece of each model (1A, 1B, 1C, 1D, 1E), named S1, and, second, a larger one with relative demands (1A, 3B, 2C, 2D, 1E), named S2. A fixed number of stations (seven) was chosen to conduct experiments with the objective of cycle time minimization.

With two problem sizes for balancing (B1 and B2), two problem sizes for sequencing (S1 and S2) and 35 instances for each combination, a total of 140 mixed-model instances are

[4] In Figure 7's case, synchronous constraints (Equations (32) and (33) should be applied to stations 5-7, representing synchronous departures from stations 4-7. Asynchronous constraints should be applied to the other stations.

defined, grouped in four sets (S1B1, S2B1, S1B2, S2B2). The proposed dataset is tested with the two mathematical models described by Karabati and Sayin (2003) and the proposed formulation for synchronous lines. For asynchronous lines, the proposed formulation was compared with Öztürk *et al.* (2015) CLP formulation.

These references were chosen as they were the ones that most closely matched the cyclic synchronous/asynchronous problems' definitions. However, most works reviewed on Section 3.2 follow (adaptations of) the performance definitions proposed by Karabati and Sayin (2003) (i.e. cycle time as either a bound on weigthed average processing times or on model-wise processing times for all stations), furthermore, the makespan minimization goal employed by Öztürk *et al.* (2015) is also used by other authors (SAWIK, 2012; LI *et al.*, 2017).

Hybrid unpaced lines were assumed to have a synchronous rotating platform containing four stations ($S_{\text{async}} = \{1,2,3,4\}$, $S_{\text{sync}} = \{5,6,7\}$ - This layout is illustrated by Figure 7). Such hybrid line was optimized by the proposed formulation and its answers are compared to those of synchronous and asynchronous counterparts.

All experiments were conducted with the same hardware and software configurations: A 64-bit Intel TM i7 CPU (2.9 GHz) with 8 GB of RAM was employed using eight threads and the IBM ILOG CPLEX Optimization Studio 12.6.3. Time limit was set as 1800 seconds. The two models described by Karabati and Sayin (2003) were implemented according to the paper, Öztürk *et al.* (2015)'s model was adapted from the one made available by the authors at the paper's supporting information.

3.5.1 Overview of literature benchmark models

The proposed formulation is compared in the next sections to three alternative formulations. For synchronous lines, the two models described by Karabati and Sayin (2003) are employed. They are the Total Processing Time Problem (TPTP) model, whose goal function is equivalent to Expression (36), and the Maximum Sub-cycle Time (MST) problem, whose goal function is equivalent to Expression (37). Karabati and Sayin (2003) show that TPTP defines a lower bound for steady-state cycle time (CT), while the second (MST) defines an upper bound. Notice that these expressions do not require any sequencing ($y_{p,m}$) or scheduling variable (Tin, Tx, $Tout$).

$$\text{Minimize} \quad \text{TPTP} = \max_{s \in S} \left(\sum_{m \in P} \sum_{t \in T} D_{t,m} \cdot x_{t,s} \right) \tag{36}$$

Subject to: (24)-(29)

$$\text{Minimize} \quad \text{MST} = |P| \cdot \max_{s \in S} \left(\max_{m \in P} \left(\sum_{t \in T} D_{t,m} \cdot x_{t,s} \right) \right) \tag{37}$$

Subject to: (24)-(29)

For asynchronous lines, the Constraint Logical Programming (CLP) model described by Öztürk *et al.* (2015), hereafter referred to as OZT, is used for comparisons. In the employed notation, the makespan minimization function would be equivalent to Expression (38). Their formulation employs two MPS replications (the P set is doubled) as well as symmetry constraints between each of these replications, which are equivalent to Equation (39). Notice that OZT's equivalent MILP formulation requires sequencing and scheduling variables.

$$\text{Minimize} \quad Tout_{|P|,|S|} \tag{38}$$

Subject to: (24)-(35)

$$\text{and } y_{p,m} = y_{p-|P|/2,\, m-|P|/2} \qquad \forall\, m \in P, p \in P: \;\; m, p > |P|/2 \tag{39}$$

Both the TPTP and MST models were re-implemented according to Karabati and Sayin (2003)'s paper. It was not necessary however, to fully translate Öztürk *et al.* (2015)'s model to mixed-integer programming: their paper's supporting information includes the full implementation files in Constraint Logical Programming and can be solved by the CP solver of the IBM ILOG interface used to develop and test the proposed formulation. Only minor adaptations on input files (.dat) were necessary, in order to adapt the OZT model to the proposed dataset.

3.5.2 Synchronous Lines

The synchronous version of the proposed model (P_S) was tested and compared to two literature benchmarks designed to address this problem in particular: the Maximum Sub-cycle Time (MST) formulation, and the Total Processing Times Problem (TPTP), both presented by Karabati and Sayin (2003). MST seeks to minimize the worst sub-cycle time by minimizing the

largest model-wise processing time at stations. TPTP seeks to optimize the best case scenario by minimizing the largest station-wise sum of model-wise processing times weighted by their demands. The optimal value of the MST model offers a valid upper bound, while the optimal value of the TPTP model offers a lower bound.

Neither TPTP nor MST explicitly considers sequencing. Therefore, in order to make a valid comparison, their answers were post-processed to report the value of steady-state efficiency generated by the best cyclical sequence. Such post-processing was not applied to the proposed formulation. Due to the combined degrees of freedom, P_S is expected to generate better answers than MST and TPTP. A comparison ratio is defined for each of the benchmark models by computing $1 - UB_{MST}/UB_{P_S}$[5] and $1 - UB_{TPTP}/UB_{P_S}$. These measures offer percentage differences obtained by the proposed model P_S. Negative values indicate that P_S was outperformed in the specific instance.

Nopt indicates the number of instances solved to optimality for the proposed formulation. For example: 13/15 indicates that 13 out of the 15 instances in that set were solved to optimality. Table 10 presents the results obtained for synchronous lines. The number of instances P_S solved to optimality (*Nopt* (P_S)) in each sub-dataset (e.g. S1-B1 OS=0.2) is reported at the rightmost column of Table 10.

Table 10 – Results for synchronous lines

Data Info		Avg. Exec. Time (s)			Avg. Ratios to P_S		Min. Ratios to P_S		
Data Set	OS	MST	TPTP	P($_S$)	MST	TPTP	MST	TPTP	Nopt (P_S)
S1-B1	0.2	2	1.2	469	5.21%	13.07%	1.78%	3.42%	15/15
	0.6	0.7	0.7	55.5	5.36%	12.58%	1%	8.32%	15/15
	0.9	0.6	0.4	1.8	2.87%	8.3%	1.18%	3.53%	5/5
S2-B1	0.2	2.2	1.1	1800	2.83%	11.53%	0.95%	8.57%	3/15
	0.6	0.7	0.6	922.5	4.11%	10.59%	0.61%	6.03%	15/15
	0.9	0.6	0.6	26.9	4.19%	8.29%	2.82%	5.31%	5/5
S1-B2	0.2	1683.3	1359	1800	4.82%	14.67%	1.25%	12.09%	3/15
	0.6	93.4	20.9	1031.9	3.24%	12.62%	1.43%	7.65%	13/15
	0.9	0.6	0.7	11.2	4.1%	6.15%	2.41%	4.36%	5/5
S2-B2	0.2	1491.6	1246.4	1800	1.84%	11.51%	-0.68%	7.61%	0/15
	0.6	63.4	18.4	1545.9	1.57%	9.61%	-0.47%	3.36%	8/15
	0.9	0.7	0.7	94	3.25%	6.89%	0.64%	4.78%	5/5
All	-	357.7	283.8	796.6	3.62%	10.48%	-0.68%	3.36%	92/140

3.5.3 Asynchronous Lines

The asynchronous version of the proposed model (P_A) was tested and compared to literature benchmarks designed to address this problem in particular: the CLP formulation

[5] Notation: UB_{method} denotes the steady-state cycle time obtained by specific method/model during experiments.

presented by Öztürk *et al.* (2015). OZT seeks to generate better cyclical schedules by minimizing the makespan of two replications of the minimal part set. This is an indirect performance measure and is not the model's goal in itself, but rather a method to achieve a low cycle time. OZT reported values of cycle time were defined according to the original implementation files provided by the authors' supporting information.

Table 11 – Results for asynchronous lines

Data Info		Avg. Exec. Time (s)		Ratios to P_A		
Data Set	OS	OZT	P_A	Avg.	Min.	Nopt (P_A)
S1-B1	0.2	1800	58.5	17.24%	7.6%	15/15
	0.6	1800	15.9	24.66%	6.36%	15/15
	0.9	1800	2.7	21.7%	13.41%	5/5
S2-B1	0.2	1800	1267.9	24.88%	17.56%	9/15
	0.6	1800	454	29.54%	16.58%	14/15
	0.9	1800	41.1	30.34%	22.27%	5/5
S1-B2	0.2	1800	1272.5	25.9%	12.14%	6/15
	0.6	1800	475	34.36%	14.34%	15/15
	0.9	1800	25.2	53.32%	29.75%	5/5
S2-B2	0.2	1800	1800	31.57%	25.9%	0/15
	0.6	1800	1627.7	42.72%	25.95%	4/15
	0.9	1800	171.9	63.79%	43.26%	5/5
All	-	1800	601	33.34%	6.36%	98/140

Just as the proposed formulation (P_A), OZT explicitly considers sequencing. Therefore, in order to make a valid comparison, no post-processing was applied to either model. The proposed formulation P_A incorporates a direct performance measure of cycle time and is, therefore, expected to generate better answers than OZT. A comparison ratio is defined for each of the benchmark models by computing $1 - UB_{OZT}/UB_{P_A}$. This measure offers percentage difference obtained by the proposed model P_A. Table 11 presents the results obtained for asynchronous lines. The number of instances P_A solved to optimality (*Nopt* (P_A)) in each sub-dataset (e.g. S1-B1 OS=0.2) is reported at the rightmost column of Table 11.

3.5.4 Hybrid Lines

Unpaced lines that combine synchronous and asynchronous parts are described as hybrid ones. To the best of the author's knowledge, no previous paper presents a model that can be used as a comparative benchmark. Therefore, the proposed formulation for the hybrid lines P_H is hereby compared to the proposed formulations for synchronous and asynchronous lines (P_S and P_A).

Given that for hybrid lines (P_H), the synchronism constraints are only applied to a

subset of stations, it is expected that these lines performance will be an intermediary between the synchronous lines (P_S) and the asynchronous ones (P_A). A comparison ratio is defined for each of the line control variants by computing $1 - UB_{P_S}/UB_{P_A}$ and $1 - UB_{P_H}/UB_{P_A}$, this measure offers percentage difference in cycle time between synchronous/hybrid lines with respect to asynchronous ones. Negative values indicate that P_A was outperformed by P_S or P_H in the specific instance, this is only possible for instances in which P_A was not solved to optimality. Table 12 presents the results obtained for hybrid lines. The number of instances P_H solved to optimality (*Nopt* (P_H)) in each sub-dataset (e.g. S1-B1 OS=0.2) is reported at the rightmost column of Table 12.

Table 12 – Results for hybrid lines

Data Info		Avg. Exec. Time (s)			Avg. Ratios to P_A		Min. Ratios to P_A		
Data Set	OS	P_S	P_H	P_A	P_S/P_A	P_H/P_A	P_S/P_A	P_H/P_A	Nopt(P_H)
S1-B1	0.2	106.3	70.3	58.5	3.71%	0.73%	1.4%	0.18%	15/15
	0.6	21.9	17.7	15.9	4.44%	1.05%	2.81%	0%	15/15
	0.9	3.6	2.7	2.7	6.76%	1.75%	3.97%	0.16%	5/5
S2-B1	0.2	1579.3	1293.9	1267.9	3.86%	0.85%	2.6%	-0.1%	8/15
	0.6	559.6	532	454	4.3%	1.16%	2.04%	0.07%	15/15
	0.9	40.9	37.7	41.1	5.69%	1.45%	5.11%	0.96%	5/5
S1-B2	0.2	1690.4	1567.6	1272.5	1%	0.1%	0.31%	-0.52%	3/15
	0.6	705.6	457.6	475	1.31%	0.38%	0%	0%	14/15
	0.9	35.7	32.6	25.2	2.28%	0.69%	1.8%	0.3%	5/5
S2-B2	0.2	1800	1800	1800	1.66%	0.62%	0.69%	0.05%	1/15
	0.6	1716.5	1591.4	1627.7	1.38%	0.23%	0.48%	-0.7%	12/15
	0.9	264	226.3	171.9	2.6%	0.71%	2.08%	0%	5/5
All	-	710.3	635.8	601	3.25%	0.81%	0%	-0.7%	103/140

3.6 DISCUSSIONS

Karabati and Sayin (2003)'s TPTP model performed significantly worse than their MST model (Table 10). In average, to simply minimize the sum of processing times at stations does not optimize the line. In a single instance (Dataset S1-B1, case 6), however, the TPTP model did produce a better solution than the MST. This indicates that minimizing the maximum model-wise processing time does not necessarily optimize the line either. That said, the MST did perform significantly better than the TPTP, but the proposed formulation P_S outperformed the MST one by 3.62% percent in average. This difference in performance is due to taking into account sequencing and scheduling problems explicitly. Note that this comes at a cost for the proposed formulation: in average, it took significantly longer processing times in all datasets.

Öztürk *et al.* (2015)'s CLP model was considerably outperformed by the proposed

MILP model (Table 11). This might be due to the fact that the search field is rather large and not very restricted, leading to weaker performance on the constraint propagation and domain reduction mechanisms CLP relies rather heavily on. Furthermore, the lack of a systematic lower bounds for the OZT formulation meant it could not solve a single instance to optimality. In average, the proposed formulation produced asynchronous cyclical schedules with 33% lower cycle times than the ones generated by Ozturk's model. Notice that the proposed model also produced solutions faster, reaching optimal solutions in 98 out of 140 instances, meaning that, for asynchronous lines, the proposed formulation also displayed a better time performance.

For both synchronous and asynchronous lines, the proposed model generated better solutions in average than the tested benchmarks. The proposed formulation was outperformed in only 2 out of the 140 synchronous instances (Dataset S2-B2, cases 14 and 25 in the supporting information). Notice that cases with higher ordering strength (OS) were solved faster for all models: Even the larger instances (S2-B2) with high OS (0.9) could be optimally solved in a relatively low computation time (172s in average for P_A), while smaller ones (S2-B1) with lower OS (0.6) took much longer (454s in average for P_A). The higher solution difficulty of problems with lower OS is also reflected by the number of solutions solved to optimality: for the harder dataset (S2-B2), only one instance with low OS (0.2) was solved to optimality (by P_H). Out of the instances with medium OS (0.6), 24 were solved to optimality (8 by P_S, 4 by P_A, and 12 by P_H).

Notice that the number of instances solved to optimality is higher for hybrid lines (103) than for synchronous (94) or asynchronous ones (98) (Table 12). This reflects the fact that hybrid lines are, on the one hand more flexible than synchronous ones (and, hence, reach lower upper bounds), and on the other hand more restrictive than asynchronous ones (and, hence, allow higher lower bounds). This can explain why significantly more instances of the large dataset (e.g. S2-B2 OS=0.6) could be solved to optimality for hybrid lines (12/15) than for synchronous (8/15) or asynchronous ones (4/15).

The hybrid line case study demonstrated that such lines behave as intermediaries between the synchronous and asynchronous ones. In average, synchronism demanded a 3.25% higher cycle time when applied to the whole line and a 0.81% higher cycle time when applied to half the line. In three cases[6], the hybrid line reached the same value of cycle time as the asynchronous one (both the hybrid and asynchronous instances were solved optimality). If the

[6] S1-B1 instance 28, S1-B2 instance 26, S2-B2 instance 34

cost of hybrid lines is also (as expected) an intermediary between the costs of synchronous and asynchronous ones, then hybrid lines can be seen as cheaper alternatives to asynchronous lines: In the computational case study, hybrid lines offer most of the productivity difference with a smaller cost.

This highlights a particular value of the proposed formulation as it allows performance evaluations of different line control systems. This, in turn, offers to managers a possibility to quantitatively measure the trade-offs allowed by the both full and partial (hybrid) asynchronism. While such evaluation was previously possible via simulation (TIACCI, 2015a), simulations would require both balancing and sequencing solutions as inputs. The presented model allows balancing and sequencing to become degrees of freedom to be optimized, leading to smaller cycle times and better economical results.

3.7 CONCLUSIONS

This chapter presented an MILP formulation that successfully optimizes small and medium sizes of mixed-model unpaced assembly lines. The proposed model simultaneously solves the balancing of mixed-model lines along with the cyclical sequencing and scheduling of products in the line. The formulation is flexible enough to consider both synchronous and asynchronous assembly lines, with the possibility of treating hybrid (partly synchronous, partly asynchronous) lines as well.

A dataset for the simultaneously balancing and cyclical sequencing of unpaced Assembly Lines containing 140 instances is proposed. Instances are based on Otto *et al.* (2013)'s SALBP instances. The dataset can be solved for both synchronous and asynchronous lines, and is used as a benchmark to compare the proposed formulation to the former best methods for both synchronous and asynchronous lines. With a time limit of 1800 seconds, the proposed model outperformed previously described alternatives in 138 out of the 140 instances of synchronous lines and all 140 instances for the asynchronous lines. In average, the proposed formulation found answers with 3.62% smaller cycle time for the synchronous cases and 33.3% for the asynchronous lines.

The structure of the proposed model allows efficiency comparisons of line control in assembly lines. To the best of the author's knowledge, no other previous work on assembly line balancing considers the optimization of hybrid synchronous/asynchronous lines. As expected, the independent movement between the stations of asynchronous lines results in lower cycle

times when compared to the synchronous counterpart. In average, for the proposed dataset, the asynchronous lines can produce 3.25% faster and in only one instance the optimal answers of both synchronous and asynchronous were the same. A hybrid line consisting of half asynchronous and half synchronous stations achieved an intermediary result with cycle times slightly higher than the fully asynchronous line (0.81%).

Although the efficiency of asynchronous lines are greater or equal to the synchronous lines, the latter have smaller implementing costs. With the flexibility of the proposed formulation, the output of lines can be determined and used in an economic evaluation of the transport systems. Hybrid lines are especially advantageous when asynchronous stations can be installed to take advantage of the more flexible piece transportation, while simpler synchronous stations are kept where these benefits are not present.

4 INCORPORATING PARALLEL STATIONS

This chapter is based on an article published in the Journal of Manufacturing Systems (LOPES *et al.*, 2019). The article is an extended version of a conference paper presented at the XLIX SBPO in Blumenau (LOPES *et al.*, 2017) - which was selected amongst the five best papers presented in the conference. This chapter extends Chapter 3's balancing-sequencing formulation to describe cyclical scheduling with parallel stations. In Lopes *et al.* (2019), a more direct comparison to Öztürk *et al.* (2015)'s formulation is given. Therefore, several basic considerations in this chapter differ from the Chapters 2 and 3. For instance, each product model has its own set of tasks and precedence relations.

4.1 CONTEXT

Assembly lines are product oriented manufacturing layouts commonly employed for large scale production of similar products (SCHOLL, 1999). The optimization of such lines has been subject of a substantial body of literature, including balancing task distributions (BOYSEN *et al.*, 2007) and scheduling product models (BOYSEN *et al.*, 2009c). While these inter-related optimization problems are usually treated in a sequential (hierarchical) manner, multiple recent works have combined them (ÖZCAN *et al.*, 2010; HAMZADAYI; YILDIZ, 2012; ÖZTÜRK *et al.*, 2013; KUCUKKOC; ZHANG, 2016).

Balancing (and balancing-sequencing) works range from studies on simple single product deterministic lines (SCHOLL; BECKER, 2006) to ones that incorporate diverse realistic features (BATTAÏA; DOLGUI, 2013). Some of the commonly considered aspects are: set-up times between tasks (AKPINAR; BAYKASOGLU, 2014a; AKPINAR; BAYKASOGLU, 2014b), ergonomic constraints (TIACCI; MIMMI, 2018; ALGHAZI; KURZ, 2018), space constraints (SAWIK, 2002; CHICA *et al.*, 2010; ÖZTÜRK *et al.*, 2012; CHICA *et al.*, 2016), worker movement between stations (SIKORA *et al.*, 2017), among others. Extensive classifications of these problems are provided by Battaïa and Dolgui (2013) and Boysen *et al.* (2008). In this chapter, parallel stations and product diversity (mixed-model) under stable demands are the main considered features.

There are multiple manners assembly lines employ parallelism (BOYSEN *et al.*, 2007): Some authors consider parallel lines with crossover workers (ÖZCAN *et al.*, 2010; KUCUKKOC;

ZHANG, 2014), others consider parallel work within a (multi-manned) station (KELLEGÖZ, 2017; ROSHANI; NEZAMI, 2017). This chapter considers parallel stations (ÖZTÜRK *et al.*, 2015) under an unpaced asynchronous line control (BOYSEN *et al.*, 2008). Parallel stations allow higher station-wise processing times, more balancing flexibility, and also offer better recovery from failure capacity as the line is not necessarily stopped when one of the stations needs maintenance (REKIEK *et al.*, 2002; BUKCHIN; RUBINOVITZ, 2003; LUSA, 2008). Asynchronous line control means that products can move downstream when processing at the current station is completed and the next stage has an empty station. Blockages occur when a product's processing is completed and all stations of the next stage are busy, and starvations occur when a station is empty and no product from the previous stage has been yet fully processed (BOYSEN *et al.*, 2008).

Product diversity can also affect assembly lines in multiple manners (BATTAÏA; DOLGUI, 2013): in multi-model lines set-up times exist between models, hence, lot-sizing becomes an important part of the optimization process (BOYSEN *et al.*, 2008). This chapter focuses on less restrictive mixed-model lines, in which there is no such model-wise set-up times. Furthermore, demand rates are considered stable enough, so that cyclically scheduling (LEVNER *et al.*, 2010) the line is viable. The mixed-model aspect of the problem has also been treated differently by multiple authors: some consider that total processing times (or averages weighted by the demand rates) should be equalized (ALGHAZI; KURZ, 2018; SAWIK, 2002; SAWIK, 2000). Chapter 2 shows that this consideration can be too optimistic without a sufficient number of buffers. A more common approach is to impose a cycle time constraint on the processing times of all product models (AKPINAR; BAYKASOGLU, 2014a; ROSHANI; NEZAMI, 2017). The minimization of such cycle time has also been proposed as a goal function (KARABATI; SAYIN, 2003). However, this approach can be pessimistic as the differences between product models can often compensate higher processing times for one model, especially if its demand rate is low, as shown in Chapter 2. Other goal functions such as workload smoothing (KUCUKKOC; ZHANG, 2014), vertical balancing (SAIF *et al.*, 2014; ZHONG, 2017) and total deviations from average (KIM *et al.*, 2000; KIM *et al.*, 2006) have also been employed, but, as Tiacci (2015b) point out, these are not goals in themselves, but rather supposed means to achieve a high and stable throughput. Tiacci (2015a) proposes a simulation-based evaluation process to assess the quality of solutions. These simulation-based works employ an assembly line simulator (TIACCI, 2012) that uses common and intuitive priority and scheduling rules, such as: moving products

forward as soon as possible and giving priority to pieces based on when they were completed. These rules have been shown to converge towards accurate steady-state assessments for straight lines without parallelism, as shown in Chapter 2. However, it is not clear if simulations based on such rules necessarily converge towards the optimal cyclical schedule (LEVNER *et al.*, 2010) given a launch order in lines with parallel stations.

Cyclical scheduling (HANEN; MUNIER, 1994) is a research field closely related to the studied problem. It consists on providing schedules that can be infinitely repeated after a certain cycle time. A review on cyclical scheduling is presented by Levner *et al.* (2010). A usual assumption on this area is that machine-wise processing times of products are known, which in the assembly line context means line balancing (task allocation) is a parameter. Cyclical scheduling is applied to project scheduling (DINECHIN; KORDON, 2014), robotic cell scheduling (BRAUNER, 2008; ELMI; TOPALOGLU, 2017), to job-shops (BRUCKER; KAMPMEYER, 2008; QUINTON *et al.*, 2018), and flow-shops (SAWIK, 2012; BOZEJKO *et al.*, 2016). The latter category is the problem class which most closely resembles the studied assembly lines.

A recent work (ÖZTÜRK *et al.*, 2015) has presented a constraint logic programming model to optimize asynchronous mixed-model lines with parallel stations. Its core rationale is to combine balancing and cyclical scheduling to minimize the makespan of multiple replications of the Minimal Part Set (LEVNER *et al.*, 2010) (MPS) and, indirectly, minimize the cycle time. Chapter 2's case study has shown that this approach does converge towards steady-state optimization as more MPS replications are considered, and presented a formulation to assess the performance of (single station) asynchronous lines with a single MPS replication. Chapter 3 presents an extension of this formulation that incorporates sequencing variables for lines without parallelism. However, generalizing that formulation to lines with parallel workstations is challenging due to the fact that the order products enter a stage can differ from the order products depart from it. Lopes *et al.* (2017) describes a simplifying ordering hypothesis (entry order equals departure order) and present the mathematical model to approximately describe the line. This chapter removes such simplification and exactly extend the mixed-integer linear programming formulation presented in Chapters 2 and 3 to incorporate sequencing and cyclical scheduling for asynchronous mixed-model lines with parallel stations. Illustrative examples are given to demonstrate some of the developed features.

The dataset presented by Öztürk *et al.* (2015) is employed as a benchmark, consequently, all benchmark features are incorporated: Mixed-model balancing, parallel stations, and cyclical

scheduling are the main features and this chapter's focus. However, Öztürk *et al.* (2015) employs space constraints, different precedence relations per product model, and model-wise flexible task assignments. These are added to the proposed model so that direct comparisons are possible.

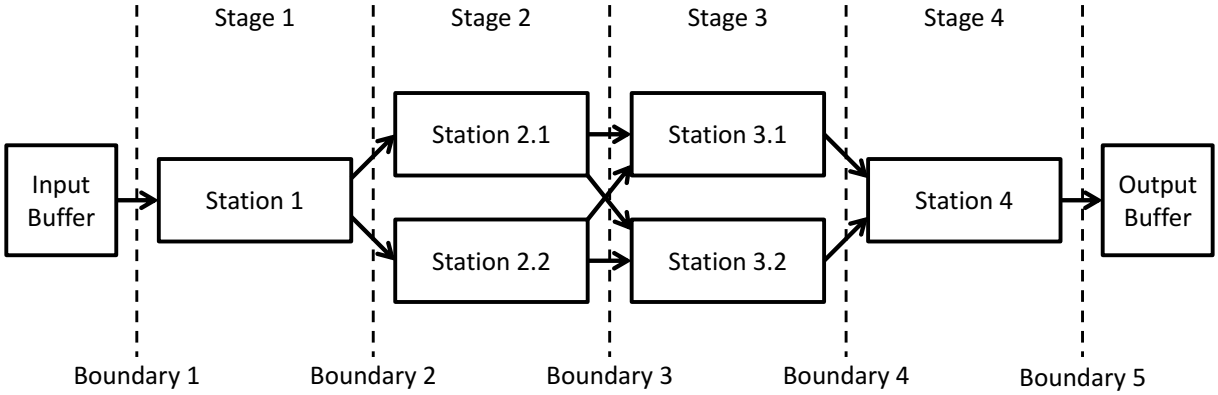
This chapter is structured as follows: Section 4.2 describes the optimization problem. Section 4.2.1 discusses the proposed cyclical scheduling concept (Extension of the formulation presented in Chapters 2 and 3) and how it compares to previous ones. Section 4.3 presents the developed mathematical model. Section 4.4 reports the results of computational tests performed on the dataset provided by Öztürk *et al.* (2015). Last, Section 4.5 summarizes the findings and contributions of the chapter.

4.2 PROBLEM STATEMENT

Each product **M**odel m (job) in the set M has a specific set T_m of **T**asks t which must be assigned to one of the **S**tages s in the set S. Tasks can be assigned to different stations for each model, and the union set of all tasks for all models is denoted as T. Each product model m has a set of **P**recedence relations P_m for task pairs (t_1,t_2), which require task t_1 to be performed before task t_2 for the model m. Each stage s has a given amount of **A**vailable space A_s, which is consumed by the assignment of tasks to them: When task t is assigned to the stage s for any model m, it **R**equires a given amount of space $Q_{(t,s)}$. The **D**uration of each task t at each stage s for each model m is given by the parameter $D_{(t,m,s)}$. Each stage s has a parallelism degree k_s, meaning it has k_s stations (workers, machines) and can process up to k_s products simultaneously. Each of the k_s products processed simultaneously in stage s are independent: they can depart the stage s at any time provided that they have completed processing at stage s and stage $s+1$ is not full (i.e. has less than $k_{(s+1)}$ products currently at it). The problem's major physical layout concepts are illustrated by Figure 9.

The problem consists of providing a line balancing (distributions of tasks to stages for each product model) and cyclical schedule for the Minimal Part Set (MPS) that minimize cycle time (maximize throughput) such that: precedence relations and space constraints are respected; No buffer exists between stages; Infinite buffers exist before the first stage and after the last one.

Figure 9 – Line layout example with four stages, $k_1 = k_4 = 1$, and $k_2 = k_3 = 2$



Source: Lopes *et al.* (2019)

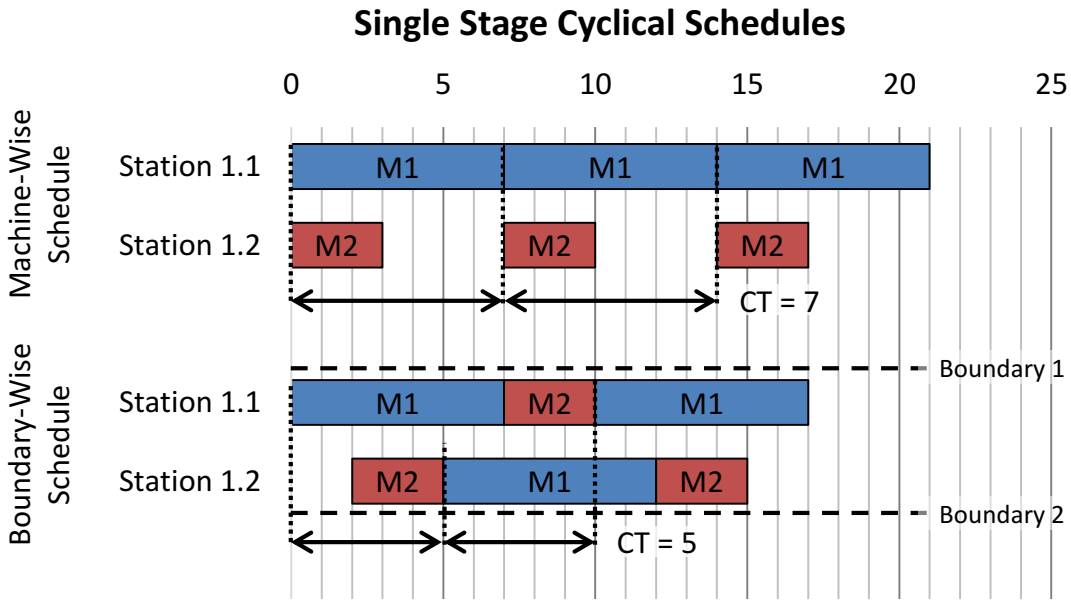
4.2.1 Cyclical Scheduling with Parallelism

The standard definition of a cyclical schedule is one that repeats every CT units of time. This period is called the cycle time (LEVNER *et al.*, 2010) and it can be interpreted as: if an event happens at a given time t, it happens again for the next MPS at $t + CT$. When there are no parallel machines or workstations, this definition leaves little margin for interpretation, as discussed in Section 4.1. With parallelism, the most common interpretation (PINEDO, 2008) for this definition is: if at time t the model m is processed at the k^{th} position (station, machine) of the stage s then at time $t + CT$ the equivalent model of the next MPS is processed at the k^{th} position of the stage s. These positions are often referred to as stations, workers or machines. This cyclical scheduling concept is hereafter referred to as machine-wise cyclical concept. This interpretation implies on a designation of pieces within an MPS to machines (workers, or stations) in each stage.

However, in the context of parallel identical machines, it is not necessary to assign models to specific machines. It is sufficient to ensure that at most k_s models are within stage s at the same time. By defining entry and departure boundaries for each stage, the cyclical schedule can be redefined in terms of when these boundaries are crossed, i.e. when pieces enter and depart each stage. These boundaries are illustrated by Figure 9, and the cyclical nature of their schedules can be defined as follows: if at time t the model m enters (departs from) the stage s, then at time $t + CT$ the equivalent model of the next MPS enters (departs from) the stage s. Notice that no mention of the specific machine is necessary. This subtle difference in interpretation means that model-machine assignments can differ in each replication of the MPS. This cyclical concept is referred to as boundary-wise cyclical concept.

At first glance, this boundary-wise interpretation might seem to violate the purpose of cyclical scheduling: provide a stable and predictable pattern. However, the boundary-based concept does provide both these results: one full MPS is still guaranteed to be produced every CT Time Units (T.U.), and the schedule is completely defined by one MPS, although this might not be as apparent. The major difference lies in the additional flexibility, which is hereafter illustrated: consider a flowshop with a single stage equipped with two parallel identical machines and an MPS with one unit of products M1 and M2, with processing times 7 and 3 T.U., respectively. If each piece is assigned to a machine, then the minimal cycle time of a cyclical schedule is 7 T.U.. However, if the boundary concept of cyclical schedule is employed, then the minimal cycle time becomes 5 T.U.: one unit of product M1 can enter the stage whenever a unit of product M2 departs it and vice-versa. By doing this, one unit of each product type crosses entry and departure boundaries every 5 T.U.. These schedules are illustrated by Figure 10. From the perspective of the machines, a cyclical pattern is achieved with two replications of the MPS, but from the perspective of the boundaries, the schedule is cyclically defined within a single MPS.

Figure 10 – Comparison between machine-wise and boundary-wise optimal cyclical schedules



Source: Lopes *et al.* (2019)

If the flowshop has a single station at each stage, then the second model can only enter it after the first one has departed it. This generalizes to the $(n+1)^{\text{th}}$ model and the n^{th} one. The cycle time is bounded by the time difference between the entry of the first model and the departure of the last one, similarly to Chapter 2. In the case without parallelism, the boundary

and the machine perspectives of cyclical scheduling are essentially the same. However, if the flowshop has multiple parallel stations, these concepts differ: Consider the case with a parallelism degree of $k = 2$ and an MPS with more than two models. The first two (k) models to enter a stage can do so independently. The third ($k + 1$) model, however, can only enter the stage after the first model to depart from the stage departs from it[1]. This generalizes to the $(n + k)^{\text{th}}$ model and the n^{th} one. Now consider the next MPS entering the line. The first model of the next MPS is able to enter the stage after the second last (k^{th} last) model of the first MPS departs it, and the second model of the next MPS is able to enter the stage after the last model of the first MPS departs it. In this case, there are two ways that cycle time is bounded per stage. Contrary to single lines, however, the n^{th} model to enter a stage is not necessarily the n^{th} one to depart it. This can happen due to processing times differences of the models in each stage. Consequently, the order products cross the entrance boundary of a stage is not necessarily the same order they cross the departure one, and hence the cyclical nature of the schedule (while fully described by one MPS) might only become fully apparent in a machine-wise sense when multiple MPSs are considered.

Table 13 – Illustrative case data: model-stage processing times (Time Units - T.U.)

Stage	S1	S2	S3	S4
Model 1	2	4	9	3
Model 2	2	8	2	2
Model 3	3	3	2	2
Model 4	3	4	5	3

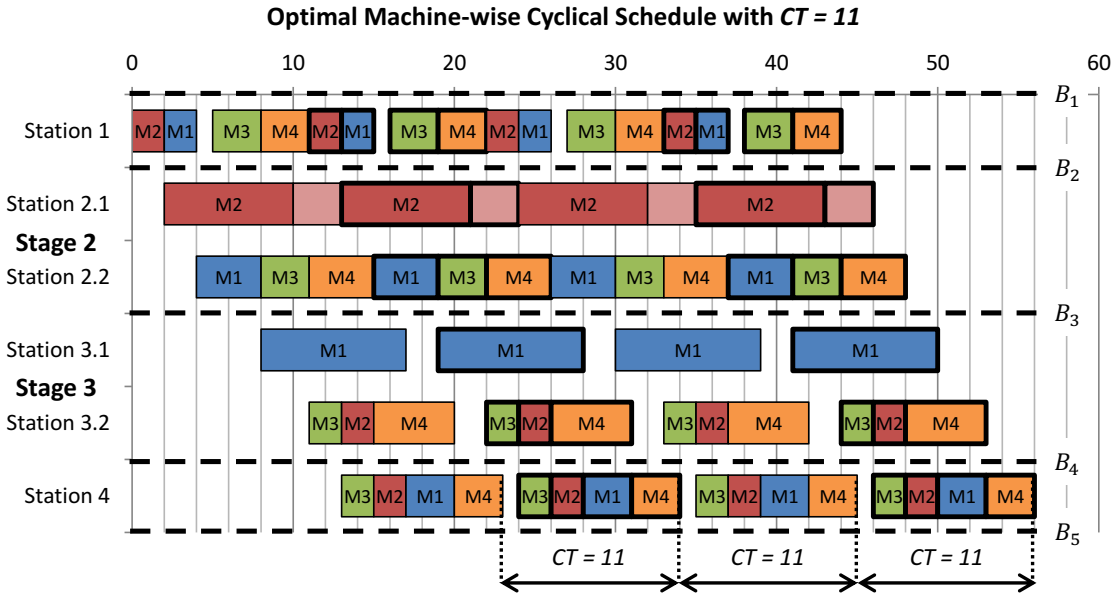
Consider an illustrative case with the processing time data from Table 13 and an MPS with one unit of each product model. The case's line layout is presented in Figure 9 (with $k_1 = k_4 = 1$ and $k_2 = k_3 = 2$). Given the processing times at the second stage, it is clear that a machine-wise cyclical schedule cannot arrange models in two groups with a cycle time smaller than 11 T.U.. Indeed, Figure 11(a) presents one such optimal schedule. However, the boundary-wise optimal shown by Figure 11(b) cyclic schedule allows a cycle time of 10 T.U.. The boundaries are indicated by the dashed lines labeled B_1,...,B_5. Cycle time measurements are presented bellow the last stage in each schedule.

Figure 11(b) further exemplifies two different blockage concepts that is employed in Section 4.3, namely regular and cyclical blockage. These concepts are based on the MPS concept: regular blockages occur between pieces of the same MPS, while cyclical blockages occur between pieces of different MPS. While both are tied to scheduling variables, the latter is

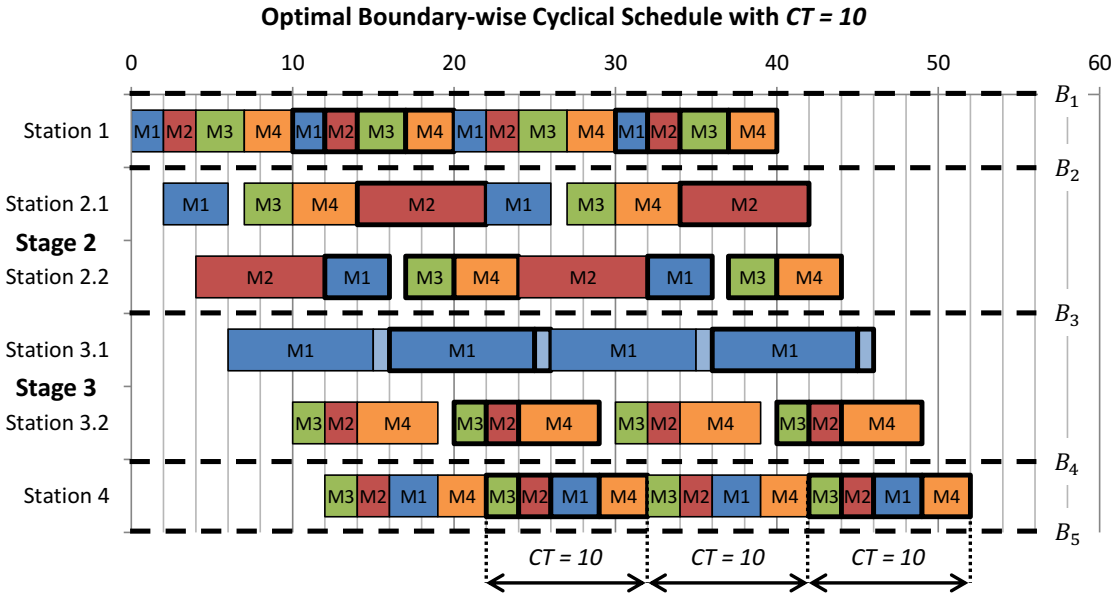
[1] The first model to depart the stage is not necessarily the first one to enter it. This is illustrated by Figure 11(b), where model M1 is the first to enter Stage 3, but the third to depart from it.

Figure 11 – Machine-wise and Boundary-wise cyclical Schedules

(a) Optimal machine-wise solution for the illustrative case



(b) Optimal boundary-wise solution for the illustrative case



Source: Lopes *et al.* (2019)

tied to the steady-state cycle time, hence the label cyclical blockage. Notice that in Figure 11, even MPSs are highlighted with thicker borders. Some examples of these blockage types are hereby provided: In Figure 11(b), model M2 "follows" model M1 at the first stage, i.e. it is blocked by it. Between minimal part sets, model M1 "cyclically follows" model M4, i.e. it is cyclically blocked by it. Due to the difference of entry orders and departure orders at each stage, different sets of blockages and cyclical blockages occur per stage.

Notice that the boundary-wise cyclical concept allows more flexible scheduling patterns. The mathematical model presented in Section 4.3 is defined based on the boundary-wise cyclical schedule concept.

4.3 MATHEMATICAL MODEL

The following mixed-integer linear programming model is proposed to describe the problem at hand. In order to do so, the following decision variables are introduced: the continuous variable CT measures the line's cycle time; the binary variable $x_{(t,m,s)}$ is set to 1 if task t is assigned to stage s for product model m; the continuous variables $Tin_{m,s}$, $Tx_{m,s}$, $Tout_{m,s}$ measure entry, processing, and departure times of each model m at, in, and from each stage s. The binary ordering variables $y_{m,n,s}$ are set to 1 if the model m is the n^{th} one to pass through the s^{th} boundary, meaning that it is the n^{th} piece to leave stage $s-1$ and enter stage s. For instance, in Figure 11(b), model 1 ($m=1$) is the first ($n=1$) model to enter stage 1 ($s=1$), therefore, $y_{1,1,1}=1$. Similarly, model 3 is the first model to enter stage 4, hence $y_{3,1,4}=1$. The binary $f_{m_2,m_1,s}$ is set to 1 when model m_2 is blocked by model m_1 at stage s. For instance, in Figure 11(b), model 4 ($m_2=4$) is blocked by model 3 ($m_1=3$) at stage 2 ($s=2$), meaning that model 3 must depart stage 2 so that model 4 can enter it, therefore, $f_{4,3,2}=1$. Similarly, model 4 is blocked by model 2 at stage 3, hence, $f_{4,2,3}=1$. Lastly, the binary $cf_{m_2,m_1,s}$ is set to 1 when model m_2 is cyclically blocked by model m_1 at stage s, meaning that m_2 can only enter stage s when model m_1 from the previous MPS departs from it. For instance, in Figure 11(b), model 1 ($m_2=1$) is cyclically blocked by model 4 ($m_1=4$) at stage 1 ($s=1$), therefore, $cf_{1,4,1}=1$. Similarly, model 1 is cyclically blocked by itself in stage 3, hence, $cf_{1,1,3}=1$, meaning it can only enter stage 3 when the model 1 piece of the previous MPS departs from it. The Minimal Part Set is assumed to contain one model of each type ($|M|$ pieces, in total), and different demand rates can be represented by having multiple models with the same processing times. For the sake of simplicity, the model presented in this section assumes that k_s is lower or equal to $|M|$ for all stages s, and a generalization of this model is presented in 4.6 removing such hypothesis.

Minimize CT (40)

Subject to:

$$\sum_{s \in S} x_{t,m,s} = 1 \qquad \forall\, m \in M, t \in T_m \qquad (41)$$

$$\sum_{t \in T} Q_{t,s} \cdot \max_{m \in M} x_{t,m,s} \leq A_s \qquad \forall\, s \in S \qquad (42)$$

$$\sum_{s \in S} s \cdot x_{t_1,m,s} \leq \sum_{s \in S} s \cdot x_{t_2,m,s} \qquad \forall\, m \in M, (t_1,t_2) \in R_m \qquad (43)$$

$$Tx_{m,s} = \sum_{t \in T_m} D_{t,m,s} \cdot x_{t,m,s} \qquad \forall\, m \in M, s \in S \qquad (44)$$

$$CT \geq \sum_{m \in M} Tx_{m,s}/k_s \qquad \forall\, s \in S \qquad (45)$$

$$\sum_{m \in M} y_{m,n,s} = 1 \qquad \forall\, n \in M, s \in S \qquad (46)$$

$$\sum_{n \in M} y_{m,n,s} = 1 \qquad \forall\, m \in M, s \in S \qquad (47)$$

$$y_{m,n,s+1} \leq \sum_{n' \in M : n' \leq n+k_s-1} y_{m,n',s} \qquad \forall\, m \in M, n \in M, s,s+1 \in S \qquad (48)$$

$$f_{m_2,m_1,s} \geq y_{m_2,n+k_s,s} + y_{m_1,n,s+1} - 1 \qquad \forall\, s,s+1 \in S, m_1, m_2, n \in M : n + k_s \leq |M| \qquad (49)$$

$$cf_{m_2,m_1,s} \geq y_{m_2,n-|M|+k_s,s} + y_{m_1,n,s+1)} - 1 \qquad \forall\, s,s+1 \in S, m_1, m_2, n \in M : n + k_s > |M| \qquad (50)$$

$$Tout_{m,s} \geq Tin_{m,s} + Tx_{m,s} \qquad \forall\, m \in M, s \in S \qquad (51)$$

$$Tout_{m,s-1} = Tin_{m,s} \qquad \forall\, m \in M, s \in S : s > 1 \qquad (52)$$

$$Tin_{m_2,s} \geq Tout_{m_1,s} - H \cdot (1 - f_{m_2,m_1,s}) \qquad \forall\, m_1, m_2 \in M, s \in S \qquad (53)$$

$$CT + Tin_{m_2,s} \geq Tout_{m_1,s} - H \cdot (1 - cf_{m_2,m_1,s}) \qquad \forall\, m_1,m_2 \in M, s \in S \qquad (54)$$

Expression (40) states the problem's goal function, i.e. cycle time minimization. Equation (41) states the balancing occurrence constraint: each task must be assigned for a stage for each product model. Inequality (42) states that the space required for the tasks assigned to a stage[2] is limited by the space available in it (parameter A_s). Notice that models will not

[2] The $\max$ function in Constraint (42) is non-linear, but also easily linearized. Furthermore, the function is available and automatically linearized by OPL ILOG CPLEX.

necessarily contribute equally to the space constraints for all tasks: each model m has its own set of tasks (T_m) and, therefore, the occurrence constraint (Equation (41)) will not affect some balancing binary variables ($x_{t,m,s}$). Expression (43) states the precedence constraints between tasks. With the balancing variables, processing times for each model at each stage are determined by Constraint (44), and a lower bound is defined for cycle time by Expression (45): the total processing time at a stage divided by the parallelism factor. Constraints (46) and (47) control the ordering variables, demanding that each model be in a single position and each position contain a single model. Constraint (48) imposes a consistency constraint in ordering variables of neighboring stages: e.g. With a parallelism degree $k_s = 2$, if model m is the n^{th} to leave the stage s (and enter stage $s+1$) it must have been at most the $(n+1)^{\text{th}}$ to enter the stage s. Notice that if $k_s = 1$ then the order in which each model m enters in a stage is the same order it departs from the stage. Constraint (49) and (50) control the blockage and cyclical blockage binary variables sets f and cf in accordance to the boundary-wise cyclical concept described at Section 4.2.1. Constraint (51) requires each model's processing to be completed at each stage before departing it. Constraint (52) states that when a model leaves a stage it immediately enters the next one. Constraint (53) states that if model m_2 is blocked by model m_1 at stage s, m_2 can only enter stage s after model m_1 has departed from it, these blockages occur within an MPS. Constraint (54) controls the cycle time by tying the analogous scheduling blockages between MPS. In Constraints (53) and (54), the relaxation factor H can take any valid value of CT multiplied by the highest parallelism factor k_s. The fact that sequencing and blockage variables are allowed to be different for each stage s and the constraints that relate to them (Constraints (46) to (50), (53), and (54)) are the key factors that extend the formulation proposed in Chapters 2 and 3 by removing the ordering hypothesis (requiring the entry orders and departure orders be the same) from, and hence generalizing, the model presented by Lopes *et al.* (2017).

The additional formulation (three sequencing variable sets: y, f and cf) complexity required to adequately represent the additional liberties of parallel stations poses a question: is it possible to simply define sequencing variables for the first boundary (line entrance) and to simulate the line's behavior based on common and intuitive scheduling rules? While this approach is feasible (TIACCI, 2015a), it is not guaranteed to lead to optimal cyclical schedules even when an optimal entry order is employed. This fact is demonstrated by an example provided in 4.4.2.

4.4 RESULTS

The model described in Section 4.3 was applied to the 36-instance data set presented by Öztürk *et al.* (2015). The summary of most relevant instance information and results are reported by Table 14. The proposed model results are reported under the "Full" column, its previous particularized Lopes *et al.* (2017) form is reported under the "Part." column, and the best solution found by Öztürk *et al.* (2015)'s model is reported under the benchmark (BMK) column. Table 14 compares the cycle time values of the best incumbent found by each method within the time limit. The proposed model was only solved to optimality for instances marked with 0% Gap (except 22 and 28). The particularized model was solved to optimality for all instances, except the largest three (33-36). Öztürk *et al.* (2015)'s benchmark, however, did not report optimality in any instance. This may be due to the model's nature: Öztürk *et al.* (2015)'s machine-wise formulation is a Constraint Logic Programming model; That approach lacks a systematic lower bound and might have difficulties in fully exploring large search fields.

In order to set aside software and hardware considerations, all models were executed in the same hardware and software conditions: Tests were performed on a Core i7-3770 CPU (3.4GHz) with 16 GB RAM with a time limit of one hour; The universal solver CPLEX v12.8 was used and the original implementation files provided by Öztürk *et al.* (2015) were employed. Cycle time comparisons are made between the upper bound of the proposed model and the average of model-wise cycle times reported by Öztürk *et al.* (2015) implementation files. Models, data, and solution files are provided by Lopes *et al.* (2019)'s Supplementary Material.

For instances with parallelism that were not solved to optimality, a relaxation of the proposed model was also used to strengthen the lower bounds: By removing the sequencing and scheduling variables and constraints, the reduced model (Expressions (40) to (45)) states a balancing lower bound for stable cycle time. The lower bound values reported by Table 14 are the maximum of the best bound found by the proposed model and the optimal answer of the relaxed model. For instances without parallelism, the lower bounds reported by the particularized version of the model Lopes *et al.* (2017) are valid and, hence, were also used to define the best lower bounds (column LB of Table 14).

For most instances (all except 1, 4, 10, and 25, in which all formulations tied), the proposed model (and its particularized version) outperformed the benchmark (ÖZTÜRK *et al.*, 2015). Out of the 36 instances, 18 were solved to optimality. However, most of them (11) are

Table 14 – Results summary for the 36-instance dataset

Instance info (Parameters)					Solution info (Cycle Time) - T.U.				
i	$\sum_m \lvert T_m \rvert$	$\lvert M \rvert$	$\lvert S \rvert$	k_s	Full	Part.	BMK	LB	Gap
1	38	5	3	1	47	47	47	47	0%
2	38	5	3	2	23.5	23.5	23.8	23.5	**0%**
3	38	5	3	3	15.7	15.7	18	15.7	**0%**
4	38	5	5	1	26	26	26	26	0%
5	38	5	5	2	**12**	13	14.8	12	**0%**
6	38	5	5	3	**8**	8.7	9.8	8	**0%**
7	52	7	3	1	57	57	58	57	0%
8	52	7	3	2	**28**	28.5	31.1	26.5	5.4%
9	52	7	3	3	**18**	19	24.7	17.7	1.7%
10	52	7	5	1	34	34	34	34	0%
11	52	7	5	2	17	17	20.4	16	5.9%
12	52	7	5	3	12	**11.3**	13.1	10.7	5.3%
13	114	5	3	1	107	107	121	107	0%
14	114	5	3	2	**52**	53.5	62.8	52	**0%**
15	114	5	3	3	**34.7**	35.7	40.4	33	4.9%
16	114	5	5	1	68	68	73.4	68	0%
17	114	5	5	2	**32**	34	36	30.5	4.7%
18	114	5	5	3	**22.3**	22.7	24.6	20.3	9%
19	156	7	3	1	142	142	149	142	0%
20	156	7	3	2	**70**	71	73.9	67.5	3.6%
21	156	7	3	3	48	**47.3**	61.4	45	4.9%
22	156	7	5	1	89	**88**	102	88	0%
23	156	7	5	2	46	**44**	61	41	6.8%
24	156	7	5	3	30	**29.3**	32.7	27.3	6.8%
25	190	5	3	1	197	197	197	197	0%
26	190	5	3	2	98.5	98.5	103.6	98.5	**0%**
27	190	5	3	3	**64.3**	65.7	80.4	64.3	**0%**
28	190	5	5	1	114	**113**	116	113	0%
29	190	5	5	2	**56**	56.5	63.2	50	10.7%
30	190	5	5	3	**36**	37.7	46.4	33.3	7.5%
31	260	7	3	1	263	263	267	263	0%
32	260	7	3	2	**131**	131.5	139	126	3.8%
33	260	7	3	3	**86**	87.7	104.1	84	2.3%
34	260	7	5	1	156	**155**	165	135	12.9%
35	260	7	5	2	**76**	77.5	87.3	67.5	11.2%
36	260	7	5	3	53	**51.7**	57.1	45	13%

instances without parallelism ($k_s = 1$). This highlights the additional complexity tied to parallel stations: sequencing variables have more flexibility as the order models enter and depart stages is not necessarily the same.

Comparing the proposed model (Full) to its particularized version (Part.) Lopes *et al.* (2017), it is clear that the additional flexibility (removal of the ordering hypothesis) comes with a trade-off convergence difficulty: In 15 instances (boldfaced in column Full), removing the ordering hypothesis leads to better answers than the particularized version. All these answers are unfeasible for the particularized model due to the ordering hypothesis. However, in 8 instances (boldfaced in column Part.), the particularized version of the model found better solutions, all of which are feasible for the generalized version - these were not found due to time limit. Overall, the additional flexibility of the generalized model did produce improvements: fifteen new best answers (boldfaced in column Full), four of which are optimal ones (instances 5, 6, 14, and 27), and three new optimality proofs (instances 2, 3, and 26).

4.4.1 Parallelism influence

A close analysis of the input data provided by Öztürk *et al.* (2015) revealed that instances are equal in groups of three, differing only by the parallelism degree: Instances 1-3 are identical except for k_s, this is also the case for instances 4-6, 7-9, etc. Furthermore, the k_s parameter is constant and equal for all stages of the same instance. This means that the performance of instances in the same group can be normalized if their cycle time values are multiplied by k_s. This comparison then allows the verification of whether or not station parallelism (cross-overs between parallel lines) allows better performance than to simply have parallel identical and independent lines. The results of the comparisons are summarized by Table 15, in which the best answers for each instance is used to generate the normalized cycle time values, and optimal answers are underlined.

Table 15 – Normalized Cycle Time Comparisons

Instance	Normalized Cycle Times - T.U.			Improvements	
Group	k_s = 1	k_s = 2	k_s = 3	2 vs 1	3 vs 1
1-3	47	47	47	0%	0%
4-6	26	**24**	**24**	7.7%	7.7%
7-9	57	**56**	**54**	1.8%	5.3%
10-12	34	34	34	0%	0%
13-15	107	**104**	**104**	2.8%	2.8%
16-18	68	**64**	**67**	5.9%	1.5%
19-21	142	**140**	142	1.4%	0%
22-24	88	88	88	0%	0%
25-27	197	197	**193**	0%	2%
28-30	113	**112**	**108**	0.9%	4.4%
31-33	263	**262**	**258**	0.4%	1.9%
34-36	155	**152**	155	1.9%	0%
			Average	1.9%	2.1%
			Maximum	7.7%	7.7%

Notice that in 15 out of 24 cases with parallelism the solution found by the model is better than k_s times the solution without parallelism ($k_s = 1$). In average, this improvement rate is close to 2%, and in the best case it is 7.7%. This demonstrates that parallel workstations can offer an increasing marginal value in the following specific sense: the use of twice (three times) as many resources lead to results better than double (triple) that of those obtained by the base quantity. This is due to the fact that parallelism allows more scheduling flexibility: entry and departure orders at each stage are allowed to differ. Notice that an analogous increasing marginal value of workstations is not possible for a serial line in a Simple Assembly Line Balancing (SALB) context: doubling (tripling) the number of serial single stations can at most half (reduce to a third) the line's cycle time. This fact is hereby proved by contradiction: suppose a solution with $n \cdot k$ stations exists with cycle time $CT_{n \cdot k}$ lower than $1/k$ the optimal cycle time CT_n for the line with n stations ($CT_n > k \cdot CT_{n \cdot k}$). By merging task-stations assignments of the solution

in groups of k stations (1 ... k,$k+1$ ... $2k$, etc.), a feasible (precedence relations are respected) balancing solution for n stations is produced with $CT_n \leq k \cdot CT_{n \cdot k}$.

In 14 out of these 15 cases (all except instance 35), the proposed formulation outperformed the lower bound obtained by its particularized version (LOPES *et al.*, 2017) tied to the ordering hypothesis (imposing entry order equal to departure order at every stage). These results show that the proposed model is capable of taking advantage of parallelism increased flexibilities, and that the imposition of the ordering hypothesis can prune part of the valid search field.

4.4.2 Comparison to Completion-Based Priority Rules

Multiple recent works have employed simulation techniques (TIACCI; MIMMI, 2018; TIACCI, 2015b; TIACCI, 2015a; TIACCI, 2017) to measure performance. These are based on a simulator (TIACCI, 2012) that employs common priority rules for sequencing and scheduling decisions: pieces move to the next station as soon as they can, and priority is given to the first piece to be completed in case more than one can move. These common scheduling rules are hereafter referred to as Completion-Based Priority Rules (CBPR). It might seem at first that a simulation approach based on CBPR (such as the simulator proposed by Tiacci (2012)) would necessarily lead to the optimal steady-state given a cyclic entry order in the flowshop. This does work for lines without parallelism as simulations gradually converge towards the steady-state, as shown in Chapter 2. However, that is not necessarily true when parallelism is present. In order to demonstrate the necessity of multiple sets of sequencing variables (one for each boundary), an illustrative example is hereby given.

Consider the cyclical scheduling problem associated to the processing time data presented by Table 16. Consider an MPS with one unit of each of the three product models and consider that both stages have a parallelism degree of two.

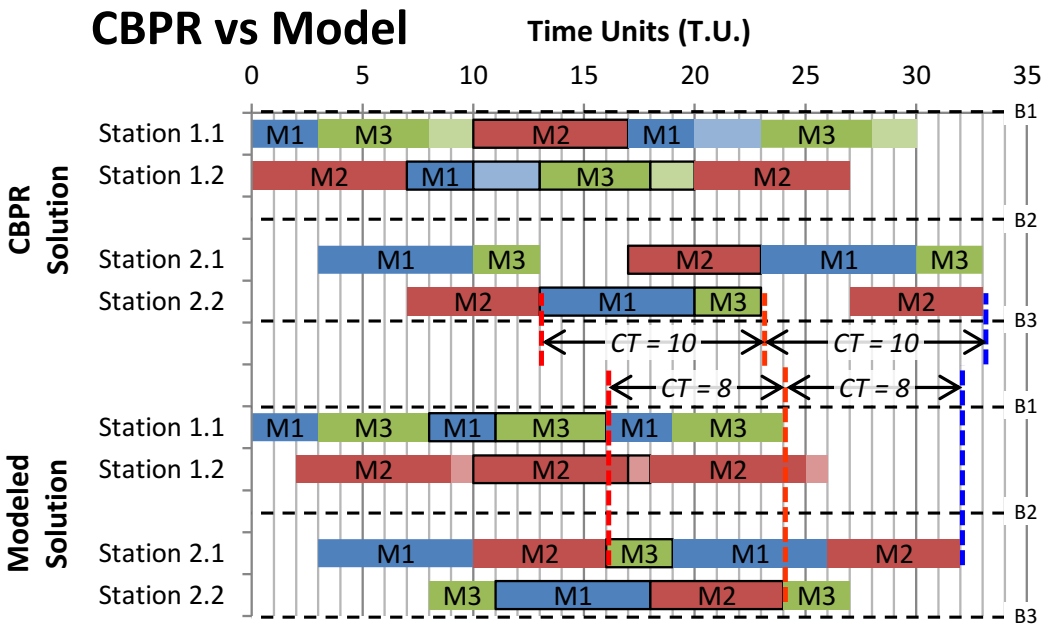
Table 16 – CBPR vs. Model example processing time data (Time Units - T.U.)

Stage	S1	S2
Model 1	3	7
Model 2	7	6
Model 3	5	3

Assuming the cyclic entry order (M1,M2,M3) at least two different cyclical schedules are possible: one by simulating the piece flow based on the CBPR rules, i.e. by having pieces move to the next stage as soon as they can with priority to pieces that are completed first; the other by using the proposed model and allowing sequencing variables at each remaining boundary,

i.e. by fixing only those tied to the first entrance boundary. These solutions are presented by Figure 12. In it, even MPSs are highlighted with thicker borders and B1, B2, and B3 stand for Boundary 1, 2, and 3, respectively.

Figure 12 – Comparison between proposed formulation and CBPR with given cyclical entry order



Source: Lopes *et al.* (2019)

Notice that the modeled solution has a cycle time of 8 T.U., while the CBPR one has a cycle time of 10 T.U.. The 2 T.U. waiting time before the entrance of M2 at the first MPS replication leads to a higher makespan value for the first MPS for the modeled solution (16 vs 13). However, due to the lower value of cycle time, this difference decreases steadily after each replication of the minimal part set. After the third replication, the better cyclical behavior dominates the worse transient one, meaning that the modeled solution has a better makespan as well (32 vs 33). The dashed colored lines in Figure 12 highlight these makespan differences.

This example demonstrates that a CBPR approach that only takes into account the entry order at the first stage is insufficient as a cyclical performance measurement: while CBPR simulations can generate cyclical schedules, they are not guaranteed to be able to lead to the best possible one. By induction, this argument demonstrates the importance of sequencing variables at every boundary.

4.5 CONCLUSIONS

Mixed-model assembly lines are often employed in industrial contexts as they balance product diversity and production efficiency. Balancing workloads (task distribution) and cyclically scheduling these lines are key to obtain a high and stable throughput. In the context of unpaced asynchronous lines in particular, blockages and starvations must be taken into account to optimize the flowshop. Parallel workstations offer further flexibility to the line by allowing stations to act as partial buffers and by enabling more sequencing and scheduling possibilities.

Authors thus far have defined cyclical schedules of flowshops in general (and assembly lines in particular) by a machine-wise cyclical concept. In that concept, the cycle time is defined as the time between equivalent events of different minimal parts sets at each machine. However, by defining boundaries between stages, as indicated by Figure 9, better and more general cyclical schedules are possible when a boundary-wise cyclical concept is employed. This formulation is introduced in this chapter, both conceptually (Section 4.2.1) and formally (Section 4.3). This formulation extends Chapters 2 and 3's simple station (no parallelism) ones, and generalizes another that incorporated parallel stations (LOPES *et al.*, 2017) by removing a simplifying hypothesis. An illustrative example (Figure 11) is provided to demonstrate the higher flexibility obtained by the boundary-wise cyclical concept. While for assembly lines without parallelism the machine-wise and boundary-wise formulations are the same, this is shown not to always be the case for assembly lines with parallelism. The proposed formulation is also shown to allow better schedules than those obtained by scheduling with common priority rules (often employed by simulation-based approaches) the assembly line with a given entry order (Section 4.4.2).

A mixed-integer linear programming model is presented to solve the asynchronous mixed-model assembly line simultaneous balancing and cyclical scheduling problem. The proposed formulation is capable of representing cyclical schedules for any number of stages and different numbers of parallel workstations in each stage. Tests on a 36-instance literature benchmark provided new best-known solutions for 15 instances, with 4 new optimal ones, and 3 new optimality proofs, as summarized by Table 14. Furthermore, by analyzing instances that only differed by the parallelism degree it was possible to show that unpaced lines with parallel stations can outperform parallel unpaced lines with single stations (Table 15): Cycle times found for instances with parallelism are, in average, 2% better than cycle times found for the equivalent instances without parallelism divided by the parallelism degree. This difference is as high as

7.7% for one instance in particular. The results summarized in Table 15 demonstrate that, aside from allowing higher recovery from failure capacity, parallel workstations can allow a crescent marginal value: it might be possible to more than double (triple) the production rate by doubling (tripling) the amount of resources.

4.6 FORMULATION EXTENSION

The MILP model presented in Section 4.3 is built under the hypothesis that the parallelism degree k_s is smaller or equal than the number of pieces in the minimal part set $|M|$ at every stage s. This might, however, not be the case: in some very specific environments, very long tasks might require degrees of parallelism k_s to be higher than $|M|$ for a specific stage.

The generalization of the proposed model for such cases is based on the stage-wise correction factor j_s, whose value is defined by the Equation (55) for each stage s. This factor states the smallest integer degree to which k_s is greater than $|M|$. For instance, with 5 pieces in the MPS, j_s is defined as 0, 1, 2, respectively, when k_s belongs to the ranges $\{1,... ,5\}$, $\{6,... ,10\}$, $\{11,... ,15\}$.

$$j_s \;=\; \left\lfloor \frac{k_s - 1}{|M|} \right\rfloor \qquad \forall\, s \;\in\; S \tag{55}$$

With the correcting factor j_s, four expressions must be revised: Constraints (56) and (57) replace Constraints (49) and (50); and Constraints (58) and (59) replace Constraints (53) and (54). Notice that when j_s is zero (i.e. the hypothesis $k_s \leq |M|$ holds) the reviewed constraints are the same as the ones presented in Section 4.3.

$$f_{m_2,m_1,s} \geq y_{m_2,n+k_s-j_s\cdot|M|,s} + y_{m_1,n,s+1} - 1$$
$$\forall\, s,s+1 \;\in\; S, m_1, m_2, n \;\in\; M : n + k_s - j_s \cdot |M| \leq |M| \tag{56}$$

$$cf_{m_2,m_1,s} \geq y_{m_2,n-(j_s+1)\cdot|M|+k_s,s} + y_{m_1,n,s+1} - 1$$
$$\forall\, s,s+1 \;\in\; S, m_1, m_2, n \;\in\; M : n + k_s - j_s \cdot |M| > |M| \tag{57}$$

$$CT \cdot j_s + Tin_{m_2,s} \geq Tout_{m_1,s} - H \cdot (1 - f_{m_2,m_1,s})$$
$$\forall\, m_1,m_2 \;\in\; M, s \;\in\; S \tag{58}$$

$$CT \cdot (j_s + 1) + Tin_{m_2,s} \geq Tout_{m_1,s} - H \cdot (1 - cf_{m_2,m_1,s})$$

$$\forall\, m_1,m_2 \in M, s \in S \quad (59)$$

With the correcting factor j_s, four expressions must be revised: Constraints (56) and (57) replace Constraints (49) and (50); and Constraints (58) and (59) replace Constraints (53) and (54). Notice that when j_s is zero (i.e. the hypothesis $k_s \leq |M|$ holds) the reviewed constraints are the same as the ones presented in Section 4.3.

5 INCORPORATING BUFFER ALLOCATION

This chapter is based on an article published in the International Journal of Production Research (LOPES *et al.*, 2020a). In it, the balancing-sequencing formulation described in Chapter 3 is further extended to also incorporate buffer allocation, resulting in a model that is the first in the literature to combine three degrees of liberty. An iterative decomposition is proposed to solve this problem.

5.1 CONTEXT

Assembly lines are product oriented production layouts commonly employed for large-scale manufacturing of similar products (SCHOLL, 1999). While early versions of such lines were dedicated to a single product model, the increase in product diversity led to mixed-model assembly lines, which are shared between a set of similar products (BOYSEN *et al.*, 2009b). Substantial research has been conducted on the optimization of assembly lines, ranging from the simpler more theoretical cases (SCHOLL; BECKER, 2006) to more general ones (BECKER; SCHOLL, 2006) that remove some typical simplifying hypothesis. The focus of this chapter is asynchronous mixed-model assembly lines, an unpaced variant in which each product moves to the next station when two conditions are met: processing at the current station is complete, and the next station is available (BOYSEN *et al.*, 2007). In these lines, three degrees of freedom are relevant: balancing, sequencing, and buffer allocation (BOYSEN *et al.*, 2008).

The first problem, assembly line balancing, consists of distributing a set of tasks amongst stations in a way that maximizes the lines' efficiency. Stations are sequential in nature and, usually, equally manned. Tasks have specific durations and precedence relations, meaning that some must be performed before others. Thus, precedence relations restrict task assignment. The sum of task processing times in each station is tied to the line's production rate. If only one product model is produced, the highest station-wise sum dictates the line's cycle time. When more than one product is produced in the same line, other factors such as line pace and product sequence become important. The line balancing problem was first introduced by Salveson (1955) and was recently object of a review by Battaïa and Dolgui (2013). The most common goals are the minimization of the number of stations given a minimal production rate or the minimization of the cycle time given the number of stations. In the particular case of asynchronous mixed-model

assembly lines, the production rate may be difficult to determine (TIACCI, 2015b), and may be a function of line balancing, product sequence, and buffer allocation.

The second problem, mixed-model sequencing, consists of defining the order in which product models are to be produced.

Product models usually have known processing times in each station and given demand rates. While virtually any product sequence is technically feasible (BOYSEN *et al.*, 2009c), the order in which products are produced affects performance measures with relevant economic impacts. Cycle time is usually considered a parameter for sequencing problems. However, in throughput maximization variants, mixed-model sequencing can be an important optimization aspect, especially when combined to line balancing.

Thomopoulos (1967) was one of the earliest authors to study mixed-model sequencing problem, and many variants have been defined since then (BOYSEN *et al.*, 2009c). In the particular context of relatively stable demands, cyclical scheduling is often employed to obtain a high and stable throughput (KARABATI; SAYIN, 2003; SAWIK, 2012). Cyclical scheduling is often based on the Minimal Part Set (MPS) concept (LEVNER *et al.*, 2010), which is defined as the smallest set of products that can be repeated indefinitely in order to meet the production target. For example, if the production target is 5000 units of product model 1 and 3000 units of product model 2, then the MPS is 5 units of product model 1 and 3 units of product model 2, which can be represented by the vector (5, 3).

The third problem, buffer allocation, consists of assigning internal storage spaces in order to increase the system's productivity.

Buffers are units with the capacity of storing incomplete products between stages in a production system. Their presence can help prevent starvations and blockages, therefore, increasing throughput. Naturally, they imply costs. Buffer allocation problems usually assume that factors such as system design and workload distributions have already been solved (DEMIR *et al.*, 2014). However, in the context of throughput maximization of mixed-model assembly lines, Chapter 2 shows that buffers can affect the optimal workload distribution.

The buffer allocation problem was first formalized by Koenigsberg (1959) and was recently reviewed by Demir *et al.* (2014). Common optimization goals are the minimization of the number of buffers required to meet a certain production rate, and the maximization of the production rate given the number of buffers.

Combining these degrees of freedom should lead to better results than optimizing them

independently or sequentially (BOYSEN *et al.*, 2007; BOYSEN *et al.*, 2008). However, most works focus separately on each of these degrees of freedom (BOYSEN *et al.*, 2009c; BATTAÏA; DOLGUI, 2013; DEMIR *et al.*, 2014). To the best of the authors' knowledge, no work has yet combined the three degrees of freedom to obtain an optimized solution as the approach herein presented. However, some papers combined two of these degrees, in particular balancing and sequencing. The strategies adopted by most authors when combining these degrees of freedom and optimizing mixed-model assembly lines are presented in Section 5.2.

In this chapter, the number of buffers and workstations are considered as problem parameters, i.e. finite available resources. The goal is to determine a line balancing, a cyclical product sequence, and a buffer allocation that maximizes the line's throughput. The formulation presented in Chapter 2 is extended to incorporate sequencing and buffer allocation decisions: a **new MILP model** is developed for mixed-model assembly lines, in which one is required to combine the three degrees of freedom to optimize the steady-state throughput. Therefore, the problem at hand is a mixed-model assembly line **B**alancing, **C**yclical **S**equencing and **B**uffer Allocation **P**roblem (hereafter abbreviated as **BCSB-P**). The three sub-problems are shown to be interconnected, and the presented MILP model presents a mathematical description of such connections. Due to the computational difficulties that arise from the combination of these degrees of liberty, an **Iterative Decomposition** procedure is presented. The decomposition is tested against the proposed monolithic model, proving to be very competitive. Both the decomposition and the model are also compared to a **sequential procedure** based on literature procedures for similar problems (SAWIK, 2004; BATTINI *et al.*, 2009).

The chapter is structured as follows. Section 5.2 provides a brief overview of recent related works. In Section 5.3 a context is yielded and the definition of a BCSB-P is explained. Section 5.4 presents the model that describes a cyclical steady-state optimization problem. Section 5.5 presents the proposed iterative decomposition. Section 5.6 presents the results of the comparisons between the monolithic model and the proposed decomposition. Lastly, the main conclusions drawn from this chapter are summarized in Section 5.7.

5.2 RELATED WORKS

Works on optimization of assembly line balancing, sequencing, related and combined problems employ a large range of methods (mathematical models, decompositions, meta-heuristics, etc.) to solve multiple variations of problems. The main focus of this section is

not on the solution method applied, but rather on the problem's decision variables (balancing, sequencing, buffer allocation), and on the throughput performance measure used, i.e. the role cycle time plays in the problem's definition.

Most works that combine some of the three aforementioned degrees of freedom (balancing, sequencing, and buffer allocation) focus on variants of the balancing-sequencing combination. However, the role of cycle time as a performance definition and goal function varies substantially: Sawik (2002), Alghazi and Kurz (2018) employ the weighted average (or sum) of model-wise processing times in each station as a performance requirement.

Multiple authors consider as goal functions a measure of processing time deviations from an ideal average value: Kim *et al.* (2000), Kim *et al.* (2006) apply such deviations as goal functions of a combined balancing-sequencing problem and Battini *et al.* (2009) use them as a goal function in a decomposition step in a problem that seeks to maximize production rates and minimize buffer requirements.

Workload smoothing is a performance measure that is also often employed along with other goal functions such as minimizing the number of workstations: Özcan *et al.* (2010) apply it for a problem definition with parallel assembly lines, and Hamzadayi and Yildiz (2012), Hamzadayi and Yildiz (2013) use it for U-lines with and without parallel workstations.

Some balancing-sequencing works on continuous lines with utility work minimization goal also consider cycle time to be a parameter that affects the goal function: Kim *et al.* (2000) present a base formulation for straight mixed-model lines, Mosadegh *et al.* (2012) present a problem variant with station-dependent assembly times and Nilakantan *et al.* (2017) present a formulation for two-sided lines. Defersha and Mohebalizadehgashti (2018) present a balancing-sequencing formulation for mixed-model paced lines with the combined goal of minimizing line length, task duplications, and number of stations. In their formulation, time between products (cycle time) is also considered a parameter. Another common goal function for balancing-sequencing problems is makespan minimization for unpaced asynchronous lines (ÖZTÜRK *et al.*, 2013; ÖZTÜRK *et al.*, 2015). Biele and Mönch (2018) presented a cost-oriented balancing problem for when (non-cyclical) product sequence is given.

Many authors also consider cycle time as a parameter that must be respected by all product models in the line, regardless of product sequence: Zhong (2017) applies this definition for hull assembly line balancing (shipbuilding); Akpinar and Baykasoglu (2014a), Akpinar *et al.* (2017) employ this definition for mixed-model problems with set-ups; Delice *et al.* (2017)

uses it for two-sided assembly lines; Roshani and Nezami (2017) apply it for multi-manned lines; Kucukkoc and Zhang (2016) uses it for complex parallel lines variants; Dong *et al.* (2018) employs a chance-constraint version of this consideration for stochastic lines. The conversion of this parameter into a problem variable and its minimization is also a studied strategy (KARABATI; SAYIN, 2003).

Tiacci (2015a), Tiacci (2015b), Tiacci (2017) has presented a simulation-based performance evaluation to have a direct performance measure for balancing and buffer allocation on asynchronous stochastic assembly lines. Tiacci (2015a) argues that the above mentioned indirect performance measures (weighted average processing times, workload smoothing, workload deviations minimization, etc.) are not goals in themselves, but rather supposed means to achieve a high and stable throughput. This is corroborated by Chapter 2, which presented a formulation for direct steady-state performance measure in deterministic asynchronous lines with given cyclical product sequence and given buffer allocation. Chapter 2 also compares its formulation to indirect performance measures and confirmed that they do not necessarily optimize steady-state. Chapters 3 and 4 extend that formulation and incorporate sequencing decisions and parallel workstations, but do not take buffer allocation into account.

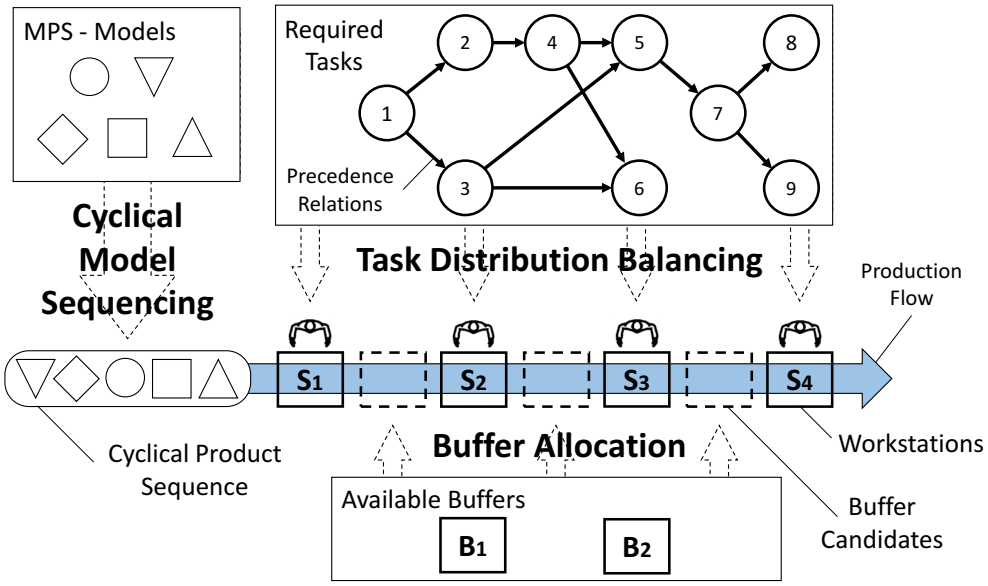
While Tiacci (2015b) presented a balancing and buffer allocation optimization procedure, its simulation-based procedure considers a random product sequence and does not include a mathematical model with decision variables and constraints. Chapter 3's model, on the other hand, does not incorporate buffer allocation decisions. To the best of the authors' knowledge, no work has addressed the combined BCSB-P: Both the direct performance measure and the combination of balancing, sequencing, and buffer allocation degrees of freedom remain a challenge and a gap in the literature.

5.3 PROBLEM STATEMENT

The optimization of a cyclical asynchronous assembly line requires the combination of three degrees of freedom: balancing the assignment of tasks to stations; assigning some buffers to the line; and cyclical sequencing and scheduling of product models in the Minimal Part Set (MPS). Figure 13 conceptually depicts the studied BCSB-P. In this figure, the models M1, M2, and M3 (represented by circles, squares and triangles) are cyclically sequenced to enter to and depart from the line in a repeated pattern. Workstations (S1-S4) perform the required tasks, and available buffers (B1 and B2) are to be placed in two out of the three candidate spots. In this

chapter, the number of workstations and buffers is considered a parameter, and the goal is to maximize the line's productivity, i.e. a type-2 problem (SCHOLL, 1999). A related problem can be defined as the minimization of the number of workstations and buffers given a maximum steady-state cycle time.

Figure 13 – Overview of the optimization problem



Source: Lopes *et al.* (2020a)

Line balancing must respect a set of precedence relations between tasks. If precedence relations are different across models, they can be combined into a joint precedence graph during pre-processing (BOYSEN *et al.*, 2009a). Furthermore, without loss of generality, tasks are assigned to the same station for all models: If a task can be performed at different stations for different product models, it can be replaced by a set of tasks with zero processing times for all but one model each (BOYSEN *et al.*, 2009a). For instance, in a line with 2 product models, consider a task t with durations of 7 and 9 time units for models 1 and 2, respectively. If t can be performed at different stations for each model, then it can be replaced by two tasks: t_a with duration of 7 and 0 time units, and t_b with duration of 0 and 9 time units. These tasks would have the same precedence relations as t.

A finite number of buffers must be assigned between stations. They help compensate model-wise deviations of processing times (BOYSEN *et al.*, 2008) by temporarily storing pieces between stations. In this chapter, dummy stations called buffer candidates are added between stations to represent potential buffer allocation positions: a buffer candidate is only able to store product pieces when a buffer is assigned to that position.

The product models in the minimal part set must be sequenced in a cyclical manner. Furthermore, the discrete entry and departure times of each product piece at each workstation must be cyclically scheduled. This scheduling ties balancing, buffer allocation, and sequencing as it must reflect the possibilities and restrictions of each of the problem's degrees of freedom. Furthermore, the scheduling variables should measure the line's steady-state performance. Chapter 2 has shown how to measure such performance when product sequences and buffer layout are parameters. Such formulation must be extended to allow these aspects to become decision variables. Cyclical scheduling is NP-Hard (MCCORMICK; RAO, 1994) and hence, by assumption, the size of the minimal part set is assumed to be small enough so that cyclical scheduling given a balancing solution is quickly solved. This means that the core combinatorial difficulty of the problem lies on its balancing component, but that the evaluation of each balancing solution still requires solving a sequencing and buffer allocation subproblem.

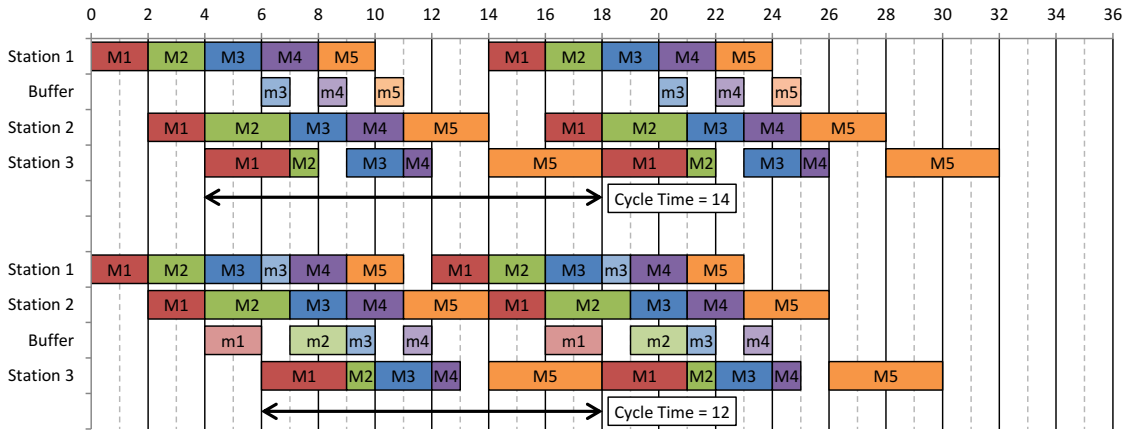
In order to illustrate the importance of each degree of freedom (balancing, sequencing, and buffer allocation) in asynchronous cyclical schedules, Figure 14 is presented. Figures 14(a) and 14(b), consider the same balancing solution, therefore, the processing times for each model at each station are constant parameters. In each of these Figures, two examples of cyclical schedules are presented with different buffer layout (Figure 14(a)) and product sequence (Figure 14(b)). Uppercase labels indicate processing times, while lowercase ones represent waiting (blocked) times. Two MPS replications are portrayed for an easier understanding of the cyclical nature of the schedules. In both Figures, arrows indicate the cycle time: the interval between entries of the first piece of each MPS at the latest station. It is clear from Figures 14(a) and 14(b) that buffer allocations and sequencing decisions can have a decisive influence on the steady-state CT value, even with a fixed set of balancing decisions. Consequently, the combination of these three degrees of freedom can provide even better solutions. Lastly, Figure 14(c) illustrates the influence of line balancing in solution quality. In it, two cyclical schedules are presented with the same sequencing and buffer allocation, but different line balancing solutions. This means different total processing time durations in each station and, therefore, different cycle time values.

5.4 MONOLITHIC MODEL

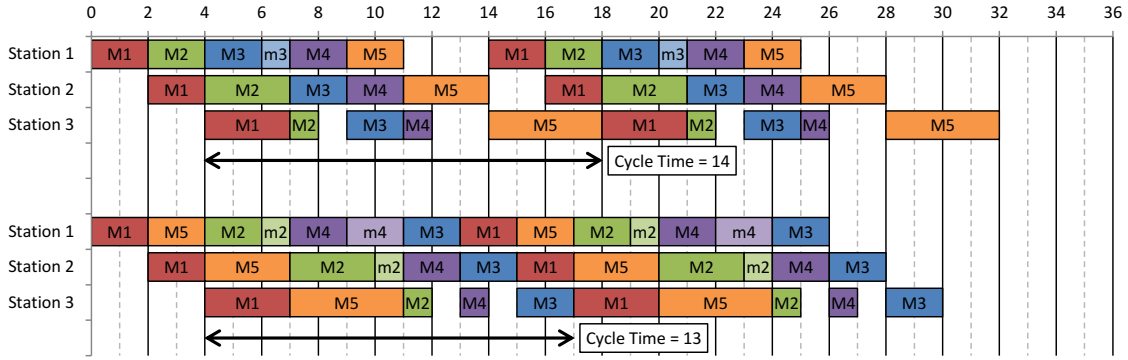
This Section presents the monolithic MILP model that combines balancing, sequencing, and buffer allocation. The problem's goal is throughput maximization, hereby considered as

Figure 14 – Examples of the influence of each degree of freedom

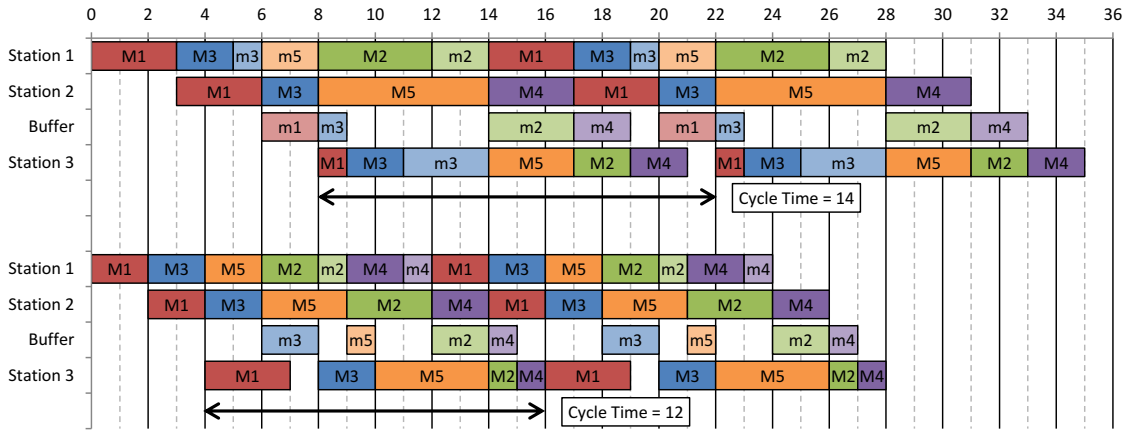
(a) Examples of Cyclical Schedules with different buffer allocations.



(b) Examples of Cyclical Schedules with different product sequences.



(c) Examples of Cyclical Schedules with different task balancing.



Source: Lopes *et al.* (2020a)

cycle time (CT) minimization. This is possible as the throughput is the inverse of the average steady-state cycle time (SCHOLL, 1999). The proposed formulation's basic concept is to produce a cyclical schedule of $|P|$ product pieces such that: processing times are determined by balancing ($x_{t,s}$) and sequencing ($y_{p,m}$) decision variables, and scheduling constraints are affected by buffer allocation (z_s) ones. Let P be a set of positions in a product sequence that repeats indefinitely.

Each (work)piece p represents a position in the cyclical product sequence. The monolithic model is defined by Expressions 60 to 72:

$$\text{Minimize} \quad CT \tag{60}$$

$$\sum_{s \in S \setminus S_b} x_{t,s} = 1 \qquad \forall\, t \in T \tag{61}$$

$$\sum_{s=1}^{s_k} x_{t_1,s} \geq \sum_{s=1}^{s_k} x_{t_2,s} \qquad \forall\, (t_1,t_2) \in R,\, s_k \in S \setminus S_b \tag{62}$$

$$\sum_{m \in M} y_{p,m} = 1 \qquad \forall\, p \in P \tag{63}$$

$$\sum_{p \in P} y_{p,m} = N_m \qquad \forall\, m \in M \tag{64}$$

$$Tx_{p,s} \geq \sum_{t \in T} D_{t,m} \cdot x_{t,s} - H \cdot (1 - y_{p,m}) \qquad \forall\, m \in M, p \in P,\, s \in S \tag{65}$$

$$\sum_{p \in P} Tx_{p,s} = \sum_{t \in T, m \in M} N_m \cdot D_{t,m} \cdot x_{t,s} \qquad \forall\, s \in S \tag{66}$$

$$Tout_{p,s} \geq Tin_{p,s} + Tx_{p,s} \qquad \forall\, p \in P,\, s \in S \tag{67}$$

$$Tout_{p,\overleftarrow{s}} = Tin_{p,s} \qquad \forall\, p \in P, s \in S :\, s > 1 \tag{68}$$

$$Tin_{p,s} \geq Tout_{p-1,s} \qquad \forall\, s \in S, p \in P :\, p > 1 \tag{69}$$

$$Tout_{p,s} \leq Tin_{p,s} + H \cdot z_s \qquad \forall\, p \in P, s \in S_b \tag{70}$$

$$\sum_{s \in S_b} z_s \leq B_{max} \tag{71}$$

$$CT + Tin_{1,s} \geq Tout_{|P|,s} \qquad \forall\, s \in S \tag{72}$$

The goal function, i.e. cycle time minimization, is stated by the Expression 60. Equation 61 states task-assignment constraint, requiring every task t to be performed at one non-buffer station s ($s \in S \setminus S_b$). Inequality 62 states the precedence relations (R), requiring that tasks t_1 be performed before their successors (t_2). Equation 63 states that a model is assigned to every position in the product sequence, i.e. to every piece. Equation 64 states the demand constraint: every product model m will be present in N_m positions in the sequence. Inequality 65 binds processing times of each piece p at each station s to balancing and sequencing decision variables, tying them to the sum of durations of tasks t assigned to station s for the product model m associated to piece p. Equation 66 defines a logical cut for processing times within each station: The sum of processing times for all pieces equals the sum of processing times for all models,

weighted by their demands. Inequality 67 states that pieces can only depart a station after entering it and being processes. Equation 68 states that a piece enters station s, when it departs from its immediate predecessor $\overleftarrow{s}$. Inequality 69 prevents two pieces from occupying the same station at the same time. Inequality 70 defines buffer allocation: If a candidate is chosen as a buffer, then pieces can stay on it after they enter it, otherwise they must depart immediately. Inequality 71 defines the buffer allocation limit. Lastly, Inequality 72 binds cycle time in each station to the elapsed time from the entry of the first piece to the departure of the last one. This proposed mathematical model, defined by Expressions 60 to 72, combines the balancing ($x_{t,s}$), sequencing ($y_{p,m}$), and buffer allocation (z_s) degrees of freedom. Therefore, it is referred to as monolithic model.

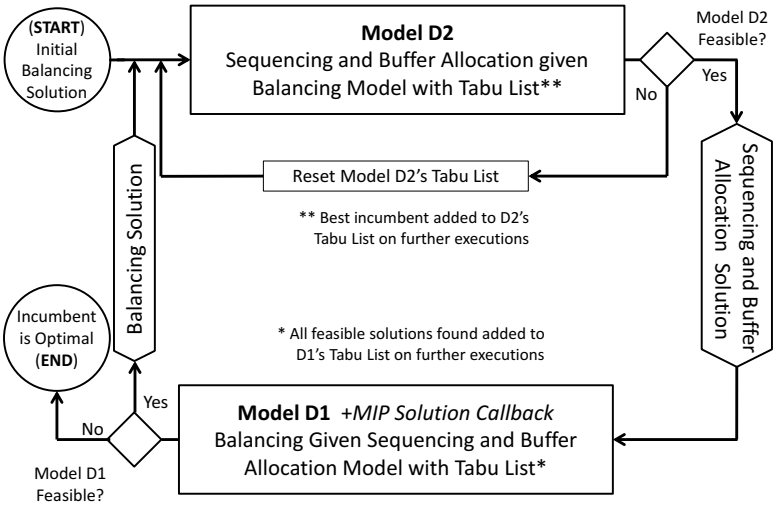
5.5 ITERATIVE DECOMPOSITION PROCEDURE

The monolithic model presented in the previous section tends to be difficult to solve, especially due to the constraints that employ the Big-M relaxation method. When separated, each degree of freedom should be faster to solve, however, to measure performance algorithmically is challenging: it is difficult to determine cycle time value of the incumbent solution ($\overline{CT}$) directly from the binary decision variables $x_{t,s}$, $y_{p,m}$, and z_{s_b} without solving a linear relaxation. Hence, in order to quickly obtain good solutions, an iterative decomposition procedure was developed. This is justified by the exact performance measurement allowed by the MILP formulation, and the expected faster exploration of search fields when part of the decision variables is fixed. Multiple authors present decompositions that first solve the (long-term) balancing problem then the associated (short-term) scheduling one (SAWIK, 2002; BATTINI *et al.*, 2009). This rationale is combined to that of the model presented in Chapter 2, in which the reverse occurs: balancing is optimized given sequencing and buffer parameters. This allows an iteration between two models, which is similar to the Fix-and-Optimize approach proposed by Helber and Sahling (2010): first the balancing variables are optimized by fixing sequencing and buffer allocation ones, then the reverse. Additionally, optimality cuts (hereafter presented) are added dynamically. These cuts are partly based on well-established graph analysis of precedence diagrams (SCHOLL; BECKER, 2006), and they function as (indirect) Combinatorial Benders' Cuts (CODATO; FISCHETTI, 2006). This type of cut consists of inequalities tied to subsystems of the linear problem and translate either optimality or feasibility requirements.

In this Section, the iterative decomposition and its notation are presented. The decom-

position is based on the monolithic model (Section 5.4): in each step of the decomposition, some variables become parameters. Figure 15 presents a high-level overview of the iterative decomposition.

Figure 15 – Iterative Decomposition Overview



Source: Lopes *et al.* (2020a)

There are two main mathematical models in the decomposition. In Model D1, the sequencing and buffer allocation variables ($y_{p,m}$ and z_s) become binary parameters. In Model D2, the balancing variables ($x_{t,s}$) are set to a fixed configuration. Therefore, Model D1 does not require Constraints 63, 64, and 71, while Model D2 does not require Constraints 61, and 62. Nonetheless, the binary parameter values given to both D1 and D2 should be feasible in regard to these constraints. The proposed method ensures feasibility by having its parameters provided by incumbent answers of MILP models that contain these constraints.

The procedure starts by defining an initial balancing solution: By assumption, the set of tasks T is topologically ordered according to the precedence relations. For instance, consider the diagram presented by Figure 13: each precedence constraints ties task pairs (t_a,t_b) such that $t_a < t_b$. This allows the initial balancing solution to be defined as follows: assign each task t to station $\lfloor 1 + (t-1) \cdot |S/S_b|/|T| \rfloor$, for example, in an instance with 9 tasks and 3 workstations, tasks 1-3 will be assigned to station 1, tasks 4-6 will be assigned to station 2, and tasks 7-9 will be assigned to station 3. Because tasks are topologically ordered in regard to precedence relations, this initial balancing solution will automatically respect the precedence relations. Balancing solutions are used by Model D2 to generate a sequencing and buffer allocation solution. Model D2's solution is used by Model D1 to generate new balancing solutions. The

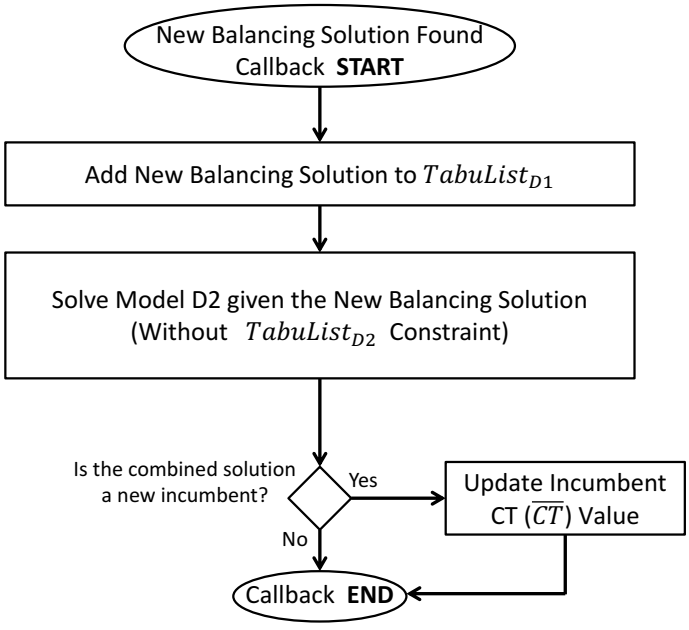
process iterates as depicted in Figure 15.

In each iteration, the best incumbent solution provided by Model D2 is added to a Sequencing and Buffer Allocation Tabu list ($\text{TabuList}_{\text{D2}}$). Let $(\text{Tabu}_{\text{Seq}},\text{Tabu}_{\text{Buff}}) \in \text{TabuList}_{\text{D2}}$ be a Tabu vector that contains the set of non-zero sequencing and buffer allocation variables, respectively. After the first time the Model D2 is run, further executions will include Constraint 73. This restriction demands at least one difference from each Tabu vector in the Tabu list, i.e., from each previously explored solution.

$$\sum_{(p,m)\ \in\ \text{Tabu}_{\text{Seq}}} y_{p,m} + \sum_{s\ \in\ \text{Tabu}_{\text{Buff}}} z_s \ \leq\ |P| + B_{max} - 1 \quad \forall\ (\text{Tabu}_{\text{Seq}},\text{Tabu}_{\text{Buff}})\ \in\ \text{TabuList}_{\text{D2}} \quad (73)$$

Similarly, every solution found by Model D1 is added to a Balancing Tabu List ($\text{TabuList}_{\text{D1}}$). However, Model D2 only adds optimal solutions to its Tabu List, and every feasible solution found by Model D1 is added to $\text{TabuList}_{\text{D1}}$. This is done by using a callback routine every time Model D1 reaches a new integer solution. The callback procedure is illustrated by Figure 16.

Figure 16 – Callback procedure flowchart



Source: Lopes *et al.* (2020a)

This callback tests the new balancing solution by applying it to Model D1 without the Tabu constraints (Inequality 73). This allows optimal sequencing and buffer allocation solutions to be obtained for that specific balancing solution. The value of cycle time is compared to the

incumbent solution and replaces the previous value if the newly found solution is better than the incumbent.

After the first time the Model D1 is executed, Constraints 74, 75, and 76 are added to Model D1. The first demands at least one difference from each previously explored Tabu balancing solution Tabu_{Bal} listed on Model D1's Tabu list $\text{TabuList}_{\text{D1}}$. The second demands the sum of processing times in each station to be smaller than the incumbent cycle time. The third one is based on the earliest and latest stations concept (SCHOLL; BECKER, 2006), defined for each task for the incumbent cycle time: $ES_{\left(t,\overline{CT}-1\right)}$ and $LS_{\left(t,\overline{CT}-1\right)}$. Such earliest and latest stations are computed by reducing the mixed-model problem into a single model one (BECKER; SCHOLL, 2006): First, the total processing times $\overline{D}_t$ for each task t are computed as the sum of processing times across models weighted by their demands N_m. Then, the total processing times before and after each task are calculated in order to define the minimum number of stations before and after each task (graph analysis based on precedence relations) given a value of cycle time (SCHOLL; BECKER, 2006). This allows some binary variables to be set to zero, as stated by Equation 76, because they necessarily lead to violations of Inequality 75, otherwise.

$$\sum_{(t,s)\ \in\ \text{Tabu}_{\text{Bal}}} x_{t,s}\ \leq\ |T|-1 \qquad \forall\ \text{Tabu}_{\text{Bal}}\ \in\ \text{TabuList}_{\text{D1}} \tag{74}$$

$$\sum_{p\ \in\ P} Tx_{p,s} \leq \overline{CT}-1 \qquad \forall\ s\ \in\ S \tag{75}$$

$$x_{t,s}=0 \qquad \forall\ t\ \in\ T, s\ \in\ S:\ s\ \notin\ \left[ES_{\left(t,\overline{CT}-1\right)}, LS_{\left(t,\overline{CT}-1\right)}\right] \tag{76}$$

By ignoring inefficiencies associated to blockages and starvations, the sum of processing times (Inequality 75) in each station is a lower bound on the cycle time: if that station operates at 100% efficiency, the MPS will take at least that much to be processed on it. This means that Inequality 75 and Equation 76 (whose violations imply in violations of Inequality 75) function as optimality cuts that act indirectly as Combinatorial Benders' Cuts (CODATO; FISCHETTI, 2006).

By demanding new solutions to respect these lower bound constraints, a large necessarily sub-optimal part of the search field is being discarded. Furthermore, if it is impossible for D1 to satisfy both the Tabu constraints and these performance constraints, then the incumbent solution must be optimal: each balancing Tabu solution was tested for optimal sequencing and buffer allocations, therefore, violating a Tabu constraint means testing a previously fully tested

balancing solution; violating the lower bound constraint means the trial solution will have a cycle time value greater or equal to $\overline{CT}$, meaning the solution is either dominated or equivalent to the incumbent.

This means that the proposed iterative decomposition can prove the optimality of its answer, as long as enough computational time is offered. It is likely that such optimality proof would take longer than the monolithic model's one. However, the fact that the decomposition can prove a solution's optimality is relevant as it means the full search field is guaranteed to be eventually tested. That said, the focus of the iterative decomposition is finding good solutions within the time limit, not proving optimality of the best solution found.

If Model D2 is infeasible for a given $\text{TabuList}_{\text{D2}}$, then all sequences and buffer allocation possibilities have been tested. If in all iterations Model D1 was solved to optimality, then the incumbent is also optimal. However, Model D1 can require substantial time to be solved. Hence, during tests, Model D1's time limit was set to a relatively short time (60 seconds, as indicated in Section 5.6) in order to allow more iterations, and therefore, more product sequences to be considered. With the tested global time limit, in not a single instance Model D2 was infeasible. If that occurs for larger global time limits, then $\text{TabuList}_{\text{D2}}$ can be simply reset.

5.5.1 Illustrative Example

The proposed method is hereby illustrated with a small example. Table 17 presents both the instance data and the proposed decomposition iterations: The problem contains nine tasks ($|T| = 9$), five product models ($|M| = 5$), three workstations ($S = \{1, 1.5, 2, 2.5, 3\}$, $S_b = \{1.5, 2.5\}$, $|S \setminus S_b| = 3$) and one available buffer ($B_{max} = 1$). Demand rates are considered equal for all models, leading to an MPS of with one product piece of each. Task durations for each model ($D_{t,m}$) are presented by Table 17, as well as precedence relations ('Pre.' column). The example's precedence diagram is presented in Figure 13.

The initial balancing solution, obtained by assigning each task t to station $\lfloor 1 + (t-1) \cdot |S/S_b|/|T| \rfloor$, is presented as 'Initial Bal.'. In Table 17, the values presented for each incumbent are the workstations to which each task is assigned (balancing, or 'Bal.' line), the product model at each position of the sequence (sequencing, or 'Seq.' line), and the position at which the buffer is assigned (buffer allocation, or 'BA.' line). The cyclical schedules of each of these solutions are presented by Figure 14(c). Thus, tasks 1-3 are initially assigned to station 1, tasks 4-6 to station 2, and tasks 7-9 to station 3. The first model D2 iteration uses the initial line balancing, leading

Table 17 – Illustrative example: data and iterations

Instance Data

	Models					
	1	2	3	4	5	
MPS	1	1	1	1	1	
Task	Durations					Pre.
1	1	0	0	0	0	-
2	1	2	1	0	0	1
3	1	2	1	0	0	1
4	0	0	1	2	2	2
5	1	0	1	1	1	3,4
6	2	0	0	0	3	3,4
7	0	1	0	1	2	5
8	1	1	0	0	0	7
9	0	0	2	1	1	8

Number of workstations: 3
Available buffers: 1

Method Iterations

		Assignments/Values								
Initial Bal.		1	1	1	2	2	2	3	3	3
Model D2 Iter. 1	Bal.	1	1	1	2	2	2	3	3	3
	Seq.	1	3	5	2	4				
	BA.	2.5								
	$\overline{CT}$	14			(First Incumbent)					
Model D1 Iter. 1	Bal.	1	1	2	1	2	3	2	3	3
	Seq.	1	3	5	2	4				
	BA.	2.5								
	$\overline{CT}$	12			(Second Incumbent)					

Model D2 Iter. - 2 does not improve solution
Model D1 Iter. 2 - infeasible
Therefore: current incumbent is optimal

to an incumbent with $\overline{CT} = 14$. The first model D1 iteration uses the incumbents' sequencing and buffer allocation, leading to a new incumbent with $\overline{CT} = 12$.

After the second incumbent is found, a second Model D2 iteration fails to improve the solution. The subsequent Model D1 iteration states that the model is infeasible, proving the optimality of the second incumbent. Table 18 helps to understand why the second Model D1 iteration was infeasible. It states the total task durations for each task ($\overline{D}_t = \sum_{m \in M} N_m \cdot D_{t,m}$) as well as the values of earliest and latest stations (SCHOLL; BECKER, 2006) for each task after each incumbent is found ($ES_{t,c}$, $LS_{t,c}$). These values are computed using the cycle time value c equal to that of each incumbent minus one ($\overline{CT} - 1$).

Table 18 – Illustrative example of earliest and latest stations after each incumbent

	Earliest and Latest Station for Each Task									
	Task:	1	2	3	4	5	6	7	8	9
$c = \overline{CT} - 1$	$\overline{D}_t$	1	4	4	5	4	5	4	2	4
13	$ES_{t,c}$	1	1	1	1	2	2	2	2	2
	$LS_{t,c}$	1	1	2	2	2	3	3	3	3
11	$ES_{t,c}$	1	1	1	1	2	2	2	3	3
	$LS_{t,c}$	1	1	1	1	2	3	3	3	3

Table 18 reports that after the second incumbent is found, the first four tasks must be assigned to the first station ($ES_{t,c}$=$LS_{t,c}$=1). However, the sum of their processing times (Table 17) is 14, which is higher than the current incumbent cycle time ($\overline{CT} = 12$). Hence, Constraint 75 will be necessarily violated, leading to the aforementioned infeasible status for Model D1's second iteration.

5.6 RESULTS

In order to verify how the proposed iterative decomposition compares to the proposed monolithic model, tests were performed on a 700-instance dataset. The dataset is based on Otto *et al.* (2013)'s SALBP instances. Chapter 2 combined single model instances of the same size to generate a set of mixed-model instances. This data set was extended by incorporating larger instances from Otto *et al.* (2013). In total, there are 350 task properties data vectors (processing times, and precedence relations) half of which have 20 tasks and the other half 50 tasks. These data vectors can be further classified in regard to Order Strength (OS), a number between 0 and 1 that reflects how restricted task assignment is by the precedence relations (SCHOLL, 1999). Each instance was tested with a fixed number of stations (10 for smaller cases, and 15 for the larger ones) and two numbers of buffers to be allocated (2 and 4 for the smaller cases, and 4 and 6 for the larger ones). All instances have five product models and the minimal part set was considered as one unit of each product model. Instance data is made available by the Lopes *et al.* (2020a)'s Supplementary Material.

Each of the 700 instances was solved with both the monolithic model (Mono) and the proposed iterative decomposition (Dec) with a time limit of 1800 seconds. All executions employed the same hardware and software conditions: Gurobi 7.5 was used to solve the models using a Core i7-3770 CPU (3.4GHz) and 16 GB RAM. The best solutions found by both methods are made available by Lopes *et al.* (2020a)'s Supplementary Material. Table 19 summarizes the upper bound results of the executions by comparing answers of Mono and Dec. In Table 19, N indicates the number of instances with the respective parameters (size, number of buffers and OS). Section 5.6.1 presents information on lower bounds and integer gaps. The solutions found by Dec and Mono were compared in regard to the goal function (cycle time) for each instance: When only one of the methods found a best solution for a specific instance, that solution counts as an exclusive best solution. The column *Nbest* reports the number of best solutions found by each method, and the number of exclusive best ones is reported in parenthesis. Similarly, the number of optimal answers is reported under the *Nopt* column. Dec often found the optimal answer (same cycle time as the optimal one by Mono), but could not prove its optimality. The number of instances in which each method was able to prove on its own the optimality of the answer it found is reported in parenthesis under the *Nopt* column.

The combined balancing, cyclical scheduling, and buffer allocation problem is complex

Table 19 – Results Summary - comparisons between *Dec*omposition and *Mono*lithic model

Dataset				Nbest(exclusive)		Nopt(proven)		CT_{Mono} - CT_{Dec} (T.U.)		
Size	Buffers	OS	N	Dec	Mono	Dec	Mono	Min.	Avg.	Max.
Small |T|=20 |S|=10	2	0.2	75	68(55)	20(7)	10(5)	13(13)	-55	26.8	97
		0.6	75	65(34)	41(10)	20(0)	23(23)	-27	15.2	92
		0.9	25	25(0)	25(0)	25(2)	25(25)	0	0	0
	4	0.2	75	62(43)	32(13)	13(4)	15(15)	-42	14.0	67
		0.6	75	69(28)	47(6)	39(16)	41(41)	-11	13.9	93
		0.9	25	25(0)	25(0)	25(17)	25(25)	0	0	0
Large |T|=50 |S|=15	4	0.2	75	75(75)	0(0)	0(0)	0(0)	31	116.0	263
		0.6	75	75(75)	0(0)	0(0)	0(0)	9	106.4	256
		0.9	25	24(23)	2(1)	0(0)	0(0)	-35	55.0	150
	6	0.2	75	74(74)	1(1)	0(0)	0(0)	-1	109.8	233
		0.6	75	71(70)	5(4)	0(0)	0(0)	-36	91.2	214
		0.9	25	22(19)	6(3)	0(0)	0(0)	-29	23.3	128

and hard to define lower bounds for. Hence, it is expected that only small very restrictive instances (high ordering strength) will be solved to optimality. Indeed, all small-size instances with high OS (0.9) were solved to optimality, but only a minority of instances with low OS (0.2), and none of the large instances. By comparing cycle time upper bounds, one can verify that the decomposition produced better answers in more instances than the monolithic model: In all problem sets, the decomposition found more or at least as many best answers as the monolithic model. In fact, the only sets in which they tied were the small ones with high OS. In all other sets (low and medium OS), the decomposition produced more best answers, both total and exclusive ones, than the monolithic model. In 70.9% (496/700) of instances, the decomposition outperformed the monolithic model, and the reverse only happened in 6.5% (45/700) of instances.

Furthermore, Table 19 presents the differences in cycle time of the best solutions obtained by the decomposition and the model: maximum, average, and minimum. Positive differences indicate that the decomposition obtained a better answer. It is clear that in larger and less restrictive instances, the decomposition produced more substantial differences. Notice that in some cases the monolithic model did outperform the decomposition by a significant margin, but those were the exception: both the average differences and the number of best solutions found indicate the superiority of the decomposition.

A drawback of the decomposition is the systematic lack of lower bounds and the consequent difficulty in proving optimality of its solutions. For instance, in the smaller dataset, while the number of optimal solutions found was similar, only in a minority of cases did the decomposition proved on its own the optimality of its solutions and most of these cases had high OS. Section 5.6.1 further explores these lower bound questions.

Table 19 also presents an interesting behavior regarding buffers: in the small dataset,

more instances with a higher number of buffers were solved to optimality by both the monolithic model and the decomposition. This is likely tied to the fact that more buffers enable lower cycle times, due to the decrease of blockages and starvations they allow. With a lower incumbent cycle time, performance constraints (Inequalities 66 and 75) can more easily discard parts of the search space. This suggests that with larger numbers of buffers, the highest sum of processing times becomes a better approximation of the realized cycle time. In other words: the naive lower bound on cycle time approaches the optimal value of cycle time when more buffers are added.

It has been repeatedly stated that combining the degrees of liberty provides better answers (BOYSEN *et al.*, 2008). However, doing so explicitly and in a single mathematical model might lead to computational difficulties and intractability, in particular for larger instances. In that regard, the proposed iterative decomposition combines these degrees of freedom by iteratively transforming part of the decision variables in parameters. This results in more easily tractable mathematical models, which have led to better performance: when instances are large and have a less restricted search-space, the decomposition generated better answers than the monolithic model. For small and very restricted instances, the decomposition still found many optimal solutions, even though it had significant difficulties to prove their optimality.

5.6.1 Sequential Procedure, Lower Bounds, and Integer Gaps

In order to evaluate how well the proposed decomposition and monolithic model combine these degrees of liberty, a sequential method was also implemented and executed with the same hardware and software conditions. This method (hereafter abbreviated as 'Seq') consisted of a two-stage process: First, a virtual single model assembly line balancing problem is solved. This model, seeks to minimize the $LBCT$ variable, bounded by the Inequality 77. The only other constraints are the occurrence (Inequality 61) and precedence relations (Inequality 62). The best answer obtained within the time limit (900 seconds) is then solved by the Model D2 to provide the optimal sequencing and buffer allocation solutions. This sequential approach mimics the procedure described by Sawik (2004) and employs the same virtual model definition used by Battini *et al.* (2009).

$$LBCT \geq \sum_{t \in T} \sum_{m \in M} N_m \cdot D_{t,m} \cdot x_{t,s} \qquad \forall\, s \in S \tag{77}$$

It is easy to show that a lower bound to $LBCT$ is a lower bound to the cycle time (CT)

of the original problem: if each station operates at 100% efficiency, each MPS will take at least the total processing time assigned to the most loaded station (Inequality 77) to flow through the line in the steady-state. Hence the MPS-wise cycle time is higher or equal to $LBCT$. Thus, if $LBCT$ is higher than the lower bound obtained by the monolithic model, its value can be used as a lower bound of the original problem. If the optimal $LBCT$ value is unknown, then the best bound obtained by the virtual single model is employed for the comparisons. This allows integer gaps to be calculated for the **Seq**uential, the **Dec**omposition and the **Mono**lithic best answers. The average integer gaps are presented by Table 20. Notice that the average Dec gap (5.3%) is 19% smaller than the average Mono gap (6.6%) and 68% smaller than the average Seq gap (16.9%). These gaps tend to be smaller for instances with more buffers: For instance, the average gap for large instances with six buffers is lower than that of the same instances with four for all values of ordering strength. This might be explained by the fact that more buffers allow better cyclical schedules and the assumption that lower bounds might not change much, since they are very difficult to compute even without taking the buffer information into account. Furthermore, instances with higher ordering strength tend to have smaller gaps. This is explained by the smaller search field, which makes finding better solutions and bounds easier.

Table 20 – Average integer gaps and lower bounds

Dataset				Integer Gaps			LB comparisons		Gap Best $-$ $LBCT$	
Size	Buffers	OS	N	Seq	Dec	Mono	Rate	$N_{Imp.}$	All	Optimal
Small |T|=20 |S|=10	2	0.2	75	20.1%	5.7%	6.6%	99.2%	30	7.6%	7.8%
		0.6	75	18.4%	3.9%	4.4%	97.0%	18	7.6%	6.8%
		0.9	25	15.0%	0.0%	0.0%	92.3%	0	7.7%	7.7%
	4	0.2	75	13.2%	3.0%	3.5%	99.4%	24	4.1%	2.4%
		0.6	75	11.7%	1.5%	2.0%	98.4%	23	3.8%	3.7%
		0.9	25	8.0%	0.0%	0.0%	96.7%	0	3.3%	3.3%
Large |T|=50 |S|=15	4	0.2	75	22.2%	8.8%	11.3%	100.2%	62	8.8%	-
		0.6	75	20.6%	8.7%	10.9%	100.6%	66	8.7%	-
		0.9	25	14.8%	6.8%	8.0%	101.0%	17	7.0%	-
	6	0.2	75	18.0%	7.1%	9.5%	100.1%	61	7.1%	-
		0.6	75	16.9%	7.1%	9.1%	100.5%	66	7.1%	-
		0.9	25	11.4%	4.5%	5.1%	100.7%	14	4.7%	-
Average/Total				16.9%	5.3%	6.6%	99.2%	381	6.7%	5.0%

Table 20 also presents lower bound comparisons between the monolithic model and $LBCT$. The 'Rate' column presents the rate between $LBCT$ and the lower bound obtained by the monolithic model: values lower than 100% indicate that the monolithic model provided better bounds in average. Notice that for small instances, in average, the monolithic model provided better lower bounds and that the reverse happened for the larger instances. Column '$N_{Imp.}$' indicates the number of instances in which $LBCT$ was higher than the bound provided by

the monolithic model. Notice that this was the case for the majority of Large instances, however the difference was relatively small. For a minority of small instances, the sequential approach improved lower bounds.

Lastly, Table 20 presents a gap between the best upper bound (usually found by the proposed decomposition) and the $LBCT$ lower bound. This rate is computed by dividing the difference between $LBCT$ and the $\overline{CT}$ for the best method by $LBCT$. The 'All' column presents the average difference across all instances and the 'Optimal' column presents the average across the instances solved to optimality. Notice that the values in these columns are similar for small instances, suggesting that gaps to the optimal solutions are smaller than reported by the Integrality Gap columns: Instances that were solved to optimality presented average gaps to $LBCT$ comparable in magnitude to those that were not. This suggests that the monolithic model and the proposed decomposition might have more difficulty in providing good lower bounds than in providing good solutions. This is expected, lower bounds for the BCSB-P tend to be difficult to establish due to both the combination of the three degrees of freedom and the difficulties in performance measure.

5.6.2 Influence of Optimality Cuts

In order to measure the influence of the Optimality Cuts (Inequality 75 and Equation 76) on the method's performance, the following tests were conducted: For a fifth of BCSB-P instances in each dataset (140 out of 700 total), 25 randomly generated sequences were supplied to model D1. For each product sequence, the model was executed twice, once with and once without the Optimality Cuts (7000 total Model D1 instances). Each Model D1 instance was solved with a time limit of 60 seconds, the same one employed by the decomposition step. The solutions obtained with and without the Optimality Cuts were compared both in terms of solution quality (cycle time) and solution time. The percentage of instances with different values (reductions and increases) of cycle time and solution time is reported by Table 21. Instances in which solution quality and time were better with the Cuts than without them are reported as reductions (Red.) and when the contrary happened they are reported as increases (Inc.). The average relative difference in those measures is also reported by Table 21. For multiple Model D1 instances, the difference in CT between solutions was only 1 time unit (less than 0.05%) of one another. In order to restrict the analysis to more substantial differences, these were considered as ties for the purpose of counting. Similarly, time comparisons are considered ties when optimality

is not reached neither with or without the optimality cuts.

Table 21 – Influence of Optimality Cuts: Cycle Time and Solution Time comparisons

Instances		CT Differences			Time Differences		
Size	OS	Nb. Red.	Nb. Inc.	Avg. Red.	Nb. Red.	Nb. Inc	Avg. Red.
Small	0.2	27.6%	11.3%	0.16%	87.5%	3.2%	44.4%
	0.6	32.8%	11.5%	0.26%	92.8%	2.2%	50.9%
	0.9	52.0%	18.8%	1.2%	98%	2%	47.7%
Large	0.2	63.7%	3.2%	0.66%	0%	0%	0%
	0.6	55.2%	4.4%	0.59%	0%	0%	0%
	0.9	72.4%	16.8%	0.44%	98.8%	0%	73.5%

For both instance sizes and all values of OS, the number of instances with CT reductions was more than double the number of instances with increases. A large number of these instances was solved to optimality, meaning that comparing the time required to do so can be relevant. In small instances (with low, medium, and high OS), as well as large ones with high OS, the majority of iterations were solved significantly faster when the optimality cuts are employed. For small instances, this time reduction was near 50%, while for large ones reductions only occurred for high OS, averaging 73.5%. This means that more iterations can be performed within the time limit, as more cyclical product sequences and buffer allocations are fully tested. Large instances with low and medium OS displayed too broad search fields so that none of their model D1 instances were solved to optimality. This means that time comparisons were not possible in these datasets, as all instances tied at the time limit. However, in the majority of executions on these instances, the optimality cuts led to better solutions (63.7% and 55.2%, as indicated by column Nb. Red.), averaging around 0.5% reductions in cycle time. Therefore, Table 21 shows that, for most instances, the optimality cuts lead to better average performance both in terms of the goal function (cycle time minimization) and in terms of time required to solve Model D1 instances.

5.7 CONCLUSIONS

Optimizing throughput of mixed-model asynchronous lines requires the solution of three optimization problems: balancing the task distribution, sequencing product models, and allocating buffers (internal storage). While many works have focused on each of these problems, and on some combinations of them, none thus far had explicitly combined all three degrees of freedom. This chapter expands a cyclical formulation capable of measuring steady-state performance of said lines when cyclical sequencing and buffer allocation are parameters, and allows these to be decision variables.

A new mixed-integer linear programming model is presented to describe and optimize the simultaneous **B**alancing, **C**yclical **S**equencing and **B**uffer allocation asynchronous assembly line **P**roblem (BCSB-P). These three degrees of freedom have not been previously combined in the literature. However, by comparing the proposed monolithic model to a sequential procedure (which mimics previous literature methods), tests on a 700-instance dataset show that combining the degrees of freedom is very important to achieve good answers: the average integer gap of the solutions obtained by the monolithic model is 60% smaller than those of the sequential approach (Table 20). However, the combined problem is computationally challenging.

Based on this computational difficulty, an iterative decomposition procedure is also presented to obtain better answers than the proposed monolithic model (Figures 15 and 16). Two mathematical models operate alternatively: a balancing model in which product sequence and buffer layouts are parameters, and a sequencing and buffer allocation model in which processing times (balancing) are parameters. These models are based on the monolithic model (Defined by Expressions 60 to 72): they retain part of the original decision variables and change the others to parameters. Optimality cuts are also incorporated by the decomposition (Expressions 75 and 76) and are shown to lead to better solution quality and time at the method's iterations (Table 21). The proposed decomposition outperforms the monolithic model in the production of upper bounds for the problem. The average integer gap of solutions obtained by the proposed decomposition is 19% smaller than those of the monolithic model (Table 19). This difference is more significant for large instances and tends to be stronger for less restricted cases. Although the decomposition's focus is to generate good solutions rather than optimality proof, it does have the capacity to do so. A key challenge for that to occur more often, however, lies in lower bounds: in most large instances, the bounds obtained by the monolithic model were weaker than the lower-bound provided by the virtual single model balancing problem.

6 BALANCING UNDER STOCHASTIC SEQUENCING

This chapter is based on an article published in Computers and Operations Research (LOPES *et al.*, 2020b). In it, a simheuristic procedure is proposed to balance asynchronous assembly lines when sequencing is beyond the line manager's control and can be treated as stochastic. While this random element prevents cyclical scheduling from finding a direct application, the article is also a follow-up to a conference paper presented at the L SBPO in Rio de Janeiro (LOPES *et al.*, 2018). In the conference paper, the stochastic steady-state is approximated by a weighted sum of cyclical scheduling steady-states of finite size. In Lopes *et al.* (2020b), however, a new algorithmic line simulator is used for that purpose, and the article focuses on the solution method. Another conference paper by the author (LOPES *et al.*, 2019), presented at the XIX CLAIO in Lima (Peru), addresses another dimension of this problem. It described a Markov-chain formulation to obtain exact performance measurements for given balancing solutions, but that is beyond the scope of this chapter.

6.1 CONTEXT

Assembly Line Balancing (ALB) problems consist of distributing operations (tasks) to workers or robots (stations) in a manner that maximizes a given performance measure under a set of practical constraints. Assembly line balancing is key to efficient design and operation of production systems (SCHOLL, 1999). The simplest ALB problem which is widely studied in the literature is the Simple Assembly Line Balancing Problem (SALBP) whose assembly line is assumed to be composed of simple serial stations producing a single product model. Furthermore, it is assumed that operations have given deterministic processing times and precedence relations, and that the sum of processing times in each station is bounded by the cycle time (either variable or parameter) which determines the line pace (BAYBARS, 1986). While several variants of these problems have been studied, this chapter is focused on the Mixed-Model Assembly Lines Balancing (MMALB), i.e. lines that are shared by a set of similar product models.

In mixed-model assembly lines, it is assumed that products can be mixed without setup times. However, differences in the processing times of tasks for different models can make the effective cycle time (or throughput) difficult to compute. The effective cycle time is not only a function of line balancing, but also of assembly line's physical characteristics and of

how products are sequenced. This chapter assumes that the assembly line is reliable and has an asynchronous line control with a given number of stations and buffers between them. An infinite input buffer before the first station and an infinite output buffer after the last one are also assumed. In asynchronous lines, a workpiece is allowed to independently and discretely move to the following station when processing is completed at the current station and the next station is empty. Mixed-model sequencing is often treated as a separate optimization problem (BOYSEN *et al.*, 2009c). However, some authors also focus on simultaneous balancing-sequencing problems (SAWIK, 2012) or consider product sequences as parameters (BIELE; MÖNCH, 2018). In this chapter, a make-to-order context is considered as usually defined in the assembly line balancing literature (BUKCHIN *et al.*, 2002; VENKATESH; DABADE, 2008; TIACCI, 2015b). The model of the next workpiece to enter the line is a random variable and the probability of being an instance of a specific model matches the model's demand rate. This means that even if processing times are considered deterministic for each task and product model, there is a stochastic sequence of product models to be produced that makes the throughput of the assembly line hard to predict.

A series of recent publications (TIACCI, 2012; TIACCI, 2015b; TIACCI, 2015a; TIACCI, 2017; TIACCI; MIMMI, 2018) have addressed variants of this problem with simulation-based evaluation functions. However, the throughput maximization variant of this problem (given the number of workstations and buffers) has not been addressed by these works. To the best of the authors' knowledge, few works have addressed this issue. Bukchin *et al.* (2002) and Venkatesh and Dabade (2008) describe goal functions and discuss surrogates such for horizontal balancing (BECKER; SCHOLL, 2006). However, Tiacci (2015b) points out that these are not objectives in themselves, but rather supposed means to achieve a high and stable throughput. A different body of literature (MCNAMARA *et al.*, 2016) discusses the "bowl phenomenon" in unpaced lines. It states that adequately unbalanced lines might outperform balanced ones as a result which has been shown to be robust (HILLIER; SO, 1996). Even though the bowl phenomenon literature presupposes stochastic components other than ones of the make-to-order context studied here, some of the obtained results present similar findings of the literature.

This chapter addresses the throughput maximization (or cycle time minimization) of possibly buffered mixed-model assembly lines with stochastic product sequences. The main contribution is to propose a new simheuristic (JUAN *et al.*, 2015) procedure (PSH) as optimization method that utilizes a new cycle time simulator (CTS) to evaluate solution quality. CTS is a specialized simulator that runs faster than other benchmark simulators. It allows PSH to find

good solutions within practical runtime. Results are compared to benchmarks of the literature, and to what is expected from the bowl phenomenon.

This chapter is structured as follows. Section 6.2 presents an overview of related works. Section 6.3 presents a formal definition of the problem and notation. Section 6.4 presents the new cycle time simulator CTS designed to simulate effective cycle time of mixed-model assembly lines. Section 6.5 presents the simheuristic method PSH developed to employ CTS and maximize the throughput of mixed-model assembly lines. Section 6.6 presents the undertaken computational experiments, and Section 6.7 summarizes the chapter' findings and contributions.

6.2 RELATED WORKS

Readers can refer to review papers (BECKER; SCHOLL, 2006; BOYSEN *et al.*, 2007; BOYSEN *et al.*, 2008; BATTAÏA; DOLGUI, 2013) for more details on basic elements of assembly line balancing. Some specific topics of particular interest for this chapter are: reductions from mixed-model to single model (BECKER; SCHOLL, 2006) and common approaches tied to mixed-model lines; classifications and discussions of line features (BOYSEN *et al.*, 2007; BOYSEN *et al.*, 2008) such as line control and buffers; an overview of problems, common goal functions and solution approaches (BATTAÏA; DOLGUI, 2013).

Mixed-model assembly lines often employ a conveyor with given constant speed and an evenly spaced product launch discipline. In that case, throughput is stable and the optimization problem consists of providing balancing and sequencing solutions that minimize the required number of stations and work overload (BOYSEN *et al.*, 2009c). However, for some types of line, the conveyor does not move with constant speed; such is the case in unpaced synchronous and asynchronous lines. For these lines, or when the optimization problem targets throughput maximization, determining throughput (or its inverse, the effective cycle time) is challenging. While product sequencing and line characteristic factors affect such throughput calculation, there are four main approaches commonly used in the literature: worst case, best case, indirect evaluation, and direct evaluation.

The most widely used analysis is the *worst case* one, in which processing times are controlled such that a given cycle time is valid for all product models regardless of product sequence and line control: Karabati and Sayin (2003) employ that concept for synchronous lines, Akpinar and Baykasoglu (2014a) and Akpinar and Baykasoglu (2014b) apply it for a variant with set-ups between tasks, Roshani and Nezami (2017) employ it for multi-manned mixed-model

lines, Dong *et al.* (2018) present a chance-constrained version of this rationale for problems with stochastic task times, Kucukkoc and Zhang (2015) and Kucukkoc and Zhang (2016) use it for problems with parallel assembly lines.

Some authors consider the *best case* scenario (usually associated with large buffer availability), in which effective cycle time is approximated by the highest station-wise weighted average of processing times: Sawik (2002) uses it as the first stage of a decomposition procedure in a balancing-sequencing problem, Alghazi and Kurz (2018) employ it in a problem variant incorporating parallel workstations and ergonomic constraints, Merengo *et al.* (1999) present a vertical balancing formulation that incorporates that core idea. Another very common approach is the use of *indirect goal functions*, in particular workload smoothing (DECKER, 1993; ÖZCAN *et al.*, 2010; HAMZADAYI; YILDIZ, 2012; HAMZADAYI; YILDIZ, 2013) and deviations from an ideal average (KIM *et al.*, 2000; KIM *et al.*, 2006; BATTINI *et al.*, 2009). The most accurate, and most problem-specific, is to explicitly take sequencing and line features into account to provide a *direct measure* of throughput: Sawik (2012), Öztürk *et al.* (2013) and Öztürk *et al.* (2015) combine balancing and sequencing to minimize the makespan of a product set, Biele and Mönch (2018) present a cost-oriented optimization problem in which specific product sequences are given, Chapters 2-5 have aimed at stable cycle time minimization on the context of given cyclical product sequences and integrated balancing with cyclical sequencing.

In the specific case of asynchronous lines with a make-to-order context (BUKCHIN *et al.*, 2002), in which product sequences are stochastic, typical indirect performance measures have been mostly replaced by simulation-based direct evaluation procedures: A recent series of works have applied these methods to lines with parallel stations (TIACCI, 2015a), buffer allocation (TIACCI, 2015b), U-shaped lines (TIACCI, 2017), and ergonomic constraints (TIACCI; MIMMI, 2018). These works, however, employ a cost-oriented framework with a target throughput, and do not aim directly at an effective cycle time minimization (throughput maximization). This chapter, therefore, presents an optimization method to bridge this specific gap: minimizing the effective cycle time of (possibly buffered) asynchronous mixed-model lines on a make-to-order context and analyze the impact of buffer allocation on efficient balancing solutions.

In this chapter, simheuristics are chosen as the method to achieve such simulation-based optimization (JUAN *et al.*, 2015). They are a rather recent research line with a diverse range of applications (GONZALEZ-NEIRA *et al.*, 2017; HATAMI *et al.*, 2018; GUIMARANS *et al.*, 2018) that go beyond using a simulation as a black-box evaluation function of a meta-heuristic.

Simheuristics use problem-specific information to closely integrate optimization and simulation, using information on feasibility and solution quality prior to simulation runs (JUAN *et al.*, 2015).

6.3 PROBLEM STATEMENT

A high-level definition of the studied problem is presented by Expressions (78) to (81). Consider an assembly line consists on series of stations and buffers, represented by the set of integers $S = 1,2,...,|S|$. Workstations are equally manned and equipped so that any task can be assigned to any station. Buffers are represented by the set S_B, a sub-set of S, and no task can be assigned to them. The line produces a set M of different product models, each model m of which has a given demand rate O_m. Product models share a set of tasks T, and each task t has a deterministic duration $D_{t,m}$ for each product model m. For specialization purposes, tasks must be assigned to the same station for all product models (Constraint (79)). Furthermore, there is a set of precedence relations R, as stated by Constraint (80). Lastly, the processing time $Tx_{m,s}$ at each station s for each product model m is the sum of processing times of the tasks assigned to the station, as stated by Constraint (81).

$$\text{Minimize} \quad E\left[\, CT(Tx_{m,s}, O_m) \,\right] \tag{78}$$

$$\sum_{s \,\in\, S \setminus S_B} x_{t,s} \;=\; 1 \qquad\qquad \forall\, t \;\in\; T \tag{79}$$

$$\sum_{s \,\in\, S} s \cdot x_{t_1,s} \;\leq\; \sum_{s \,\in\, S} s \cdot x_{t_2,s} \qquad\qquad \forall\, (t_1,t_2) \;\in\; R \tag{80}$$

$$Tx_{m,s} = \sum_{t \,\in\, T} D_{t,m} \cdot x_{t,s} \qquad\qquad \forall\, m \;\in\; M,\, s \;\in\; S \tag{81}$$

Product flow is asynchronous: each workpiece can move forward discretely when processing at the current station is completed and the next station (or buffer) is empty. Starvations occur when a station is empty and the previous one has not completed processing of the next piece while blockages occur when a piece has completed processing and the following station is occupied by a product. By hypothesis, an infinite input buffer exists before the first station and an infinite output buffer exists after the last one, meaning the first station is never starved and the last one is never blocked. Product sequence is stochastic, i.e. the model of the next product to enter the line is a random variable dictated with probabilities equal to the demand rates O_m of each model m.

The optimization problem consists of defining a task-station assignment that respects precedence relations (feasible) and that maximizes the expected throughput (i.e. minimizes the effective cycle time). While precedence relations can be quickly checked, throughput must be simulated in order to be measured.

6.4 CYCLE TIME SIMULATOR (CTS)

Consider the first workpiece p to enter a straight asynchronous assembly line without parallel stations: its schedule will not depend on subsequent workpieces. Furthermore, departure times from each station are the only information needed to describe blockages or starvations tied to the second piece. For instance, if the first piece leaves a station after the second piece is ready to depart the previous one, the second piece will be blocked; on the other hand, if the first piece leaves a station before the second one can enter it, that station will be starved. This behavior can be generalized by knowing the departure times of workpiece p from each station. These departure times are enough to describe the departure times (scheduling) of workpiece $(p+1)$.

Algorithm 1 synthesizes the proposed cycle time simulator CTS. Given a balancing solution sol, $|K|$ replications of simulations with $|P|$ workpieces are conducted. $Tx_{M,S}$ stores the processing times of each model $m \in M$ at each station $s \in S$. The NextModel() subroutine sets the next product model to enter the assembly line either randomly (based on product demand rates, as presented by Algorithm 2) or according to a given fixed sequence. The time each station s is available to receive the next workpiece is stored in $Tfree_s$. Initially, these values are set to zero. Each simulated workpiece i updates these values one station at a time: at each station s, these values are updated by considering possible starvations by station $(s-1)$ and blockages by station $(s+1)$. Naturally, the update rules are slightly different for the first and last stations (lines 6 and 10 in Algorithm 1, respectively). The first station is never starved and the last station is never blocked.

The NextModel() function is presented by Algorithm 2. In line 1, it generates random number generator, with uniform distribution between 0 and 1, which is used to determine the next product to enter the line given the demand rates O_m. It is assumed that $\sum_m O_m = 1$, meaning the demand rates add up to 100%. This function is employed by the proposed Cycle Time Simulator (CTS) in the case of stochastic product sequences.

CTS is designed to simulate assembly lines with stochastic product sequence and deterministic processing times. However, it can be easily adapted to incorporate deterministic

Algorithm 1 – CycleTimeSimulator(sol, $|K|$, $|P|$)

1: $Tx_{M,S} \leftarrow$ ProcessingTimesFromSolution(sol)
2: $sumCT = 0$
3: **for** $k = 1, k \leq |K|; k++$ **do** ▷ Outer loop: $|K|$ simulation replications
4: $\quad Tfree_{1...nS} \leftarrow \mathbf{0}$
5: $\quad$ **for** $p = 0; p \leq |P|; p++$ **do** ▷ Inner loop: Simulation size of $|P|$ products
6: $\quad\quad m \leftarrow$ NextModel()
7: $\quad\quad Tfree_1 \leftarrow \max(Tfree_1 + Tx_{m,1}, Tfree_2)$ ▷ First station is never starved
8: $\quad\quad$ **for** $s = 2; s < |S|; s++$ **do**
9: $\quad\quad\quad Tfree_s \leftarrow \max(Tfree_{s-1} + Tx_{m,s}, Tfree_{s+1})$ ▷ Standard rule for central stations
10: $\quad\quad$ **end for**
11: $\quad\quad Tfree_{|S|} \leftarrow Tfree_{|S|-1} + Tx_{m,|S|}$ ▷ Last station is never blocked
12: $\quad\quad$ **if** $p = 0$ **then**
13: $\quad\quad\quad First \leftarrow Tfree_{|S|}$
14: $\quad\quad$ **end if**
15: $\quad$ **end for**
16: $\quad sumCT \leftarrow sumCT + (Tfree_{|S|} - First)/|P|$
17: **end for**
18: **return** $sumCT/|K|$

Algorithm 2 – NextModel()

1: $ran \leftarrow$ Random() ▷ Random number: $0 \leq ran \leq 1$
2: $sum \leftarrow 0$
3: **for** $m = 1; m < |M|; m++$ **do** ▷ $|M|$: number of product models
4: $\quad sum \leftarrow sum + O_m$
5: $\quad$ **if** $sum \geq ran$ **then**
6: $\quad\quad$ **return** m
7: $\quad$ **end if**
8: **end for**
9: **return** $|M|$

sequences and stochastic processing times. For instance, the processing time $Tx_{m,s}$ at lines 6, 8, and 10 of Algorithm 1 is set as a parameter, but it can be easily replaced by a random variable. Furthermore, a warm-up period can be implemented at line 11 by considering the first (valid) workpiece as being different from zero, i.e., a warm-up period is measured in number of pieces rather than time units.

An illustrative example of the proposed cycle time simulator (CTS) for an assembly line with 2 product models and 5 stations is presented by Table 22. Demand rates are set to 50% of each product model with production of 10 workpieces ($|P| = 10$), and the number of simulation replications is set to one ($|K| = 1$). Processing times for each model m at each station s are presented on the left side of Table 22. Columns of the right side of Table 22 presents values of $Tfree_s$ for each station when each product leaves the station. In this short simulation, the average cycle time (Algorithm 1, lines 15 and 17) is $(86 - 20)/10 = 6.6$ time units per product. Notice that product workpiece 0 functions as a warm-up for the simulation, as described in Algorithm 1 (Lines 12 to 14).

Table 22 – Illustrative example of CTS execution

Processing ($Tx_{M,S}$)

	Model m	
Station s	1	2
1	7	3
2	2	4
3	2	5
4	4	7
5	5	5

$Tfree_s$ for each station s after each product p

Product p	0	1	2	3	4	5	6	7	8	9	10
Model m	1	2	1	1	1	1	2	2	1	1	2
$s = 1$	7	10	17	24	31	38	41	45	50	57	64
$s = 2$	9	14	19	26	33	40	45	50	57	64	69
$s = 3$	11	19	26	31	36	42	50	57	64	69	74
$s = 4$	15	26	31	36	41	46	57	64	69	74	81
$s = 5$	20	31	36	41	46	51	62	69	74	79	86

6.4.1 Comparison to Literature Benchmark

The Cycle Time Simulator (CTS) presented by Algorithm 1 is hereby compared to the most recent benchmark Assembly Line Simulator (ALS) presented by Tiacci (2012). CTS is more specialized than ALS, which can incorporate other features such as parallel workstations and U-shaped lines. However, this section restricts the comparison to the specific capabilities of CTS: namely straight lines without parallel workstations. There are two variable dimensions that both CTS and ALS can process: stochastic vs. deterministic processing times, and stochastic vs. deterministic product sequences. ALS is made available for download by its author at the DOI:10.13140/RG.2.2.16680.72961. The data used in these simulation comparisons originates from Chapter 2's practical case study, and is made available in Lopes *et al.* (2020b)'s supplementary material. A very long simulation length ($2.5 \cdot 10^6$ workpieces/loads) was used to reduce the impact of the transient behavior and provide the total simulation time in seconds. Table 23 presents the results.

Table 23 – Average cycle time (CT) in time units and simulation runtime in seconds for ALS and CTS

		ALS		CTS	
Processing	Sequencing	CT	Runtime	CT	Runtime
Deterministic	Deterministic	1605.7	2.66	1605.7	0.26
Deterministic	Stochastic	1592.6	2.99	1592.4	0.29
Stochastic	Deterministic	1617.5	8.08	1617.3	0.80
Stochastic	Stochastic	1617.3	8.01	1618.5	0.85

According to Table 23, the computed cycle times for both simulators are very similar. Such small differences (less than 0.1%) are easily attributable to the sampling process using random variable generators. However, runtimes are quite different. CTS is approximately ten times faster than ALS. This is expected as CTS is more specialized than ALS. However, the development of such fast specialized simulator is key to the simheuristic presented in Section 6.5 as simulation is extensively used.

Algorithm 3 – Proposed Simheuristic

```
1: while TimeLimitNotReached do
2:     best ← GenerateInitialSolution(|T|, |S|)
3:     ini ← null
4:     Zinc ← ∞
5:     Zbest ← ∞
6:     while ini ≠ best do                          ▷ Until convergence to local minimum
7:         ini ← best
8:         for all s1,s2 ∈ S \ S_B such that s1 is adjacent to s2 do
9:             sol ← ini
10:            UnassignTasks(sol, s1, s2, Ltasks)          ▷ Ltasks: list of unassigned tasks
11:            IterativeSearch(sol, best, Zbest, s1, s2, Ltasks, 1)   ▷ Local search subrouting
12:        end for
13:    end while
14:    LocalMinima.Add(best)                        ▷ LocalMinima: list of local minimal solutions
15:    Zsol ← CycleTimeSimulator(best, |K|_l, |P|_l)   ▷ |K|_l, |P|_l: external parameters
16:    if Zsol < Zinc then
17:        Zinc ← Zsol
18:        Incumbent ← best
19:    end if
20: end while
21: return (Incumbent, Zinc)
```

6.5 PROPOSED SIMHEURISTIC (PSH)

In order to exploit CTS presented in Section 6.4, the following simheuristic procedure is proposed. It is based on Iterated Local Search (ILS) (LOURENÇO *et al.*, 2003), which starts with randomized feasible solutions and performs local searches until a local minimum is found. Local minima are then stored in a solution list that prevents duplicate solutions, effectively acting as a Tabu list (GLOVER, 1986).

The proposed simheuristic assumes a given number of stations and buffers between them. Buffers are modeled as dummy stations with no task assigned to them. The neighborhood search is based on repeated two-station optimization: tasks are fixed on all but two stations, and all relevant ways to reassign tasks within that station pair are tested for all adjacent station pairs. Blockages and starvations happen between adjacent pairs. Therefore, it is expected that neighborhoods defined in terms of adjacent pairs of stations will contain solutions with different amounts of blockages and starvations. This means these neighborhoods will lead to incremental improvements as required by ILS algorithms (LOURENÇO *et al.*, 2003).

Algorithm 3 presents the high level operations of the proposed simheuristic. The initial solution at Line 2 is provided by a simple randomized priority rule: tasks are given random priorities and re-ordered accordingly. They are then assigned one at time such that tasks with high priority and whose predecessors were already assigned are selected first. The task i to be selected

is then assigned to the station $\lfloor 1+(i-1)\cdot|S|/|T|\rfloor^{\text{th}}$ in $S \setminus S_B$. For instance, if there are 20 tasks and five stations, the first four task selected will be assigned to the first *non-buffer* station, the next four tasks to the second *non-buffer* station, and so on. At Lines 8 to 12, Algorithm 3 selects pairs of adjacent *non-buffer* stations to remove all tasks and test all possible re-assignments of such tasks. This makes feasibility verification easier, as only precedence between tasks assigned to the station pair (s_1, s_2) must be checked. At Line 15, CTS is used to measure the performance of the most recent local minimum, which is saved in the $LocalMinima$ list (Line 14) and stored as incumbent if appropriate (Lines 16 to 19). Notice that the CTS takes into account buffers between stations, meaning that solution performance will be dependent on the buffer layout. The IterativeSearch() subroutine (Line 11) is further explained by Algorithm 4.

Algorithm 4 – IterativeSearch(sol**,** $best$**,** $Zbest$**,** s_1**,** s_2**,** $Ltasks$**,** i**)**

```
1:  t ← Ltasks(i)                                          ▷ Current task to be assigned
2:  if Assignable(sol, t, s1, Zbest) then                  ▷ Assign t at s1 branch
3:      Assign(t, s1, sol)
4:      if i = Ltasks.Count then                           ▷ Complete solution check
5:          if not LocalMinima.Contains(sol) then
6:              Zsol ← CycleTimeSimulator(sol, |K|s, |P|s)   ▷ |K|s, |P|s: simulation parameters
7:              if Zsol < Zbest then
8:                  best ← sol                             ▷ Store incumbent
9:              end if
10:         end if
11:     else IterativeSearch(sol, best, Zbest, s1, s2, Ltasks, i + 1)   ▷ Incomplete solution, assign next task
12:     end if
13:     Unassign(t, s1, sol)
14: end if
15: if Assignable(sol, t, s2, Zbest) then                  ▷ Assign t at s2 branch
16:     Assign(t, s2, sol)
17:     if i = Ltasks.Count then                           ▷ Complete solution check
18:         if not LocalMinima.Contains(sol) then
19:             Zsol ← CycleTimeSimulator(sol, |K|s, |P|s)   ▷ |K|s, |P|s: simulation parameters
20:             if Zsol < Zbest then
21:                 best ← sol                             ▷ Store incumbent
22:             end if
23:         end if
24:     else IterativeSearch(sol, best, Zbest, s1, s2, Ltasks, i + 1)   ▷ Incomplete solution, assign next task
25:     end if
26:     Unassign(t, s2, sol)
27: end if
28: return
```

The IterativeSearch() procedure is the local search aspect of the proposed simheuristic. Tasks unassigned from stations s_1 and s_2 are stored in list $LTasks$. Algorithm 4 begins with $i = 1$, taking the first task of the list and attempting to re-assign it to each station. Lines 2 to 14 present such attempt for s_1 and Lines 15 to 27 present the analogous attempt for s_2. The Assignable() test at Line 2 serves a dual feasibility-optimality role: on one hand it tests whether

the assignment of task t to s_1 violates a precedence relation in regard to a task assigned to s_2 (feasibility); on the other hand, it tests whether the sum of processing times of tasks already assigned to s_1 is too large if t is assigned to it (optimality condition). The optimality condition of this test is based on the observation that each workstation is a potential line bottleneck (BECKER; SCHOLL, 2006). Therefore, the expected assembly line performance (and, hence, cycle time) is bounded by the expected performance of each station, which is trivially determined by the weighted averages of processing times. Such comparison is dynamically strengthened during the iterative search as it is based on the performance of the current best neighbor ($Zbest$). This allows substantial reductions on the number of simulations on unpromising neighbors. By doing this, simulation and optimization are more closely integrated (JUAN *et al.*, 2015): answers reported by simulations are used to accelerate neighborhood searches by informing performance requirements, and problem-specific information is used to reduce the number of simulations runs on low-quality solutions.

A logical test at Line 4 of Algorithm 4 verifies whether all tasks were reassigned. If so, the current solution sol is tested as a member of $LocalMinima$ list (Line 5). If it is a new solution, sol is tested for performance, and saved as the best solution of the current iteration ($best$), if appropriate (Lines 6 to 9). Back to Line 4, if not all tasks were assigned, the search iterates by incrementing the depth i as stated at Line 11. Tasks are unassigned at Line 13 from s_1 so that they can be tested as possible assignments for s_2 (Lines 15 to 27).

In that sense, the neighborhood of each solution tested in Algorithm 3 is the set of all relevant (feasible, new, and promising) two-station reassignments: they must be feasible regarding precedence relations, new in regard to the Local Minima List, and promising in the sense that their performance lower bound must be better than the current solution ($best$). Until the time limit or a stopping criterion is reached, this procedure is iterated by randomized re-starts. The Local Minima restriction prevents the method from converging to previous solutions and, therefore, enhances solution diversity.

Notice that, while both Algorithm 3 and Algorithm 4 use the proposed CTS (Algorithm 1), the parameters change. During the local search, simulations are shorter and, therefore, less accurate (for experiments, $|K|_s$, $|P|_s$ were set to 1 and 1000, respectively). While in the outer loop of the algorithm, simulations are longer ($|K|_l$, $|P|_l$ were set to 50 and 50000, respectively). This is justified as it is not desirable to overload the simheuristic with long simulations. In order to allow the search procedure to converge, quick approximate evaluations are used in the

lower-level loops, while longer, more precise, and exhaustive simulations are reserved for the subset of promising solutions (CHICA *et al.*, 2017).

Lastly, this procedure can be easily parallelized. Multiple local searches can be executed on different CPU cores with several threads. The only objects that must be shared between threads are the solution $Incumbent$, its performance value ($Zinc$), and the list $LocalMinima$. The later can be implemented as a hash table to improve the computational performance.

6.6 COMPUTATIONAL EXPERIMENTS

The proposed simheuristic PSH was applied to an expanded dataset (Based on Chapter 2's instances) with 2625 instances, in which five buffer layouts were considered for each of the 525 task-property data vectors, with five product models each. These instances are based on the dataset provided by Otto *et al.* (2013). Instance data is made available in Lopes *et al.* (2020b)'s supplementary material, as well as the solutions obtained by PSH and the benchmarks. All tests were conducted with a Core i7-8700 (3.2GHz, 12 CPUs) with 32GB RAM. The time limit was set according to the number of tasks (3 seconds per task) in each instance for each PSH execution using 12 available threads.

The one-way ANOVA analyses have been computed with a confidence level of 99% for comparing results obtained by simulation, indicating that there are statistically significant differences whenever p-value$<$1%. Bonferroni corrections and Tukey's post-hoc test were used for multiple comparisons.

Tests were separated into two categories: (i) small and (ii) medium and large instances. For the 875 small instances, PSH is compared with four benchmarks. Its convergence behavior is analyzed and the influences of buffers on efficient solutions are discussed in light of the bowl phenomenon literature (MCNAMARA *et al.*, 2016). For 1750 large instances, the performance of PSH is compared with the best benchmark of item (i) in terms of average cycle time (inverse of throughput).

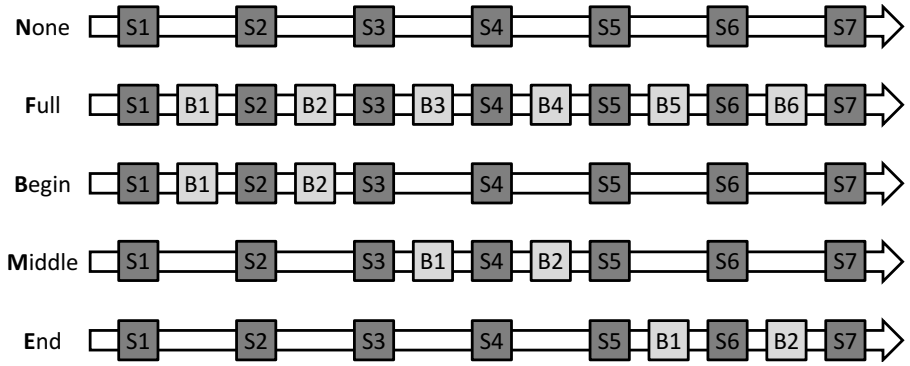
6.6.1 Small Instances: Influence of Buffer Layout

This section describes the observed impact of buffer layout in light of the "bowl phenomenon" introduced by (HILLIER; BOILING, 1979). Several authors have shown that unpaced lines with stochastic processing times can have better performance when adequately unbalanced.

Readers can refer to a recent review (MCNAMARA *et al.*, 2016) for more information. This section's tests were conducted using 875 instances generated combining 175 task-property data vectors to 5 buffer layouts. All small instances have 20 tasks, and the number of workstations was fixed to seven.

In order to verify how buffers affect the processing time distribution among stations, five buffer layouts are tested. They are presented by Figure 17. Buffers are naturally expected to enhance the performance of asynchronous assembly lines even when balancing solutions are kept unchanged (BOYSEN *et al.*, 2008). The simulator CTS presented in Section 6.4 considers buffers as dummy stations with zero processing times. Therefore, PSH takes the given buffer layout explicitly into consideration.

Figure 17 – Tested buffer layouts. Darker squares represent workstations, lighter ones represent buffers.



Source: Lopes *et al.* (2020b)

ANOVA and post-hoc tests confirm that there are differences in processing time distributions for each buffer layout (LOPES *et al.*, 2020b). In order to visualize the impact of buffer layout on balancing solutions found by PSH, the results are averaged by the following procedure: (i) the weighted (by demand rates) average processing time in each station is divided by sum of weighted average processing times in the whole line; (ii) such normalized values are averaged over 175 instances and used to determine the average normalized processing time profile for a given buffer layout, i.e., a perfectly balanced assembly line has average normalized processing time of 100% in every station. Figure 18(a) and Figure 18(b) presents the profiles for unbuffered and fully buffered layouts, respectively.

The unbuffered profile in Figure 18(a) presents an interesting bowl phenomenon variant: processing times are increased near the entrance and exit stations of the assembly line. As expected, post-hoc tests (LOPES *et al.*, 2020b) show statistically significant differences between the processing time averages of the extreme stations (1 and 7) and their neighbors (2 and

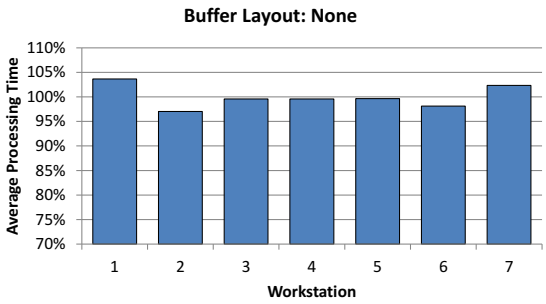
6). Interestingly, this happens despite the differences in stochastic components between the studied problems and the bowl phenomenon literature. However, the observed profile violates the "monotonicity" assumption (HILLIER; SO, 1991) of the bowl phenomenon literature. The second and second-last stations have lower average processing times (around 97%) than the center ones (around 99%). Such "ripple effect" might be justified as a way to better exploit the high average processing times at the first and last stations. In other words, the second and second-last station act as "partial buffers" for their heavy loaded neighbors. This difference in regard to standard bowl phenomenon literature can be explained by the aforementioned differences in the nature of the problem's stochastic dimensions.

The fully buffered profile (Figure 18(b)) present a much milder bowl phenomenon, with processing time averages differing at most 2.5% compared to the 6.5% observed in the unbuffered layout. Indeed, post-hoc tests (LOPES *et al.*, 2020b) still show statistically significant differences in processing time averages of the extreme stations (1 and 7) and their neighbors (2 and 6). This verifies a result from the bowl phenomenon literature, as the ideal degree of imbalance decreases when increasing of internal storage (HILLIER; BOILING, 1979). Furthermore, this resembles the desired results of vertical balancing (MERENGO *et al.*, 1999) by homogenizing the average workloads across workstations.

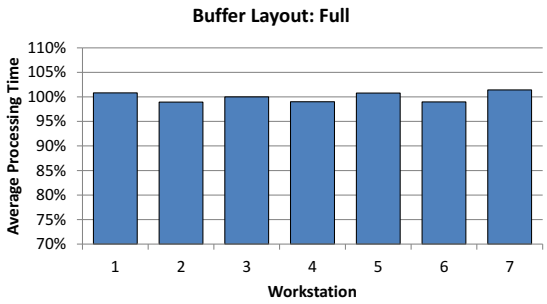
Figures 18(c) 18(e) helps explaining the differences between the None and Full layouts. For the three remaining layouts, processing times are concentrated around buffers: whether they are around the entrance of the line (Figure 18(c)), its middle (Figure 18(e)), or its exit (Figure 18(d)). Incumbent solutions concentrated, for each layout, processing times around buffers. Indeed, post-hoc tests (LOPES *et al.*, 2020b) verify statistically significant differences between key station pairs such as stations 2 and 6 for both begin and end layouts, and stations 4 and 6 for the middle layout. A possible explanation for such phenomenon corroborates the standard understanding of why the bowl phenomenon exists (MCNAMARA *et al.*, 2016): buffers can compensate for differences in processing times between models, therefore, buffered stations can afford to have longer processing times without the same risk of starving and blocking adjacent stations. The assumed infinite input and output buffers, before the first station and after the last one, further explain the profile observed in Figure 18(a). More importantly, the proposed simheuristic is shown to be able to properly take advantage of buffer layouts, hinted by the differences in average solution profiles, which are further discussed in the following sections.

Figure 18 – Weighted average processing time profiles for each buffer layout

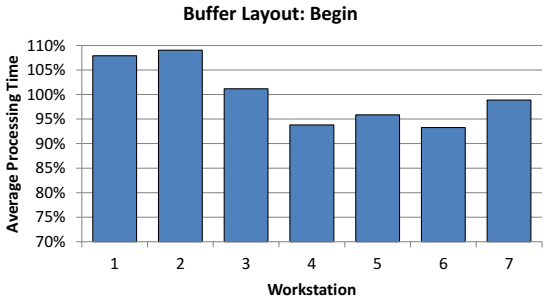
(a) Profile for the unbuffered layout



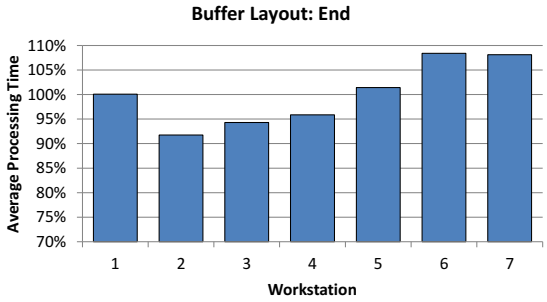
(b) Profile for the fully buffered layout



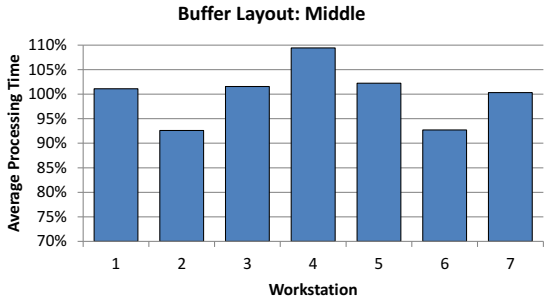
(c) Buffers between stations pairs 1-2 and 2-3



(d) Buffers between stations pairs 5-6 and 6-7



(e) Buffers between stations pairs 3-4 and 4-5



Source: Lopes *et al.* (2020b)

6.6.2 Small Instances: Comparison to Alternative Goal Functions

Due to the stochastic component of the problem, a direct measure of effective cycle time is challenging. Consequently, several indirect approaches were developed to optimize throughput indirectly. This is the case of four benchmarks considered here for comparison with PSH results: Horizontal Balancing (HBal), Vertical Balancing (VBal), Maximum Sub-cycle Time (MST), and the squared Base Model deviation plus smoothness Index (BMI). These formulations were presented by Decker (1993), Merengo *et al.* (1999), Karabati and Sayin (2003), and Venkatesh and Dabade (2008), respectively. The formal definitions of these formulations are presented by the Expressions (82), (83), (84), and (85).

$$\text{HBal: Minimize } \max_{m \in M} \left(\max_{s \in S} \left(Tx_{m,s} - \sum_{t \in T} \frac{D_{t,m}}{|S|} \right) \right) \tag{82}$$

Subject to: (79)-(81)

$$\text{Vbal: Minimize } \sum_{s_1 \in S} \left(\max_{s \in S} \left(\sum_{m \in M} O_m \cdot Tx_{m,s} \right) - \sum_{m \in M} O_m \cdot Tx_{m,s_1} \right) \tag{83}$$

Subject to: (79)-(81)

$$\text{MST: Minimize } \max_{m \in M,\ s \in S} Tx_{m,s} \tag{84}$$

Subject to: (79)-(81)

$$\text{BMI: Minimize } \sum_{m \in M} O_m \cdot \sum_{s \in S} \left(\left(Tx_{m,s} - \max_{\substack{s' \in S \\ m' \in M}} Tx_{m',s'} \right)^2 + \left(Tx_{m,s} - C_m \right)^2 \right) \tag{85}$$

Subject to: (79)-(81)

Expression (82) seeks to homogenize station-wise workload distributions for each model. Expression (83) minimizes the highest station-wise weighted average of processing times, which can be seen as a best-case scenario optimization. Expression (84) minimizes the highest processing time value, which can be seen as a worst case scenario optimization. Expression (85) attempts to balance the minimization of variability in regard to the highest value of processing time and the minimization of processing time variability for each model. In it, C_m states the theoretical minimum cycle time of model m, defined by the ceiling of the division of total task processing time by the number of workstations (SCHOLL; BECKER, 2006).

The first three benchmarks (Hbal, VBal, and MST) are either linear or easily linearizable goal functions. Given the size of instances, it is possible to compare PSH solutions to the optimal solutions of these benchmarks that have linearizable goal functions (HILLIER; LIEBERMAN, 2015). Therefore, they were converted into mixed-integer programming instances and solved with Gurobi 8.0 under the same hardware conditions PSH was executed. BMI, on the other hand, cannot be straightforwardly linearized and was instead tested using the simheuristic procedure described in Section 6.5 by replacing CTS by Expression (85) as the performance evaluation routine on Algorithm 4. The first three benchmarks produced MILP instances that were solved to optimality. The forth one (BMI) was executed with a time limit of 3 seconds per task per instance, the same used by PSH.

The average cycle time obtained for each benchmark solution was compared with the

cycle time obtained for PSH solution using long simulations of CTS (50 replications with 50,000 products each). The average cycle time improvements (avg) and standard deviations (sdv) of PSH solutions compared to each benchmark for different buffer layouts are presented in Table 24. It also presents the results of a one-way ANOVA, which show that the performance differences between benchmarks are statistically significant for all buffer layouts. Post-hoc Tukey's HSD tests (LOPES *et al.*, 2020b) confirmed (p-value$<$0.1%) that the performance difference between PSH and all benchmarks is statistically significant. Furthermore, these tests confirmed (p-value$<$0.1%) that BMI outperforms the other benchmarks, hence justifying its use for comparisons in larger instances.

Table 24 – Cycle time improvement (%) of PSH solutions compared to each benchmark for each buffer layout

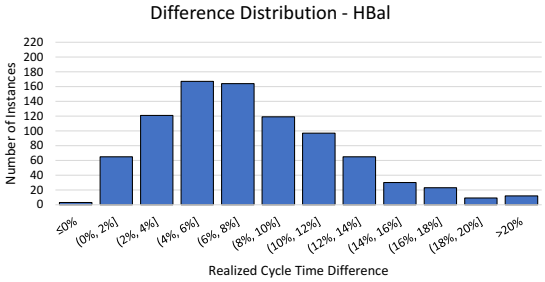
		Benchmark				
Buffer Layout	Rate	MST	Vbal	Hbal	BMI	ANOVA
None	avg	3%	10.9%	4.6%	2%	F-value=306
	sdv	1.8%	4.9%	2.5%	1.8%	p-value$<$0.01%
Begin	avg	6.3%	12.1%	6.7%	4.1%	F-value=166
	sdv	2.8%	4.9%	3.3%	2.5%	p-value$<$0.01%
Middle	avg	4.3%	9.3%	5.3%	4.1%	F-value=102
	sdv	2.5%	4.2%	3.1%	2.5%	p-value$<$0.01%
End	avg	6.7%	13.7%	7.9%	6.5%	F-value=149
	sdv	2.7%	5.4%	3.2%	2.8%	p-value$<$0.01%
Full	avg	13%	16.5%	13.1%	8.2%	F-value=131
	sdv	3.7%	4%	4%	4%	p-value$<$0.01%

Notice that all benchmarks' relative performances worsen when more buffers are added. This further highlights the proposed algorithm's better capacity to exploit buffers as resources. For the fully buffered case, PSH had solutions with homogenized station-wise averages (Figure 18(b)). However, the poor performance of VBal shows that balancing average station-wise workloads is not sufficient to ensure good stable cycle time, even for the fully buffered case. The other cases (MST, VBal and BMI) show that homogenizing model-wise workload distribution can often be better than homogenizing station-wise average workload distribution, but are also insufficient to guarantee a good performance, especially when buffers are added to the line.

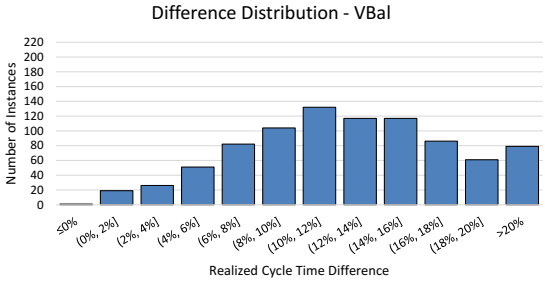
Figure 19 presents the distribution of cycle time improvement (%) of PSH solutions compared to each benchmark for all buffer layouts. Notice that, while PSH is outperformed by the benchmarks on a few instances, for the vast majority of them, it outperforms the benchmarks substantially. Out of 875 small instances, PSH obtained better results than all benchmarks on 853 (97.5% of them), performing 3.7% better in average than the instance-wise best out of the four benchmarks.

Figure 19 – Distribution of cycle time improvement (%) of PSH solutions compared to each benchmark

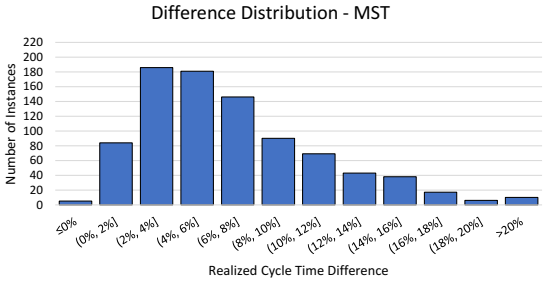
(a) HBal benchmark



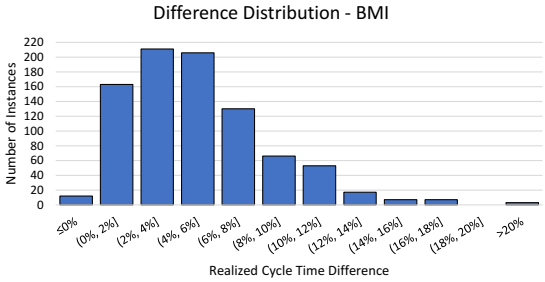
(b) VBal benchmark



(c) MST benchmark



(d) BMI benchmark



Source: Lopes *et al.* (2020b)

Table 25 – Cycle time reduction (%) for each buffer layout compared with unbuffered solutions

	PSH	MST	Vbal	Hbal	BMI
Begin	4.6%	1.6%	3.6%	2.7%	2.7%
Middle	4.1%	2.9%	5.5%	3.4%	3.1%
End	4.6%	1.1%	2.2%	1.5%	1.0%
Full	12.0%	3.5%	7.5%	4.9%	4.7%

Lastly, the impact of each buffer layout in each approach was measured by dividing the performance of buffered solutions with unbuffered ones. The cycle time improvement (%) of PSH and benchmark solutions for each buffer layout compared with the respective unbuffered solutions are presented in Table 25. Notice that the proposed method was able to better exploit buffers than all benchmarks as the improvement rates for PSH were greater than those of benchmarks. The single exception was VBal for middle layout buffer (improvement of 5.5%>4.1%), but that can be attributed to that benchmark's poor performance on the unbuffered layout (Table 24). Furthermore, notice that for all benchmarks, the "Middle" layout led to greater improvements than the "Begin" and "End" ones. This echoes established results on the bowl phenomenon literature as (HILLIER; SO, 1991) states that buffers are better assigned to the center of the assembly line. However, this was not observed for PSH: buffer allocations in the Begin and End of the line led to better performance than in the Middle (4.6% versus 4.1%). A one-way ANOVA confirms that there are statistically significant differences between the performances of PSH for

Begin, Middle, and End layouts (F-value=5.27, p-value=0.54%). This can be explained by: (i) the differences in balancing solutions that accumulate processing times around buffers (Figure 18); (ii) the aforementioned difference nature of the stochastic component of the problem.

6.6.3 Convergence Behavior

During PSH computational experiments, the time limit was set to 3 seconds per task, meaning 60, 150 and 300 seconds for small, medium and large instances, respectively. During the optimization procedure, for each instance, all incumbents were saved along with the elapsed time necessary to find them. For most instances, few solutions were found until the best incumbent. However, time required to find them was different for each instance. In order to evaluate the algorithm's convergence behavior, the following steps were undertaken: The objective value of each incumbent was divided by its instance's last incumbent's one, defining a rate to the best solution. These rates define instance-wise convergence steps, which allow a convergence curve to be drawn for each percentile. Such curves are presented by Figure 20 with a logarithmic (base 2) time axis. They can be interpreted by generalizing the following: the 10% curve indicates that only 10% of the instance convergence curves are above it and 90% are below it.

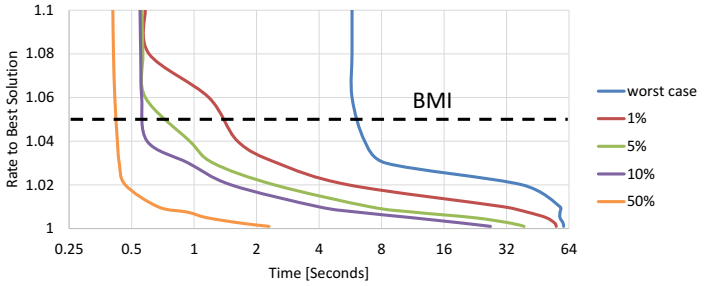
Figure 20(a) compares the best benchmark's (BMI) average performance to the convergence behavior for small instances. This highlights that PSH quickly finds solutions that outperform the benchmark's average relative performance. Figures 20(b) and 20(c) confirm that the 3 second per task time limit is reasonable compromise: Medium and large datasets display slower drops for all percentiles, meaning more instances took longer to converge, which suggests that larger time limits could lead to further improvements. However, for half of the instances (50% curves), no improvement was found after half the time limit and, for most of those that did, the improvement was lower than 2%. In the following section, PSH's performance is compared to BMI, i.e. the benchmark with best performance observed in Section 6.6.2.

6.6.4 Medium and Large Instances

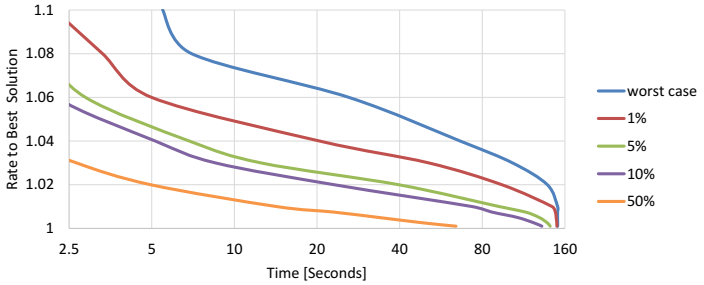
Medium instances had 50 tasks and 13 workstations, while large ones had 100 tasks and 25 workstations. Buffer layouts were defined similarly to the small instances with partially buffered layouts representing buffer allocations at the first, central and last third part of stations for Begin, Middle and End layouts, respectively (4 buffers for medium instances and 8 buffers

Figure 20 – PSH convergence behavior for each dataset

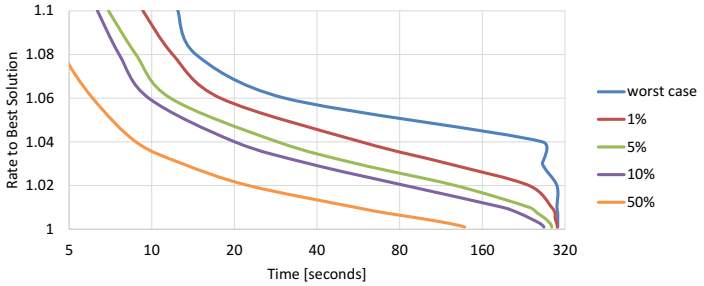
(a) Small instances



(b) Medium instances



(c) Large instances



Source: Lopes *et al.* (2020b)

for large ones). Given the results reported in Section 6.6.2, BMI was chosen as benchmark due to its better performance compared to other alternatives (HBal, VBal, MST).

A total of 1750 instances (875 medium and 875 large) were tested with the same hardware and time limit used for small instances. For each instance, the cycle time improvement of PSH compared with BMI was measured as a percentage. Furthermore, the number of local minima (number of iterations) found by both PSH and BMI were also saved. Table 26 presents a comparison between PSH and BMI for medium and large instances with different buffer layouts. For convenience, results for small instances (from Table 24) are also included in Table 26.

One of the noteworthy results reported by Table 26 is that more neighborhoods are explored for the 'Full' layout of each dataset. This reflects the optimality nature of the valid reassignment concept described in Section 6.5. Only reassignments that are not trivially dominated

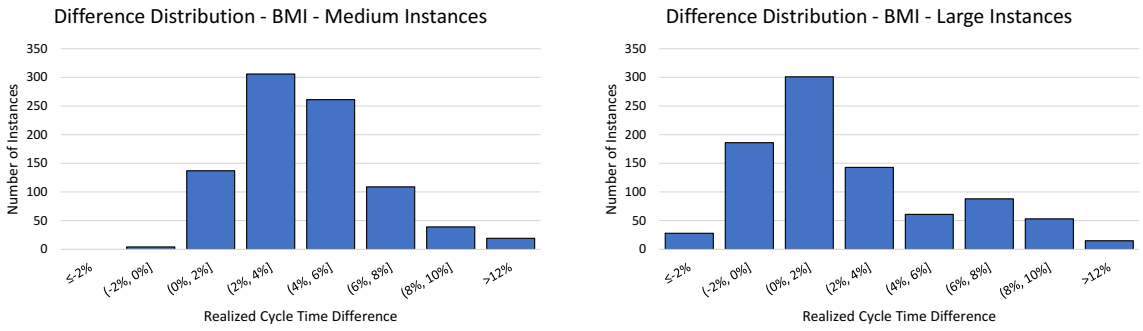
Table 26 – Cycle time improvement (%) and number of iteration comparison between PSH and BMI

Instances		Improvement avg	Improvement sdv	Iterations PSH	Iterations BMI
Small	Empty	2.0%	1.8%	2000	73800
	Begin	4.1%	2.5%	2850	
	Middle	4.1%	2.5%	2450	
	End	6.5%	2.8%	2350	
	Full	8.2%	4.0%	5750	
Medium	Empty	2.4%	1.8%	1500	5100
	Begin	4.7%	2.2%	1700	
	Middle	4.4%	2.3%	1750	
	End	4.5%	2.1%	1700	
	Full	6.3%	3.2%	3100	
Large	Empty	0.8%	1.1%	750	2700
	Begin	1.6%	1.7%	700	
	Middle	1.2%	1.5%	800	
	End	1.8%	1.7%	680	
	Full	8.3%	2.2%	1100	

by the incumbent are simulated. Given that more buffers tend to lead to better cycle times, it is only natural that better incumbents allow a larger portion of neighborhoods to be dominated. Another expected result reported by Table 26 is that increasing the instance size leads to fewer iterations. This is expected, as neighborhoods of each solution tend to increase significantly in size. The computation of the BMI performance measure is much faster than the simulation performed by CTS and, therefore, the number of neighborhoods explored using it is consistently larger than that of PSH. However, this does not necessarily reflect in better solution quality, especially when buffers are added between stations. Figure 21 presents the distribution of cycle time improvement (%) of PSH solutions compared to BMI.

Figure 21 – Distribution of cycle time improvement (%) of PSH solutions compared to BMI

(a) Comparison for medium instances **(b) Comparison for large instances**



Source: Lopes *et al.* (2020b)

Notice that Figure 21(a), similarly to Figure 19(d), reports few instances in which the difference in solution quality obtained by PSH and BMI favors BMI (difference smaller or equal than zero). However, as reported by Figure 21(b), this was more common for larger instances.

PSH outperformed BMI in 870 instances, and 661 instances (out of 875) from the medium, and large dataset, respectively. Table 26 indicates that this difference is primarily due to the instances with ‘Empty’ and ‘Middle’ layouts, indicating that it is hard to outperform BMI for these assembly line layouts. These results suggest that, for extremely large instances (more than 100 tasks), neighborhood-based searches are overwhelmed by the amount of neighboring solutions, especially if time-consuming performance measures are required. This would explain a reduced improvement of PSH over BMI by increasing the instance sizes.

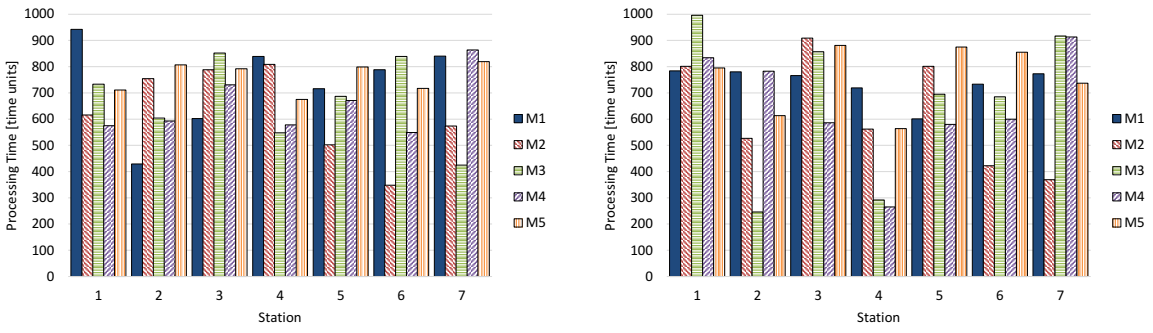
Despite difficulties observed for larger instances, both in average performance (Table 26) and number of instances (Figure 21), PSH outperformed BMI for all datasets. Furthermore, a one-way ANOVA confirms that the difference in performance is statistically significant for every instance size and buffer layout (F-value$>$14, p-value$<$0.02%). This difference was more substantial for fully buffered cases, highlighting the capacity of CTS to explicitly take buffer information into account.

6.6.5 A Case-Study on the Impact of Demand Rates

In order to evaluate the proposed simheuristic approach regarding different demand rates, an instance (small dataset, number 1, available in this Lopes *et al.* (2020b)’s supplementary material) was solved by PSH with two demand scenarios: (i) Equal Demands (ED), in which all five models have 20% demand rate, and (ii) Dominant Model (DM) scenario, in which product model 1 has a 90% demand rate, and all others have 2.5%. This leads to two different solutions, whose processing times for each product model (M1 to M5) at each station are presented by Figure 22.

Figure 22 – Processing time of solutions obtained for each demand scenario

(a) Equal Demands (ED) scenario **(b) Dominant Model (DM) scenario**



Source: Lopes *et al.* (2020b)

Figure 22 highlights product model 1 for both scenarios using a darker blue color and solid fill. Notice that its processing times are much better balanced in the DM scenario (Figure 22(b)), at the cost of higher variance of other product models. This happens in the DM scenario because model 1 is much more common than the other models. Thus, PSH balances the processing time of model 1 across stations regardless of the impact on other product models.

This is corroborated by applying the solution obtained in one scenario to the other and comparing the realized cycle times, as depicted by Table 27. Notice that applying the balancing solution obtained for one demand scenario to the other leads to worse performance. This case study confirms that solution quality is sensible to the demand rates, and illustrates PSH's capacity to take them into account.

Table 27 – Realized cycle time for each solution at each scenario

	Solution from Scenario:		Performance
Tested Scenario:	Equal Demands	Dominant Model	difference
Equal Demands	811.5	852.3	4.8%
Dominant Model	914.5	794.7	13%

6.7 CONCLUSIONS

This chapter deals with a variant of the throughput maximization of mixed-model assembly line balancing problem. It presupposes a given number of stations, asynchronous line pace, number of buffers between stations, and deterministic processing times. However, the product sequence is considered stochastic in a make-to-order environment as defined in the assembly line balancing literature. A specialized simulator CTS is proposed to measure the solution quality as well as a simheuristic PSH that finds good solutions using CTS as a performance measure.

CTS outperforms a literature object and event-oriented benchmark simulator by an order of magnitude as, expected due to its higher degree of specialization. However, due to fast simulation, it allows a more extensive use of simulations by PSH. Even not representing features as U-shaped assembly lines and parallel stations, CTS can easily be adapted to include features such as stochastic processing times and deterministic product sequence.

Computational experiments with PSH have shown that buffers affect task distributions. They allow processing times to be more substantially concentrated around buffers. Interestingly, this lead to a bowl phenomenon variant which partially verifies established results and concepts of the bowl phenomenon literature. Furthermore, surrogate benchmark goal formulations have

been compared to the proposed simheuristic PSH. The later produced better solutions for 90.8% of instances, outperforming the best benchmark in average by 5.0%, 4.2%, and 2.3% for small, medium, and large instances, respectively. Lastly, a case study has illustrated the relevance of demand rates in solution quality, as well as PSH capacity to adequately take them into account.

7 A CONTRIBUTION TO MULTI-MANNED BALANCING

This chapter is based on a note published in Journal of Manufacturing Systems (LOPES *et al.*, 2019). It is a simpler contribution, which fundamentally points out inconsistencies and limitations of a previous work (CHEN, 2017). Consequently, this chapter is less self-contained than the others in this document. The studied problem is a multi-manned assembly line balancing problem, in which the cyclical scheduling component is significantly smaller and essentially internal to each station. The solution method described in this chapter also plays a key warm-start role in a Combinatorial Bender's Cut solution strategy presented in a article published in the Operations Research Perspectives journal (MICHELS *et al.*, 2020).

7.1 CONTEXT

Multi-manned Assembly Lines Balancing Problem (MALBP) is a type of generalized assembly line that allows multiple operators (workers) to share the same workstation (station). In contrast, the Simple Assembly Line Balancing Problem (SALBP) only allows one worker per station. MALBP has three basic elements: assignment of task to workers, workers to stations, and scheduling tasks for each worker within each station. Tasks have deterministic durations, are indivisible, and must be performed by a single worker. Precedence relations between tasks impose a partial ordering in task execution and the line's cycle time (productivity requirement) limits the time a product can stay in each station. While each worker operates in a single station, multiple workers can be assigned to the same station. This means intra-station task-scheduling is required to ensure that task performance respects both precedence relations and cycle time constraints. Multiple variants of this problem have been studied (BATTAÏA; DOLGUI, 2013), but the most common ones seek to minimize the required number of workers and stations. A recent work presented a heuristic procedure and a case study based on data from a real-world assembly line (CHEN, 2017). One of the results of such case study is the following observation: allowing more workers to share the same station reduced the required number of workers. However, further examination shows that this observation is more easily attributable to features (possibly weaknesses) of the heuristic procedure, rather than to the characteristics of the problem itself or data itself. This chapter addresses that observation as well as other inconsistencies in the results reported in the paper. *Italicized* fonts are used to refer to sections and tables of Chen (2017)'s

article, while regular fonts are used to refer to this chapter.

In Chen (2017)'s *Section 4.1*, CPLEX is compared to the heuristic procedure proposed by the paper. *Table 2* states that the presented heuristic procedure obtained solutions with better fitness values (0.4516 and 0.2286) than CPLEX (0.4856 and 0.3973) for problems 4 and 5. However, *Table 3* states that CPLEX obtained solutions with fewer stations (3 and 2) and fewer workers (2 and 1) than Chen (2017)'s proposed algorithm (for problems 4 and 5 respectively). Furthermore, in *Section 4.3*, a case study is presented. In it, the solution reported in *Tables 8 and 9* requires 60 workers, and 32 stations. Nonetheless, *Figure 12* suggests that, for the case study's parameters (Maximum of three of workers per station), the solution requires 58 workers and 22 stations. Besides, *Figure 13* suggest that, for the case study's parameters (Cycle time of 172 time units.), the solution requires 60 workers and 28 stations.

The practical case study also presents a provocative result: *Figure 12* shows that increasing the maximum number of workers per station leads to fewer total workers required along the line. While this is theoretically possible (BECKER; SCHOLL, 2009), it is not very commonly observed. For instance, Kellegöz (2017) presented a 89-instance case study in which it did not happen once: in not a single instance, the required number of workers for the best solutions found was smaller than that of the equivalent SALBP instance (SCHOLL; BECKER, 2006).

Attempts to replicate the results reported by *Figure 12* started with solving the problem for SALBP version of the data presented by Chen (2017), leading to a solution with 53 workers. This solution is naturally feasible for MALBP if each worker is assigned to a different station (53 stations). Chen (2017)'s reported solutions ranged from 58 to 62 workers. Therefore, using SALBP models, a solution with fewer workers was found. A natural question followed: given the SALBP solution with 53 workers, what is the best MALBP solution? That is, what is the minimum number of stations required for the 53 workers, and their task assignments, so that precedence relations and cycle time constraints are respected in every station? In order to discover such minimum number of multi-manned stations, a residual multi-manned problem is defined, i.e. one in which task-worker assignments are parameters, while worker-station assignments and task-scheduling remain variables. A mathematical model for this problem is presented in Section 7.2, and the obtained results are presented in Section 7.3.

7.2 WORKER-STATION ASSIGNMENT AND TASK-SCHEDULING MODEL

The residual problem is defined with the following parameters: let T be the set of tasks t, each with duration D_t. Each task t is assigned to some/a worker; task-worker assignments are informed by A_t for each task t. Let W be the set of workers w. Workers must be assigned to stations (set S) such that no more than w_{max} workers are allowed per station. Tasks must be scheduled in a manner that respects the set of precedence relations (listed by (t_1,t_2 vectors in the set R) and the cycle time CT in each station. The worker-station assignment decisions are controlled by the binary variables $x_{k,j}$, which are set to 1 when the worker w is assigned to station s. The binary variable set y_s controls whether stations are open or not, and is set to 1 when the station s is open. The task-scheduling decisions are controlled by two variable sets: the continuous $Tstart_t$ control the starting time of each task t, and the binary set f_{t_1,t_2} controls the order workers perform their tasks. These variables are set to 1 when workers perform task t_1 before task t_2. The following model formally defines the residual problem.

$$\text{Minimize} \quad \sum_{s \in S} y_s \tag{86}$$

Subject to:

$$\sum_{s \in S} x_{w,s} = 1 \qquad \forall\, w \in W \tag{87}$$

$$\sum_{w \in W} x_{w,s} \leq w_{max} \qquad \forall\, s \in S \tag{88}$$

$$x_{w,s} \leq y_s \qquad \forall\, w \in W, s \in S \tag{89}$$

$$y_s \leq y_{s-1} \qquad \forall\, s \in S : s > 1 \tag{90}$$

$$Tstart_t \geq CT \cdot \sum_{s \in S} (s-1) \cdot x_{A_t,s} \qquad \forall\, t \in T \tag{91}$$

$$Tstart_t + D_t \leq CT \cdot \sum_{s \in S} s \cdot x_{A_t,s} \qquad \forall\, t \in T \tag{92}$$

$$Tstart_{t_2} \geq Tstart_{t_1} + D_{t_1} \qquad \forall\, (t_1,t_2) \in R \tag{93}$$

$$Tstart_{t_2} \geq Tstart_{t_1} + D_{t_1} - CT \cdot (f_{t_1,t_2}) \qquad \forall\, t_1,t_2 \in T : A_{t_1} = A_{t_2}, t_2 > t_1 \tag{94}$$

$$Tstart_{t_1} \geq Tstart_{t_2} + D_{t_2} - CT \cdot (1 - f_{t_1,t_2}) \qquad \forall\, t_1,t_2 \in T : A_{t_1} = A_{t_2}, t_2 > t_1 \tag{95}$$

Expression 86 states the goal function, i.e. minimizing the number of required stations. Equation 87 states that each worker must be assigned to a station. Inequality 88 states that at

most w_{max} workers can be assigned to each station. Constraint 89 states that a worker can only be assigned to stations that are open. Inequality 90 serves as a symmetry breaking constraint. Constraints 91 and 92 require tasks to be performed within the station their worker is assigned to, furthermore, these constraints impose the cycle time restriction. Inequality 93 imposes the precedence relations. Lastly, Constraints 94 and 95 require each worker to perform only one task at a time.

7.3 RESULTS

All tests reported herein were obtained using Gurobi 8.0 on a Intel Core i7-3610QM CPU (2.3GHz). Using a standard SALBP model (SCHOLL; BECKER, 2006), the solution reported by Table 28 was obtained for the problem in 135 seconds. Alternatively, any literature SALBP method such as SALOME (SCHOLL; KLEIN, 1997) can be employed to find a solution with the same number of operators.

Table 28 – Simple assembly line balancing solution with 53 workers

Worker	Tasks	Worker	Tasks	Worker	Tasks
1	49, 67	19	6	37	5, 15
2	11, 60	20	7	38	20, 34
3	16, 56	21	14, 90	39	59, 84
4	17, 41	22	43, 82	40	87, 95
5	21, 25	23	38, 39	41	47, 96
6	26, 35	24	24	42	48, 93
7	27, 42	25	40, 61	43	74
8	22, 50	26	44, 76	44	75, 88
9	29, 30	27	8	45	71, 78
10	23	28	19, 62, 66	46	86
11	36, 85	29	1, 31, 46	47	55
12	18, 63	30	32, 77	48	79
13	52, 64	31	57, 58	49	91
14	37, 69	32	9, 33, 73	50	68, 92, 94
15	28, 53	33	10, 83	51	72
16	65, 81	34	2, 51	52	89
17	12, 54	35	3, 80	53	97, 98
18	13, 45	36	4, 70		

Given that this solution respects precedence relations, it is automatically feasible with 53 workers and 53 stations. However, that number can be substantially reduced: the model described in Section 7.2 was applied to the assignment solution A_t provided by the SALBP solution reported by Table 28. The three values of w_{max} used by Chen (2017) (2, 3, and 4) were used leading to the answers reported by Table 29. The residual problems were all solved in less than 15 seconds each.

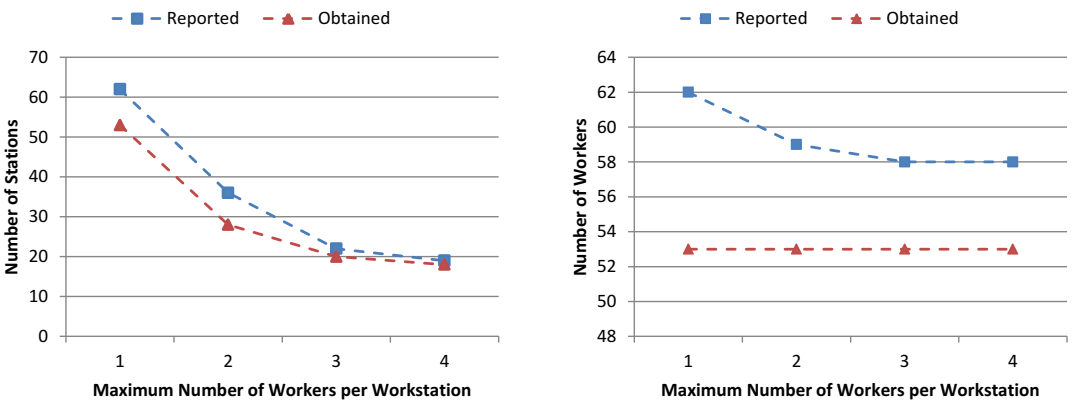
Table 29 – Multi-manned balancing solutions with 53 workers

Station	Workers w_{max}=2	Workers w_{max}=3	Workers w_{max}=4	Station	Workers w_{max}=2	Workers w_{max}=3	Workers w_{max}=4
1	1, 19	3, 4, 5	3, 4, 5, 6	15	29, 30	40, 44	47, 50
2	9, 20	6, 12, 13	7, 11, 12, 13	16	32, 34	41, 42, 48	51
3	27, 45	7, 11, 16	1, 2, 15, 23	17	28, 35	47, 50	52
4	23	2, 22, 23	8, 9, 17, 18	18	33, 36	46, 51	53
5	5, 6	14, 25, 26	16, 19, 25, 29	19	21, 37	52	-
6	2, 11	1, 15, 19	10, 20, 22, 34	20	31, 38	53	-
7	8, 25	9, 17, 18	14, 26, 27, 35	21	39, 49	-	-
8	3, 4	8, 20, 29	24, 28, 31, 36	22	40, 43	-	-
9	12, 13	10, 30, 34	21, 30, 32, 37	23	41, 44	-	-
10	10, 14	24, 27, 35	33, 43, 45	24	42, 48	-	-
11	7, 15	28, 32, 36	38, 46, 49	25	47, 50	-	-
12	16, 22	21, 33, 37	39	26	46, 51	-	-
13	24, 26	31, 38, 49	40, 44	27	52	-	-
14	17, 18	39, 43, 45	41, 42, 48	28	53	-	-

Figure 23 compares the solutions reported by Chen (2017) to those obtained by the method described herein-above. Solutions presented by Table 29 outperform those presented by Chen (2017) in both the number of workers and stations. Furthermore, the reduction in number of workers reported by Chen (2017) associated to increases in w_{max}, while theoretically possible, seems to be a consequence of the employed algorithm rather than a typical feature of the problem itself.

Figure 23 – Influence of the maximum number of workers per station

(a) Influence on number of stations **(b) Influence on number of workers**



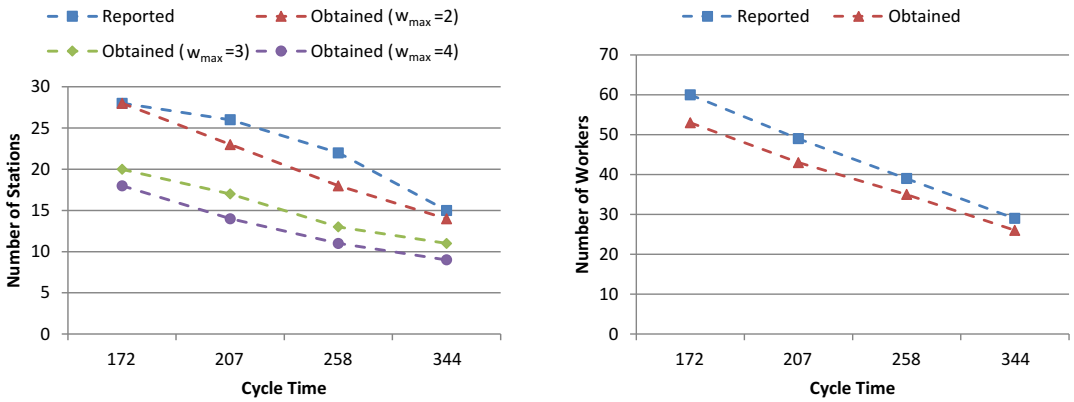
Source: Lopes *et al.* (2019)

The presented method was also tested for higher values of cycle time described by Chen (2017) (172, 207, 258, and 344) and the solutions were compared to those reported therein. Figure 24 compares the reported and obtained solutions. This time, the obtained results confirm the tendencies observed by Chen (2017), but once again the answers reported by the original article are dominated both in terms of number of workers and number of stations. Figure 24(a)

presents three curves for the number of stations, as it was not clear from Chen (2017) which value of maximum number of workers per station (w_{max}) had been employed to generate the reported values.

Figure 24 – Influence of the cycle time

(a) Influence on number of stations **(b) Influence on number of workers**



Source: Lopes *et al.* (2019)

The obtained solutions reported in the comparisons in Figures 23 and 24 are made available in Lopes *et al.* (2019)'s supporting information. They include task-worker assignments, worker-station assignments and task-scheduling that respect precedence relations and cycle time constraints. All tests required less than 150 seconds total per instance, to both find the SALBP solution and solve the residual problem.

7.4 DISCUSSION

The obtained results outperformed those reported by Chen (2017) both in number of workers and stations. Features of Chen's algorithm might contribute to that outcome. He proposes a Hybrid Simulated Annealing (HSA) composed of two key elements: A Constructive procedure that employs priority vectors for task-assignment, and an SA procedure that generates and exploits neighborhoods of such priority vectors. The constructive procedure produces solutions that maximize average station-wise worker efficiency, one station at a time. This is a greedy aspect of the algorithm that can lead to sub-optimality in large instances, since allowing some larger inefficiencies for the first stations might lead to better long-term efficiency for latter stations, and hence, to a better solution.

However, a more likely source of lower solution quality is the priority rule-based task assignments: Consider that, for the original cycle time value (CT=172), the average worker

efficiency of a solution with 53 workers (obtained via SALBP model) is 94.5%, meaning workers would only have 5.5% idle time in average. This is quite high to be randomly achieved by a priority rule (Chen (2017)'s *Table 8* reports achieving only 85.3%), specially when the greedy aspects are taken into account.

Furthermore, priority-based neighborhoods do not necessarily produce similar solutions: small perturbations in the priority vector can lead to entirely different solutions. Chen (2017)'s priority vectors, produced by the SA aspect of the algorithm (*NS* in the original article) are first converted to a task-sequence vector (*FAST* in the original article). The conversion process includes a rather arbitrary modulus operation (*Step 2.4* in the original article), which produces task-sequence vectors (*FAST*) that bear little resemblance to the SA's priority vector (*NS*). The author does not provide in his paper explanations about the purpose of such modulus operation. The *FAST* task-sequence vectors are then used to build solutions (*Steps 3-9* in the original article). A problem with this approach is that neighborhood operations on the *NS* vector tend to produce solutions that bear little resemblance to the previous solution, i.e. task-station and task-worker assignments in neighbor solutions tend to be unrelated.

Perturbation operations more directly related to the actual key operational decisions (task-worker assignments) would have greater chances of producing promising neighborhoods and successful search procedures. Naturally, this can impose the challenge of more complex codifications. Further works should, therefore, attempt to balance the codification's simplicity and its distance to actual operational decisions.

7.5 CONCLUSIONS

In this chapter, inconsistencies in the results reported by Chen (2017) are highlighted. Attempts to solve the practical case study reported therein lead to solutions that dominate those presented by Chen (2017) in both number of workers and number of stations. Moreover, these solutions were obtained in relatively small computational times (less than 150 seconds per instance). The method used to find these solutions is hereby described, and the solutions themselves are made available as supporting information. Furthermore, Chen (2017)'s reported results suggested that increasing the number of allowed workers per station reduces the number of required workers. This chapter shows that such result, while theoretically possible, is uncommon and the fact that it occurred in Chen (2017)'s case study is more easily attributable to characteristics of the employed algorithm (possibly weaknesses) than to the problem's specific data and nature.

Features of Chen (2017)'s algorithm that might cause the aforementioned performance issues are discussed.

8 A PARADIGM SHIFT FOR MULTI-MANNED BALANCING

This chapter is based on an article published in the journal Omega (LOPES *et al.*, 2020). In it, a paradigm shift is proposed for multi-manned assembly lines. Typically, these lines consider discrete stations with rigid boundaries within which workers can perform tasks. Under such consideration, line pace does not affect the problem definition. However, as shown in Chapter 3, the line's physical pace can be tied to relevant opportunities if explicitly considered. This chapter considers how to benefit from continuous line pace in the context of multi-manned assembly lines by allowing flexible station boundaries.

8.1 INTRODUCTION

Assembly lines are production systems commonly used in the large-scale production of standardized or similar products (SCHOLL, 1999). In them, the workpiece flows through the workstations, moved by a transportation system. The time available for operators or machines in each station to perform tasks is called cycle time, and is tied to the system's production rate. The optimization problem of distributing workload among the stations is called assembly line balancing problem. Naturally, there are many different features among assembly lines (BECKER; SCHOLL, 2006) and, therefore, it is necessary to make assumptions regarding the referred system.

The most commonly studied type of balancing problem is the simple assembly line balancing (SALB), which models assembly lines with several restrictive hypotheses (BAYBARS, 1986). First studied by Salveson (1955), the problem is defined (SCHOLL; BECKER, 2006) by: production of one standardized product, fixed process, serial paced line with common cycle time, tasks sequence subjected to precedence constraints, deterministic task times, each task must be assigned to only one station, and all stations are equally equipped.

While SALB is the most studied balancing problem, it has several limitations regarding its applicability. Due to its simplifying assumptions, not all assembly lines are accurately described by the SALB formulation. Consequently, multiple researchers have studied and described variants that remove or replace SALB's simplifying hypotheses (BOYSEN *et al.*, 2008; BOYSEN *et al.*, 2009c; BATTAÏA; DOLGUI, 2013). These generalized assembly line balancing (GALB) works take into account different aspects of an assembly line, such as product diversity

(PASTOR *et al.*, 2002; KARABATI; SAYIN, 2003; AKPINAR; BAYKASOGLU, 2014a), skilled-workers allocation (COROMINAS *et al.*, 2008), ergonomic restrictions (ALGHAZI; KURZ, 2018; TIACCI; MIMMI, 2018), traveling workers (SIKORA *et al.*, 2017), lines with non-deterministic task times (PEREIRA; ÁLVAREZ-MIRANDA, 2018), and parallelism. Parallelism can take multiple forms, such as parallel lines (ÖZCAN *et al.*, 2010; KUCUKKOC; ZHANG, 2015), parallel workstations (TIACCI, 2015b), and parallel work within stations (DIMITRIADIS, 2006; KELLEGÖZ, 2017). This chapter focuses on the later characteristic, which is the defining feature of two-sided and multi-manned assembly lines.

Two-sided assembly lines were first studied by Bartholdi (1993). Since then multiple variants have been proposed, taking into account factors such as product diversity (ÖZCAN; TOKLU, 2009) and stochastic task times (ÖZCAN, 2010). Two-sided Assembly Line Balancing (TALB) allows up to two workers to operate simultaneously in the product: one at the left side and the other at the right side. Due to precedence constraints, it is necessary to schedule tasks within each station to respect the cycle time constraint.

Multi-manned Assembly Line Balancing (MALB) further generalizes TALB by allowing up to w_{max} (with $w_{max} \geq 2$) workers to perform tasks simultaneously in the same workpiece. The maximum number w_{max} of workers that can work in parallel is primarily determined by the product's size. Hence, multi-manned assembly lines are often dedicated to the production of large products, such as in the automobile industry (CHEN, 2017). The most common practical application is found in the final assembly of vehicles. Dimitriadis (2006) was one of the first to study MALB, establishing basic formulations and concepts. Multiple variants have been presented and studied since then: Becker and Scholl (2009) introduced various MALB with variable parallel workplaces formulations, such as a model that allows the allocation of extra-long tasks. Moon *et al.* (2009), Yilmaz and Yilmaz (2016) studied a problem variant with workers with different skills and wages. Chang and Chang (2010), Roshani and Nezami (2017) present mixed-model multi-manned assembly line models and solution methods. Kazemi and Sedighi (2013) developed a cost-oriented model for mixed-model multi-manned assembly lines. Roshani and Giglio (2016) introduced cycle time minimization variants.

For some MALB definitions, the cycle time is given and the objective is to minimize the required number of workers and stations. Fattahi *et al.* (2011) developed a mixed-integer linear programming (MILP) formulation used to optimally solve small-size instances and an ant colony optimization metaheuristic to solve larger instances. Kellegöz and Toklu (2012), Kellegöz

and Toklu (2015) presented exact and heuristic solution methods for that problem. More recently, Kellegöz (2017) developed a new MILP formulation with fewer variables and constraints, as well as a simulated annealing method to solve medium and large-size problems. These outperformed Fattahi *et al.* (2011)'s model and algorithm, and, therefore, Kellegöz (2017)'s instances and results are used as a benchmark for this chapter.

While it is possible to provide specific examples in which multi-manned stations allow a reduction of the required number of workers (BECKER; SCHOLL, 2009), this was not the case in any of the instances tested by the literature benchmark (KELLEGÖZ, 2017): the number of stations is substantially reduced when compared to the equivalent SALB instance, but the number of workers remained unchanged. Therefore, the main contribution of MALB, evidenced in literature, is to reduce the required space (stations). This is particularly relevant considering that multi-manned lines are most often employed to produce large products such as vehicles. Therefore, the line length tends to be long, and reducing it implies on saving significant amounts of money, especially on the context of line design.

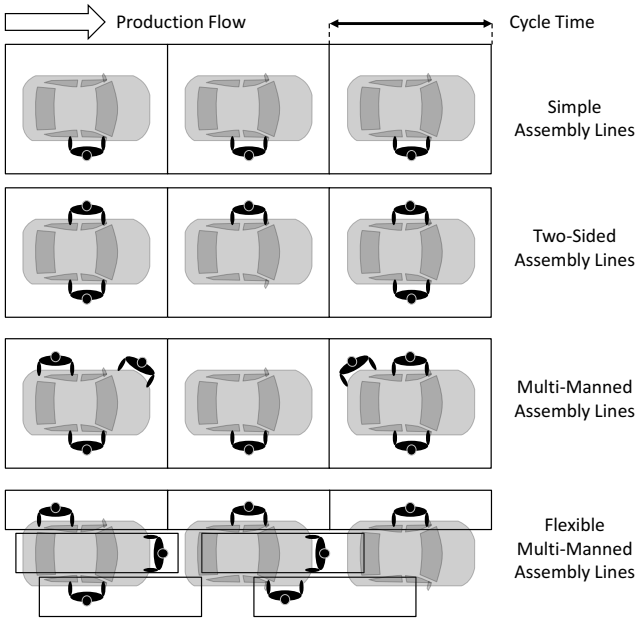
Line control (e.g., paced, unpaced asynchronous or unpaced synchronous) is a key characteristic of assembly lines (BOYSEN *et al.*, 2008). While most MALB works consider a continuous paced line, works on unpaced lines demonstrate the importance of explicitly taking line control into account. Unpaced lines can be further classified as synchronous (KARABATI; SAYIN, 2003; CHIANG *et al.*, 2012; URBAN; CHIANG, 2016), asynchronous (SAWIK, 2000; SAWIK, 2002; ÖZTÜRK *et al.*, 2013; TIACCI, 2015a), and hybrid (KOUVELIS; KARABATI, 1999). Chapter 3 shows that it's possible to obtain better solutions by explicitly taking line control into account in the mathematical definition of the problem.

In order to better exploit line control, multiple authors have presented flexible variants to assembly lines. For unpaced lines, it is common to incorporate parallel stations (SAWIK, 2004; ÖZTÜRK *et al.*, 2015) and buffers (SAWIK, 2012; TIACCI, 2015b). Flexible definitions have also been applied to paced lines, such as open stations (SARKER; PAN, 2001) that allow line length minimization (DEFERSHA; MOHEBALIZADEHGASHTI, 2018). Incorporating these flexibilities tends to offer superior performance. However, it might also imply on more complex formulations and more challenging problems.

As aforementioned, most previous works on multi-manned lines consider a paced continuous line. However, station boundaries are rigid, and the specific features of such line control are not explicitly considered: for most previous works (mathematical models and algorithms),

changing the line control from paced to unpaced (synchronous or asynchronous) would have no apparent consequence. This suggests that the continuous paced nature of the line controls might not have been properly explored. Contrary to discrete or unpaced lines, paced ones allow stations to have arbitrary starting and ending positions. Given that the line moves forward at constant speed, the length of the station is superiorly bounded by the cycle time. Adding such station-boundary flexibility to the problem defines the flexible multi-maned assembly lines. Figure 25 conceptually compares the different herein-above described line types: simple, two-sided, multi-manned, and flexible multi-manned. Notice that, for the later, at any given line position, at most w_{max} (in this case, $w_{max} = 3$) workers operate on the workpiece at the same time. Therefore, assembly line segments can have different numbers and sets of workers performing tasks on the product. Furthermore, the positions in which such numbers and sets change are flexible.

Figure 25 – Comparison of line concepts.



Source: Lopes *et al.* (2020)

Previous multi-manned formulations allow multiple workers to operate simultaneously, but require that they share starting and ending station positions if they do so. If this restriction is eliminated, workers would have more flexibility, leading to shorter line lengths - which is one of the main contributions of MALB when compared to SALB. Naturally, this requires more complex constraints to ensure simultaneous respect to precedence relations, cycle time, and the maximum number of operators working at the same workpiece. Furthermore, line balancing problems are known to be NP-Hard as they can be seen as generalized bin packing, which is

NP-Hard (ÁLVAREZ-MIRANDA; PEREIRA, 2019). FMALB is at least as complex as SALB given that every feasible SALB solution is a feasible FMALB solution. Therefore, FMALB is NP-Hard, which motivates the development of heuristic procedures. The purpose of this chapter is to introduce this Flexible Multi-Manned Assembly Line Balancing (FMALB) problem, as well as present a formal mathematical definition with a MILP model, a heuristic solution method, and lower bounds for it.

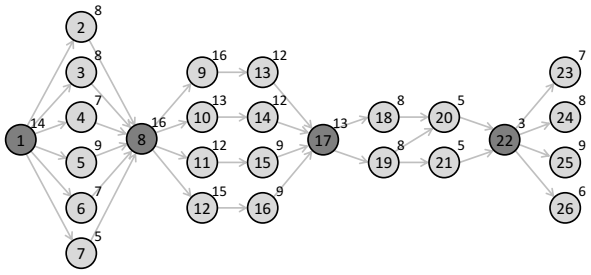
This chapter is structured as follows: the problem is presented and described in Section 8.2. Section 8.3 presents the exact mathematical model. The proposed heuristic solution method is presented in Section 8.4. Section 8.5 presents the lower-bounds defined for this problem. Section 8.6 presents the results obtained on a literature dataset, compares the solution to those of a multi-manned assembly line balancing benchmark, and presents a study on the relationship between efficient values of cycle time and line length. Section 8.7 summarizes this chapter's contributions.

8.2 PROBLEM STATEMENT

The Flexible Multi-Manned Assembly Line Balancing Problem (FMALB) consists on assigning a set of tasks T, with given durations D_t for each task t, to a set of workers W, and consistently schedule them such that both precedence relations (R) and the cycle time (CT) are respected. Workers are represented by integers listed in W=$\{1,2,...,|W|\}$, and the number of workers ($|W|$) is a parameter. They are ordered by their starting position. Each worker can only perform one task at a time, and at most w_{max} workers can operate simultaneously in the same workpiece. The problem's goal is to minimize the line length L, a continuous variable. By hypothesis, the line's speed v is normalized at one Length Unit (L.U.) per Time Unit (T.U.). The time required by the worker to walk back to the starting position is neglected, a common hypothesis for MALB (FATTAHI *et al.*, 2011; KELLEGÖZ, 2017). If such time is not negligible, then it can simply be subtracted from the cycle time. Figure 26 is used as an example of precedence diagram: in it, the numbers within circles indicate the task indexes (elements of set T), and those on the upper right corner indicate processing times(D_t). The arrows indicate direct precedence relations between tasks, i.e. the elements of set R. They are represented mathematically by ordered pairs (t_1,t_2) interpreted as "t_1 must precede t_2". By hypothesis, tasks are topologically ordered, meaning $t_2 > t_1$ for all precedence relations, as in Figure 26. This is a common consideration for assembly line formulations (SCHOLL, 1999;

OTTO *et al.*, 2013; SCHOLL *et al.*, 2013).

Figure 26 – Illustrative problem's precedence diagram



Source: Lopes *et al.* (2020)

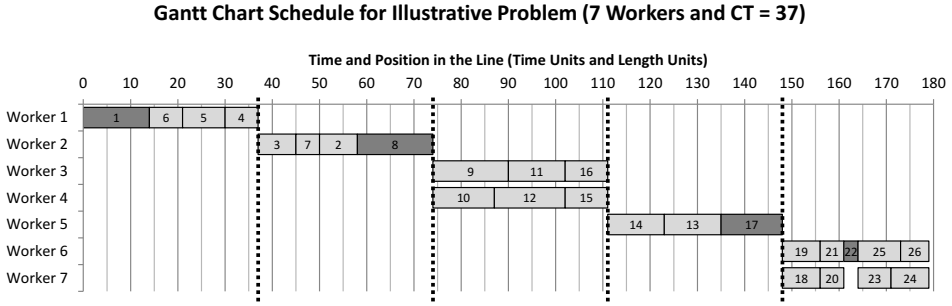
Consider an instance with seven workers ($|W| = 7$), a cycle time of 37 T.U. ($CT = 37$), and task properties (durations and precedence relations) given by the diagram in Figure 26. Figure 27 compares the optimal answers of regular and flexible multi-manned balancing-scheduling problem. These answers were obtained by employing Kellegöz (2017)'s model and the MILP model described in Section 8.3, respectively. In both solutions, the number of workers that simultaneously operate in a workpiece is limited by three ($w_{max} = 3$). Also, workers execute at most one task at a time. The main difference between the regular (Figure 27(a)) and flexible (Figure 27(b)) Multi-Manned Assembly Line Balancing (MALB) problem is their stations boundary definition: The regular version imposes strictly equal starting and finish positions for workers that operate simultaneously, as represented by the dashed lines in Figure 27(a). The flexible version, however, requires only that each worker finishes his tasks within the cycle time (time between starting the first task and finishing the last one), but they are not required to do so within fixed station borders. For instance, in Figure 27(b), Workers 1 and 2 perform tasks simultaneously in the workpiece from position 16 L.U. to 37 L.U.. However, they start at different positions (0 L.U. and 16 L.U., respectively). Workers 6 and 7 share starting positions and operate in parallel from position 103 L.U. to 134 L.U.. A similar parallelism occurs with Workers 3, and 4. However, between positions 69 L.U. and 90 L.U., workers 3, 4, and 5 operate in triple parallelism.

Notice that the spaces in which each worker operates are flexible, in that, while workers can operate from the same starting to ending position (like Workers 6 and 7 in Figure 27(b)), they are not required to do so. This is particularly valuable if there are many precedence relations that form "bottleneck" like tasks, such as tasks 1, 8, 17, and 22, highlighted in dark gray in Figures 26 and 27. In this example, the regular MALB optimal solution (Figure 27(a)) has less parallelism and, consequently, requires a longer line length.

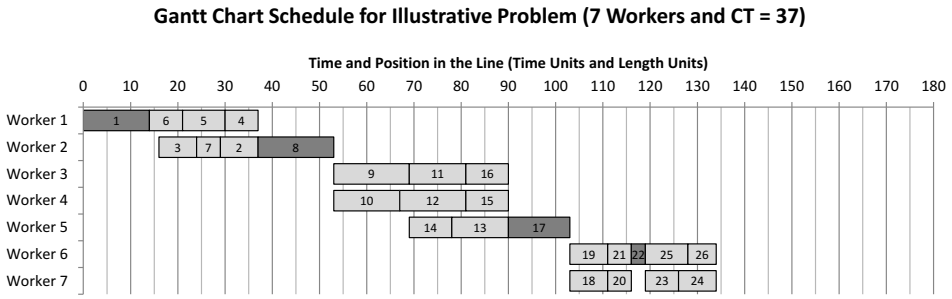
This example highlights how special structures in the precedence relations might limit

Figure 27 – Comparison between flexible and regular multi-manned optimal balancing-sequencing solutions

(a) Standard multi-manned optimal solution for $|W| = 7$ and $CT = 37$ with $L = 179$



(b) Flexible multi-manned optimal solution for $|W| = 7$ and $CT = 37$ with $L = 134$



Source: Lopes *et al.* (2020)

the degree to which MALB can lead to reductions in line length in comparison to sequential SALB lines without parallelism. By allowing station boundaries to be flexible, the FMALB formulation allowed a reduction of 25% (179 to 134 L.U.) in line length when compared to MALB in this example. As shown in Section 8.6, this might not be a rare event. However, a major modeling difficulty arises from such flexibility, namely describing a schedule in which, simultaneously: cycle time is respected by all workers; precedence relations between tasks are enforced; and at most w_{max} workers perform operations at the workpiece at any given time. Section 8.3 describes the proposed modeling approach to this challenge.

8.3 MATHEMATICAL MODEL

The Mixed-Integer Linear Programming (MILP) model is based on two binary variable sets: $x_{t,w}$ (assignment variable set), whose variables are set to one when task t is assigned to worker w, and f_{t_2,t_1} (ordering variable set), whose variables are set to one when task t_2 is performed after task t_1 is completed. Three continuous variable sets are employed as scheduling variables: $Tstart_t$ states the starting time of task t, $Wstart_w$ and $Wend_w$ state the starting and finishing times of worker w's operations. The FMALB formal mathematical definition is given

by Expressions (96) to (106):

$$\text{Minimize} \quad L \tag{96}$$

Subject to:

$$\sum_{w \in W} x_{t,w} = 1 \qquad \forall\, t \in T \tag{97}$$

$$Tstart_{t_2} \geq Tstart_{t_1} + D_{t_1} \qquad \forall\, (t_1,t_2) \in R \tag{98}$$

$$Wend_w - Wstart_w \leq CT \qquad \forall\, w \in W \tag{99}$$

$$Wstart_w \leq Tstart_t + H \cdot (1 - x_{t,w}) \qquad \forall\, t \in T, w \in W \tag{100}$$

$$Wend_w \geq Tstart_t + D_t - H \cdot (1 - x_{t,w}) \qquad \forall\, t \in T, w \in W \tag{101}$$

$$f_{t_1,t_2} + f_{t_2,t_1} \geq x_{t_1,w} + x_{t_2,w} - 1 \qquad \forall\, w \in W, t_1,t_2 \in T : t_2 > t_1 \tag{102}$$

$$Tstart_{t_2} \geq Tstart_{t_1} + D_{t_1} - H \cdot (1 - f_{t_2,t_1}) \qquad \forall\, t_1,t_2 \in T : t_2 \neq t_1 \tag{103}$$

$$L \geq Tstart_t + D_t \qquad \forall\, t \in T \tag{104}$$

$$Wstart_w \geq Wend_{w-P} \qquad \forall\, w \in W : w > w_{max} \tag{105}$$

$$Wstart_w \geq Wstart_{w-1} \qquad \forall\, w \in W : w > 1 \tag{106}$$

Expression (96) states the goal function, which is line length minimization. Equation (97) states the occurrence constraint: each task is performed by one worker. Inequalities (98) and (99) state precedence and cycle time constraints, respectively, in terms of the scheduling variables. Inequalities (100) and (101) state that, if a task is assigned to a worker, it must be performed between his starting and ending positions. Inequality (102) states that if two tasks are assigned to the same worker, one of them must be performed before the other. Inequality (103) enforces the desired effect of the f_{t_2,t_1} variable set. Inequality (104) states that all tasks must be performed within the line. Constraint (105) allows that at most w_{max} workers to operate simultaneously, and Constraint (106) complements Constraint (105) with an ordering requirement. The Big-M factor H in Constraints (100), (101), and (103) is a large integer, that acts as a relaxation factor. One valid value of H is the sum of tasks' processing times, however, any feasible value of L is also a valid value of H.

8.4 MILP-BASED HEURISTIC PROCEDURE

The model presented in Section 8.3 tends to be computationally difficult to solve. This mainly occurs because all binary variables are tied to Big-M constraints (Inequalities (100), (101), and (103)). In order to generate good solutions for medium and large cases, an MILP-based heuristic procedure is hereby proposed. Its main strategy is presented in Section 8.4.1. The two employed MILP models are presented in Section 8.4.2.

8.4.1 Method Overview

The proposed heuristic procedure is based on Combinatorial Benders' Cuts (CODATO; FISCHETTI, 2006): the original FMALB instance, hereafter referred to as "Main Problem", is separated in two parts according to the type of binary decision variables: assignment ($x_{t,w}$) and order (f_{t_2,t_1}). The Main Problem is split into a Master Problem (MP) - limited to the assignment variables, and a Slave Problem (SP) - limited to order and scheduling variables. SP employs the assignment variables of MP solutions as parameters and any feasible solution to the SP is feasible for the Main Problem. This choice is explained by the hierarchal relationship between assignment and ordering decision variables: Constraint (102) states that assignment variables only affect order variables for tasks assigned to the same worker. This justifies deeming the FMALB's assignment dimension as the Master Problem, and the order/scheduling part as the Slave Problem.

Every time MP finds an assignment solution that can be feasible in regard to the Main Problem, SP is solved for the assignment vector provided by MP. A Combinatorial Benders' Cut is added to the main problem regardless of the solution found by SP. This can be implemented using callbacks and lazy constraints, techniques available for both Gurobi and CPLEX, two broadly used universal solvers for MILP problems.

Contrary to straightforward standard Combinatorial Benders' Cuts Method (CODATO; FISCHETTI, 2006), the proposed procedure is not exact, but rather heuristic: The employed Master Problem is not simply a relaxation of the Main Problem, but rather an approximation that incorporates a version of the precedence relations that can eliminate feasible solutions on the Main Problem. Furthermore, the Slave Problem is not solved to optimality, but rather to a short time limit.

8.4.2 Master and Slave Mathematical Models

Consider the model defined by Expressions (107) to (110). It is a standard SALB-2 Model (SCHOLL; BECKER, 2006) and is much more computationally tractable than the Main Problem defined in Section 8.3: its constraints state occurrence (Equation (108)), precedence (Inequality (109)), and cycle time (Inequality (110)) without the need of scheduling variables. This model is hereafter referred to as "Master Problem".

$$\text{Minimize} \quad CT' \tag{107}$$

Subject to:

$$\sum_{w \in W} x_{t,w} = 1 \qquad \forall\, t \in T \tag{108}$$

$$\sum_{w \in W} w \cdot x_{t_1,w} \leq \sum_{w \in W} w \cdot x_{t_2,w} \qquad \forall\, (t_1,t_2) \in R \tag{109}$$

$$\sum_{t \in T} D_t \cdot x_{t,w} \leq CT' \qquad \forall\, w \in W \tag{110}$$

Any task-assignment obtained by the Master Problem with a cycle time value (CT') lesser or equal to the Main Problem's parameter (CT) is guaranteed to generate feasible solutions for the Main Problem. This occurs because every feasible SALB solution can be converted to feasible FMALB solutions. Let A_t state the worker to which each task t is assigned in the solution given by the Master Problem. Its feasibility and goal function in regard to the Main Problem can be checked by the "Slave Problem" defined by the Expressions (111) to (115):

$$\text{Minimize} \quad L \tag{111}$$

Subject to (98), (99), (104)-(106) and:

$$Wstart_{A_t} \leq Tstart_t \qquad \forall\, t \in T \tag{112}$$

$$Wend_{A_t} \geq Tstart_t + D_t \qquad \forall\, t \in T \tag{113}$$

$$Tstart_{t_2} \geq Tstart_{t_1} + D_{t_1} - H \cdot (1 - f_{t_2,t_1}) \qquad \forall\, t_1,t_2 \in T : t_2 > t_1, w_{t_1} = w_{t_2} \tag{114}$$

$$Tstart_{t_1} \geq Tstart_{t_2} + D_{t_2} - H \cdot (f_{t_2,t_1}) \qquad \forall\, t_1,t_2 \in T : t_2 > t_1, w_{t_1} = w_{t_2} \tag{115}$$

The hierarchal relationship between task-worker assignment decisions and task-scheduling decisions is reflected in the strength of the Slave Problem's constraints: Notice

that Constraints (112) to (115) are stronger versions of Constraints (100) to (103): Inequalities (112) and (113) do not require the Big-M relaxation factor H because the assignment of tasks are parameters in the Slave Problem. Constraints (114) and (115) employ less than half of binary f_{t_2,t_1} variables and Big-M relaxations required by Constraints (102) and (103). This is only possible because the task assignment is a parameter, and hence, the sets of tasks assigned to the same workers are known. The number of constraints is also substantially reduced as they can be defined only for the required cases.

Every solution found by the Slave Problem is feasible in regard to the Main Problem (FMALB). This means that when the Slave Problem is solved, its solutions can be directly compared in goal function to the incumbent, and eventually replace it. In either case, the current Master Problem assignment solution A_t is used to generate the combinatorial Benders' Cut presented by Inequality (116) to be added to the Master Problem problem (Provided $CT' \leq CT$).

$$\sum_{t \in T} x_{t,A_t} \leq |T| - 1 \tag{116}$$

If the Master Problem's search field is exhausted within the time limit, the heuristic procedure is terminated. Because its precedence constraint (Inequality (109)) is more restrictive than that of the Main Problem, it is not possible to assure the optimality of the incumbent. Section 8.5 presents the lower bound efforts undertaken to address this limitation of the proposed heuristic procedure.

8.5 LOWER BOUNDS

In order to measure the quality of the obtained values of line length, two lower bounds are defined. In this Section, their high-level pseudo-codes are presented and explained. The instance's information is: the tasks' duration (D_t), precedence relations (R), the cycle time (CT), the number of available workers ($|W|$), and the maximum number of parallel workers (w_{max}). By dividing the sum of processing times by the maximum degree of parallelism ($\sum_{t \in T} D_t / w_{max}$), a trivial bound on line length is obtained. A stronger version of this perfect division bound is presented by Algorithm 5.

By considering account the number of available workers, Lower Bound 1 takes into account the following fact: when the number of workers is not divisible by w_{max}, some of them will not work 100% of their time in maximum parallelism capacity (w_{max} workers simultane-

Algorithm 5 – Lower Bound 1: Perfect Division of Processing Times

```
1: function LowerBound1(T, D, CT, |W|, w_max)
2:     reqProc ← Σ_{t∈T} D_t                          ▷ Total processing required
3:     avWorkers ← |W|                                 ▷ Total workers available
4:     lb1 ← 0
5:     while reqProc ≥ CT · w_max and avWorkers > P do   ▷ Best case full station
6:         reqProc ← reqProc − CT · w_max
7:         avWorkers ← avWorkers − w_max
8:         lb_1 ← lb_1 + CT
9:     end while
10:    lb_1 ← lb_1 + ⌈reqProc/ min(avWorkers,w_max)⌉   ▷ Lower bound for remaining length
11:    return lb_1
12: end function
```

ously). This perfect division bound is also used to compute lower bounds on sub-problems as a part of the critical path bound (Lower Bound 2), presented by Algorithm 6.

The critical path bound is presented by Algorithm 6. In order to compute it, the precedence relations set (R) is generalized to include all indirect relations (R^*). In the algorithm's first step (Lines 2 to 9), for each task, an initial bound on line length required by its successors and predecessors is computed: For each task t, the set of predecessors (Tp_t) and successors (Ts_t) is computed (Lines 3 and 4); The minimum number of dedicated workers required by t's predecessors ($minWp_t$) and successors ($minWs_t$) are computed with floor operations based on the tasks' processing times (Lines 5 and 6). The first approximation for the minimum length required to perform all the predecessors of t ($minLp_t$) is computed using Algorithm 5, with Tp_t as the set of relevant tasks, and $|W| - minWs_t$ as the number of available workers (Line 7). This subtraction occurs because the $minWs_t$ workers, that are fully dedicated to successors of t, cannot contribute to the predecessors. This is also the reason why Lines 4 and 5 employ floor operations rather than ceiling ones. The analogous process is required to compute the minimum length required to perform all the successors of t ($minLs_t$) in Line 8.

In the algorithm's second step (Lines 10 to 17) the critical path for predecessors of each task is computed. For each task t_a and for each of t_a's predecessors t_b, a lower bound is computed for the in-between length (I_{t_b,t_a}) required by the set of tasks that are simultaneously successors of t_b and predecessors of t_a. Algorithm 5 is once again used for that purpose, and the number of available workers is decreased by the number of workers that are dedicated to successors of t_a and predecessors of t_b (Line 13). The minimal starting position ($minLp_{t_a}$) of task t_a is then revised in Line 14 if a more critical path is found in task t_b. In order for this to be most accurate, t_b's critical path must be already computed. However, tasks are topologically ordered (by hypothesis), and following such order (Lines 10 and 11) ensures the desired consistency.

Algorithm 6 – Lower Bound 2: Critical Path Bound

1: **function** LOWERBOUND2(T, R^*, D, CT, $|W|$, w_{max})
2: **for all** $t \in T$ **do**
3: $Tp_t \leftarrow \{t_p \in T | (t_p,t) \in R^*\}$ ▷ Set of tasks that precede t
4: $Ts_t \leftarrow \{t_s \in T | (t,t_s) \in R^*\}$ ▷ Set of tasks that succeed t
5: $minWp_t \leftarrow \lfloor \sum_{t_p \in Tp_t} D_{t_p} / CT \rfloor$ ▷ Minimum number of workers exclusively required by Tp_t
6: $minWs_t \leftarrow \lfloor \sum_{t_s \in Ts_t} D_{t_s} / CT \rfloor$ ▷ Minimum number of workers exclusively required by Ts_t
7: $minLp_t \leftarrow$ LowerBound1(Tp_t,D,CT,$|W| - minWs_t$,w_{max}) ▷ Minimum line length before t
8: $minLs_t \leftarrow$ LowerBound1(Ts_t,D,CT,$|W| - minWp_t$,w_{max}) ▷ Minimum line length after t
9: **end for**
10: **for** $t_a = 1;\ t_a \leq |T|;\ t_a$**++ do** ▷ Topological order
11: **for** $t_b = 1;\ t_b \leq t_a - 1;\ t_b$**++ do**
12: **if** $(t_b,t_a) \in R^*$ **then** ▷ Improve bound based on tasks between t_b and t_a
13: $I_{t_b,t_a} \leftarrow$ LowerBound1($Tp_{t_a} \cap Ts_{t_b}$,D,CT,$|W| - minWs_{t_a} - minWp_{t_b}$,$w_{max}$)
14: $minLp_{t_a} \leftarrow \max(minLp_{t_a}, minLp_{t_b} + D_{t_b} + I_{t_b,t_a})$
15: **end if**
16: **end for**
17: **end for**
18: **for** $t_a = |T|;\ t_a \geq 1;\ t_a$**- - do** ▷ Reverse topological order
19: **for** $t_b = |T|;\ t_b \geq t_a + 1;\ t_b$**- - do**
20: **if** $(t_a,t_b) \in R^*$ **then** ▷ Improve bound based on tasks between t_a and t_b
21: $I_{t_a,t_b} \leftarrow$ LowerBound1($Ts_{t_a} \cap Tp_{t_b}$,D,CT,$|W| - minWp_{t_a} - minWs_{t_b}$,$w_{max}$)
22: $minLs_{t_a} \leftarrow \max(minLs_{t_a}, minLs_{t_b} + D_{t_b} + I_{t_a,t_b})$
23: **end if**
24: **end for**
25: **end for**
26: $lb_2 \leftarrow 0$
27: **for all** $t \in T$ **do** ▷ Tasks bind length by their duration added to preceding and succeeding length bounds
28: $lb_2 \leftarrow \max(lb_2, minLp_t + D_t + minLs_t)$
29: **end for**
30: **return** lb_2
31: **end function**

The analogous procedure for successors, and for $minLs_{t_a}$, is performed (Lines 18 to 25) in the reverse topological order (Lines 18 and 19) for the same consistency reason.

Once the critical paths before and after each task are computed, the Lower Bound 2 is computed in Lines 26 to 30: The duration of each task, summed to the critical paths for both predecessors and successors is a lower bound for the line's length for each task. The highest such value is returned as Lower Bound 2.

These lower bounds tend to be stronger in different occasions: Lower Bound 1 provides a better approximation when precedence relations are weaker, but tends to be weak when the value of w_{max} is large. Lower Bound 2 tends to be better when precedence relations are stronger and more chains of tasks are present (OTTO *et al.*, 2013). The maximum of both bounds is employed for the computational experiments.

8.6 RESULTS

The developed heuristic procedure was applied to FMALB instances based on classical SALB instances (SCHOLL, 1999) available at www.assembly-line-balancing.de/salbp/benchmark-data-sets-1993/. Out of these classical SALB, only the subset tested by Kellegöz (2017) were used, so that a direct benchmark comparison is possible. Table 30 presents the number of tasks ($|T|$) and order strength (OS) of each precedence graph, which measures how restrictive precedence relations are (OTTO *et al.*, 2013), with 0 meaning no precedence relations and 1 meaning all tasks belong to a single chain of precedence relations. Table 30 also presents instance parameters: maximum number of parallel workers (w_{max}), cycle time (CT), and total number of workers for each cycle time ($|W|$). While the latter was set in accordance to the answers reported by Kellegöz (2017), they match the SALB optimal solution for their respective cycle time for all instances except for Barthol2 (CT=106) and Scholl (all CT values), for which it is one worker more. In total, the case study includes 88 instances.

Tables 31, 32, and 33 summarize the obtained results for small, medium, and large instances, respectively. The line lengths are converted in normalized (decimal) number of stations for each instance, by dividing them by the cycle time, then reported under the "Proposed" columns. Tests were conducted on a Core-i7-8700 (3.2GHz CPU with 32GB RAM) with a time limit of one hour per instance and Gurobi 8.0 was employed as the MILP solver.

Table 30 – Instance Information

Precedence Diagram			Instance Parameters		
Instance	$\|T\|$	OS	w_{max}	CT	$\|W\|$
Mitchell	21	0.71	2	14, 26, 39	8, 5, 3
Heskiaoff	28	0.22	2, 4	138, 256, 342	8, 4, 3
Sawyer	30	0.58	2, 4	25, 27, 54, 75	14, 13, 7, 5
Kilbridge	45	0.45	2, 4, 6	57, 110, 184	10, 6, 3
Tonge	70	0.59	2, 4, 6	176, 410, 527	21, 9, 7
Arcus1	83	0.59	2, 4, 6	5048, 7571, 10816	16, 11, 8
Mukherje	94	0.44	2, 4, 6	176, 248, 351	25, 18, 13
Arcus2	111	0.40	2, 4, 6	5755, 10743, 17067	27, 15, 9
Barthol2	148	0.26	2, 4, 6	84, 106, 170	51, 41, 25
Barthold	148	0.26	2, 4, 6	403, 513, 805	14, 11, 7
Scholl	297	0.58	2, 4, 6	1394, 1883, 1394	51, 38, 26

Answers reported by Kellegöz (2017) are presented on the Benchmark columns (Bmk.) and those obtained by the proposed heuristic procedure are reported under the "Proposed" columns. The "Reduction" columns present how shorter the "Proposed" ($Prop$) is when compared to the benchmark (Bmk): $(Bmk - Prop)/Bmk$.

Table 31 – Result Summary for small size instances

Instance Information				Solutions			Lower Bound		Solution Time
Name	CT	w_{max}	$\|W\|$	Proposed	Bmk.	Reduction	LB	Gap	(seconds)
Mitchell	14	2	8	5.43	7	22.45%	5.29	2.63%	0.2
	26	2	5	2.85	4	28.85%	2.85	0%	0.6
	39	2	3	2	2	0%	1.9	5.13%	0.2
Heskiaoff	138	2	8	4.01	5	19.71%	3.97	1.08%	47.1
	138	4	8	3.38	4	15.4%	3.38	0%	0.1
	256	2	4	2.28	3	23.96%	2.14	6.16%	0.6
	256	4	4	1.82	3	39.19%	1.82	0%	0.1
	342	2	3	1.99	2	0.29%	1.99	0%	0.1
	342	3	3	1.37	2	31.73%	1.37	0%	1.0
Sawyer	25	2	14	6.72	8	16%	6.48	3.57%	73.9
	25	4	14	5.88	8	26.5%	5.88	0%	0.7
	27	2	13	6.41	8	19.91%	6	6.36%	1777
	27	4	13	5.44	8	31.94%	5.44	0%	24.7
	54	2	7	3.09	4	22.69%	3	2.99%	39.9
	54	4	7	2.72	4	31.94%	2.72	0%	0.2
	75	2	5	2.32	3	22.67%	2.32	0%	1936
	75	4	5	1.96	3	34.67%	1.96	0%	18.1
Kilbridge	57	2	10	5.33	6	11.11%	5.14	3.62%	958
	57	4	10	3.6	5	28.07%	3.6	0%	1002
	57	6	10	3.6	5	28.07%	3.6	0%	0.6
	110	2	6	2.66	3	11.21%	2.62	1.71%	168
	110	4	6	1.86	3	37.88%	1.82	2.44%	27.2
	110	6	6	1.86	3	37.88%	1.82	2.44%	0.1
	184	2	3	2	2	0%	2	0%	0.1
	184	3	3	1.26	2	37.23%	1.17	6.93%	533
Averages	-	-	-	-	-	23.2%	-	1.8%	264.4

For each instance, the best out of the two lower bounds described in Section 8.5 is reported under the "LB" columns, and the integrality gap of the best solution is presented under the "Gap" columns. It is computed as the percentage difference between the best upper bound (UB) and the lower bound (LB): $(UB - LB)/UB$. The time required to reach the best incumbent is reported by the rightmost columns. As expected, this time is greater, in average, for larger instances (Table 32 and 33 versus Table 31). This reflects the algorithm's capacity to improve its solution with more computational time, suggesting early convergence is not a problem.

Out of 88 instances, the proposed heuristic procedure obtained strictly better answers than the benchmark in 81. In average, the line length of solutions obtained by the proposed procedure is 18% smaller (weighted average of 23.2%, 14.7%, and 16.3%) than those associated to the benchmark solutions. Furthermore, 17 optimal solutions were identified in regard to line length. In three instances (Mitchell-39-2-3, Kilbridge-184-2-3 in Table 31, and Barthold-805-2-7 in Table 33) both methods reached the optimal solution. In one instance (Barthold-403-2-14 in Table 33) the proposed heuristic procedure was outperformed by the benchmark. Lastly, in three

Table 32 – Result Summary for medium size instances

Instance Information				Solutions			Lower Bound		Solution Time
Name	CT	w_{max}	$\|W\|$	Proposed	Bmk.	Reduction	LB	Gap	(seconds)
Tonge	176	2	21	10.64	12	11.32%	9.97	6.3%	39.7
	176	4	21	7.34	10	26.65%	6.72	8.37%	2275
	176	6	21	6.98	10	30.23%	6.72	3.66%	2806
	410	2	9	4.67	5	6.63%	4.56	2.3%	3166
	410	4	9	3.42	4	14.39%	2.89	15.74%	703
	410	6	9	3.68	4	8.11%	2.89	21.5%	441
	527	2	7	3.68	4	8.06%	3.66	0.46%	162
	527	4	7	2.83	3	5.57%	2.46	13.19%	493
	527	6	7	2.82	3	6.01%	2.46	12.79%	708
Arcus1	5048	2	16	8.59	10	14.13%	8.49	1.08%	1014
	5048	4	16	8.01	10	19.88%	8.01	0%	0.5
	5048	6	16	8.01	10	19.88%	8.01	0%	0.5
	7571	2	11	5.81	6	3.17%	5.66	2.52%	41.6
	7571	4	11	5.43	6	9.42%	5.34	1.7%	1.0
	7571	6	11	5.43	6	9.42%	5.34	1.7%	1.1
	10816	2	8	4.15	5	16.92%	3.96	4.56%	619
	10816	4	8	3.85	5	23.08%	3.74	2.77%	4.3
	10816	6	8	3.85	5	23.08%	3.74	2.77%	4.9
Mukherje	176	2	25	14.11	16	11.83%	13.36	5.28%	4.1
	176	4	25	10.12	13	22.16%	9.07	10.33%	1157
	176	6	25	9.4	13	27.71%	8.28	11.91%	2.8
	248	2	18	9.75	10	2.46%	9.48	2.77%	36.3
	248	4	18	7.31	8	8.62%	6.44	11.91%	204
	248	6	18	6.62	7	5.47%	5.88	11.21%	46.4
	351	2	13	7.04	8	12.04%	6.7	4.78%	2246
	351	4	13	5.3	7	24.22%	4.55	14.23%	218
	351	6	13	5.01	6	16.48%	4.15	17.17%	2072
Averages	-	-	-	-	-	14.7%	-	6.9%	840.2

other instances (Marked with "*" in Table 33) the heuristic procedure failed to reach a feasible solution within the time limit. This is due to the fact that the SALBP instance (Barthol2, CT=84, $|W|$=51) associated with it is particularly challenging (SCHOLL; KLEIN, 1997). Consequently, the Master Problem of the heuristic procedure have not reach the target cycle time[1].

The best solutions found by the procedure for all instances, are made available by Lopes *et al.* (2019)'s Supplementary Material. Solutions are reported in terms of assignment and scheduling decisions for each task and can be easily checked for feasibility or converted in a Gantt chart.

[1] By providing a feasible SALB solution as a MIP start to the heuristic procedure, solutions with normalized line length of 25.8, 16.1, and 13.9 were obtained for w_{max} values of 2, 4, and 6, respectively. Such solutions are not reported by Table 33 as they required this specific warm-start, which is not a part of the procedure described in Section 8.4. These solutions are nevertheless available in Lopes *et al.* (2019)'s Supplementary Material.

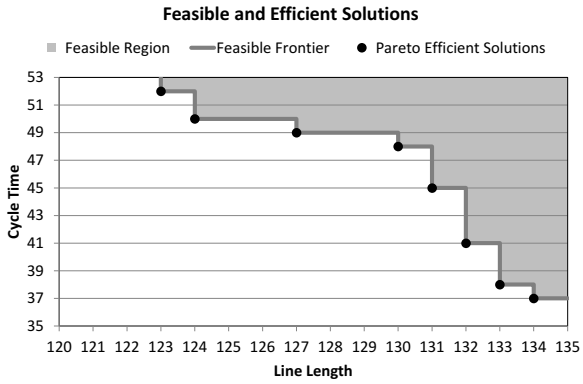
Table 33 – Result Summary for large size instances

Instance Information				Solutions			Lower Bound		Solution Time
Name	CT	w_{max}	$\|W\|$	Proposed	Bmk.	Reduction	LB	Gap	Seconds
	5755	2	27	14.61	16	8.71%	13.61	6.84%	237
	5755	4	27	11.68	13	10.19%	10.62	9.04%	987
	5755	6	27	11.16	13	14.16%	10.62	4.84%	1827
	10743	2	15	7.72	9	14.24%	7.29	5.56%	3590
Arcus2	10743	4	15	6.17	8	22.83%	5.69	7.86%	58
	10743	6	15	5.97	8	25.37%	5.69	4.72%	1081
	17067	2	9	4.86	5	2.71%	4.81	1.08%	1212
	17067	4	9	3.93	5	21.35%	3.58	8.94%	1625
	17067	6	9	3.91	5	21.74%	3.58	8.5%	1163
	84	2	51	*	26	*	25.4	1.61%	*
	84	4	51	*	16	*	12.81	19.94%	*
	84	6	51	*	14	*	9.89	28.61%	*
	106	2	41	20.51	21	2.34%	19.97	2.62%	2369
Barthol2	106	4	41	11.7	13	10.01%	9.99	14.6%	3283
	106	6	41	9.69	12	19.26%	7.84	19.08%	2878
	170	2	25	12.94	13	0.5%	12.91	0.23%	510
	170	4	25	7.69	8	3.82%	6.91	10.24%	2513
	170	6	25	5.93	8	25.88%	4.91	17.26%	3264
	403	2	14	7.12	7	-1.74%	6.99	0.14%	979
	403	4	14	4.27	5	14.64%	3.99	6.51%	8.4
	403	6	14	3.39	5	32.16%	2.99	11.85%	2872
	513	2	11	5.98	6	0.29%	5.98	0%	297
Barthold	513	4	11	3.23	4	19.35%	2.99	7.19%	1251
	513	6	11	2.84	4	29.09%	2.2	22.27%	2062
	805	2	7	4	4	0.03%	4	0%	3.5
	805	4	7	2	3	33.33%	2	0%	871
	805	6	7	2	3	33.37%	2	0%	40
	1394	2	51	27.97	30	6.77%	25.28	9.62%	1896
	1394	4	51	20.44	25	18.24%	16.54	19.08%	408
	1394	6	51	19.26	26	25.92%	16.25	15.64%	3126
	1883	2	38	20.18	23	12.25%	18.71	7.28%	453
Scholl	1883	4	38	14.75	20	26.26%	12.25	16.97%	143
	1883	6	38	13.79	20	31.07%	12.03	12.74%	143
	2787	2	26	13.52	18	24.91%	12.64	6.46%	365
	2787	4	26	9.76	15	34.91%	8.27	15.26%	2158
	2787	6	26	9.27	16	42.08%	8.13	12.3%	3303
Averages	-	-	-	-	-	16.3%	-	9.3%	1423.5

8.6.1 Line Length versus Cycle Time

The relationship between cycle time and optimal line length might be of particular interest when a range of possible cycle time values are considered rather than one specifically. In order to investigate the way in which these factors correlate, the illustrative problem presented by Figures 26, and 27 (with $|W| = 6$, and $w_{max} = 3$) was solved to optimality (using the model presented in Section 8.3) for multiple values of cycle time (including fractional ones, with increments of 0.5 T.U.). The feasible region, as well as the Pareto-efficient solutions for that instance are presented by Figure 28.

Figure 28 – Illustrative problem's feasible solutions: cycle time and line length



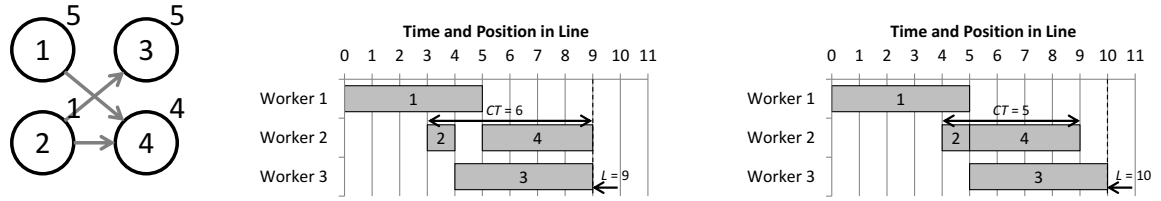
Source: Lopes *et al.* (2020)

Notice that the Pareto frontier is discontinuous even though the feasible region is not. Furthermore, all Pareto efficient solutions have integer values of both line length and cycle time. This relationship between L and CT is further explored by a small example presented by Figure 29.

Figure 29(a) presents the instance data, Figures 29(b) and 29(c) depict two solutions with different values of line length and cycle time for three workers ($|W| = 3$, $w_{max} = 3$). Notice that the task assignments (tied to the $x_{t,w}$ binary variable set) and ordering (tied to the f_{t_2,t_1} binary variables) are the same for both solutions. By fixing these binary decisions, the model presented in Section 8.3 defines a residual linear problem with continuous scheduling variables for tasks ($Tstart_t$), workers ($Wstart_w$ and $Wend_w$), as well as cycle time (CT) and line length (L). This means that if two solutions are feasible, there is a continuum of intermediary solutions between them (HILLIER; LIEBERMAN, 2015). Consider line length and cycle time (L, CT): Figure 29(b) presents a solution with (9, 6) and Figure 29(c) presents a solution with (10, 5). This means that any linear interpolation such as (9.5, 5.5) is also a feasible solution of the problem.

Figure 29 – Illustrative example with continuous Pareto frontier

(a) Instance data **(b) Solution with $CT = 6$ and $L = 9$** **(c) Solution with $CT = 5$ and $L = 10$**



Source: Lopes *et al.* (2020)

This leads to the conclusion that, while the Pareto frontier can be discontinuous (Figure 28), it can also have continuous segments depending on the instance parameters, in particular

task durations and precedence relations.

8.7 CONCLUSIONS

In this study, multi-manned assembly lines are addressed. These lines are usually employed when products are large and, therefore, allow multiple operators to perform tasks simultaneously on them. Previous literature defined the problem in terms of discrete workstations with fixed frontiers to which multiple operators can be assigned. However, when lines are continuous (in contrast to unpaced synchronous and asynchronous lines), there is no need to impose such constraint. This chapter presents a formulation that allows the definition of flexible workstation boundaries, which allow much more freedom in regard to the combined balancing-scheduling problem that defines such multi-manned lines. While Chapter 3 shows that line control is a relevant performance factor for mixed-model balancing problem variants, this chapter shows that it is also relevant for single-product balancing problem variants.

A new Mixed-Integer Linear Programming (MILP) model is presented. It incorporates simultaneously balancing and scheduling decisions such that integer solutions will: respect the cycle time and precedence relations, allow at most a given number of workers to perform tasks in the workpiece simultaneously, and allow each worker to perform one task at a time. The model incorporates multiple Big-M constraints and can be intractable for medium and large instances. Hence, a MILP-based heuristic procedure is presented. Furthermore, two algorithmic lower bounds are introduced and explained.

Tests were conducted on a 88-instance dataset. The proposed heuristic procedure outperformed the previous multi-manned benchmark in 81 instances. Incumbent solutions have line length 18% smaller than the benchmark in average, and up to 42% in the best case. This highlights the contribution of allowing flexible workstation frontiers. Algorithmic lower bounds allowed to demonstrate that 17 out of the solutions found by the proposed procedure are optimal.

Further analysis show that a dispute exists between efficient values of cycle time and line length, indicating it as a direction for further works on multi-objective optimization. Examples show that the Pareto frontier in terms of cycle time and line length can be either continuous or discontinuous, depending on the instance's parameters.

9 A CONTRIBUTION ON FRACTIONAL TASK ALLOCATIONS

This chapter is based on an article published in the International Journal of Production Research (LOPES *et al.*, 2021). It contributes to a previous body of literature (dynamic balancing, work-sharing) that had already considered fractional task allocations as a means to potential performance gains. By applying cyclical scheduling analyses to two problem variants, this chapter describes how those performance gains are tied to internal storage requirements. Mathematical formulations that allow for their minimization describe these costs for each line control type: for asynchronous lines, buffers allow lower cycle times (similarly to what is discussed in Chapter 5); meanwhile, for paced lines a trade-off between line length and cycle time is observed (similarly to what was found in Chapter 8).

9.1 INTRODUCTION

Assembly lines are product-oriented production layouts commonly employed in various industrial settings. The most classical optimization problem associated to them is the assembly line balancing problem (SCHOLL, 1999), which consists in assigning task to stations. The most basic of such problems is the Simple Assembly Line Balancing Problem (SALBP), which has multiple simplifying assumptions (BAYBARS, 1986). These have been questioned and relaxed by many authors, defining many general problem variants (BOYSEN *et al.*, 2008). Out of SALBP's basic hypotheses, fewer works challenge the binary nature of the task-station assignments.

To the best of the authors' knowledge, two overlapping bodies of literature have considered challenging this hypothesis: dynamic balancing and work-sharing (CHEN; ASKIN, 2006; ANUAR; BUKCHIN, 2006). These literature bodies allow some tasks to be performed sometimes at one station and sometimes at the next or previous one. This means that for that task, its allocation decision variable is a fraction that states how often the task is performed on each station. Naturally, some restrictions apply: e.g. each station pair can only share one task, and each task can only be shared by two stations. Nonetheless, this represents an additional balancing flexibility, and the common conclusion drawn from these literature bodies is this added flexibility allows higher throughputs (or equivalently, lower cycle times). The concept of fractional task allocations is also found in lines with station parallelism: in them, tasks are performed 50% of the time in each of the parallel stations (assuming there are two of them). This body of literature

also concludes that higher throughput is achievable under the greater assignment flexibilities. However, parallel stations require significant layout changes and increased worker training costs due to the increase in number of tasks performed. A current limitation of the work-sharing and dynamic balancing literatures is not adequately describing the costs and trade-offs required by the fractional allocation of tasks. Consequently, task-sharing has not yet been formalized in terms of a proper variant of the assembly line balancing problem: The most recent reviews on the topic (BOYSEN *et al.*, 2008; BATTAÏA; DOLGUI, 2013) do not address this potential flexibility.

This chapter formally defines the Fractional Allocation Assembly Line Balancing Problem (FA-ALBP) as a direct variant from SALBP. Its main contribution to the related literature is the description of how internal storage costs are tied to these fractional allocations for each line type. This formalization allows a better understanding of the trade-offs tied to this performance enhancing opportunity.

This Chapter is structured as follows: Section 9.2 presents the relevant literature and contextualizes the Chapter's contribution. Section 9.3 presents the formal notation and general definitions of the considered problem. Section 9.4 presents the cycle time minimization (throughput maximization) aspect of the problem, while Section 9.5 presents the internal storage minimization one. Experiments are presented in Section 9.6 and the key discussions and contributions are presented in Section 9.7.

9.2 LITERATURE REVIEW

Assembly line balancing problems are generally NP-Hard optimization problems (ÁLVAREZ-MIRANDA; PEREIRA, 2019). Its most basic version, the Simple Assembly Line Balancing Problem (SALBP) has several simplifying hypotheses that reflect the earliest studies on production lines. In the definition presented by Baybars (1986), the hypothesis that "a task cannot be split among two or more stations" is second only to "all input parameters are known with certainty". Multiple problem variants are defined by deviating from the other hypothesis, but are beyond the scope of this review. Readers can refer to Boysen *et al.* (2008) and Battaïa and Dolgui (2013) for classifications of such variants.

Chen and Askin (2006) define flexibility on task-station assignments for specific tasks as "dynamic line balancing": some tasks are fixed to stations while others are allowed to be shared between two adjacent stations. In their experiments, this flexibility leads to higher productivity. However, other authors define dynamic balancing differently: Shtub (1993) discusses it as part of

the just-in-time philosophy of Japanese manufacturers (SCHONBERGER, 1982), and Ahn and Righter (2006) present it in terms of a flexible assignment of cross-trained workers to stations. These works often employ simulations to analyze policies or combine them to optimization approaches (DAGKAKIS *et al.*, 2019). For instance, Pasupa and Suzuki (2019) recently presented a simulation-based case study for dynamic line balancing under heterogeneous workers. In this literature, a common theme is the achievement of a higher throughput.

Anuar and Bukchin (2006) share Chen and Askin (2006)'s definition of dynamic line balancing, but later works (BUKCHIN; SOFER, 2011; BUKCHIN; WEXLER, 2015) simply refer to it as "work-sharing". Anuar and Bukchin (2006) consider that the assembly sequence is fully given, and that each station can share a task with both its upstream and downstream neighbor simultaneously. Bukchin and Sofer (2011) go further by allowing more general task sequences. Both works, however, limit their analyses to asynchronous unpaced lines and the costs of task-sharing are considered given parameters which do "not depend on the sharing proportion of the task" - a hypothesis that is questioned by the present chapter. Furthermore, the bi-directional work-sharing complicates analytical descriptions of buffer requirements, requiring simulations to address that aspect of the problem (ANUAR; BUKCHIN, 2006). Nonetheless, work-sharing tends to offer better efficiency, both as lower cycle times (BUKCHIN; SOFER, 2011) and as lower makespan (GULTEKIN, 2012; BUKCHIN; WEXLER, 2015).

Other examples of work-sharing encompass performing tasks in different stations for different product models (YANG *et al.*, 2014) and splitting some tasks via alterations in the precedence diagram (GRZECHCA; FOULDS, 2015). Hence, task splitting/sharing, or partial/fractional task-station allocations are research topics that were addressed by some authors in multiple manners. In a recent paper, Jeong and Jeon (2020) further investigated the conditions for balanceability (reaching the minimum possible cycle time) tied to work-sharing for both straight and U-shapped assembly lines. However, the authors consider unlimited buffers between stations and do not investigate the impact of work-sharing on internal storage requirements.

However, a limitation of the aforementioned literature is that line control (BOYSEN *et al.*, 2007) is either not discussed or presumed to be unpaced asynchronous, a context in which buffers are naturally used to compensate performance oscillations and to allow higher throughputs, as shown in Chapters 2-6. Chapter 3 shows that a line's control can influence its performance. Paced lines can also employ line length to compensate for fluctuations in processing times (BOYSEN *et al.*, 2008), a feature that allows lower cycle time in specific contexts, as

shown in Chapter 8. Work-sharing or fractional task allocations naturally cause station-wise performance oscillations, which must be compensated by internal storage. However, the current literature on the subject lacks an analytical description on how internal storage is tied to fractional allocations.

Lastly, it is important to compare the flexibilities of task-sharing to those of parallel stations. The later are common to various practical contexts and have been extensively studied (SAWIK, 2012; ÖZTÜRK *et al.*, 2015; MICHELS *et al.*, 2018). In some contexts, parallelism is employed if a task exceeds the cycle time in duration (BECKER; SCHOLL, 2009) or as a manner to add additional scheduling flexibility to mixed-model lines, as shown by Chapter 4. Parallel stations can be seen as a form of fractional task allocation, i.e. tasks are 50% (assuming 2 simple parallel stations) assigned to each workstations. This means that they tend to have most of the allocation flexibilities of task-sharing. However, not all contexts are adequate for parallel stations: loss of worker specialization, layout complications, and increases in tool duplication costs tend to be larger for parallel stations than to fractional task allocations. In fact works that consider parallel stations often seek to limit or minimize their number (ÁLVAREZ-MIRANDA *et al.*, 2020) in order to minimize these costs.

This chapter defines a deterministic fractional allocation assembly line balancing problem in which each station is only allowed to share one task. By doing so, it is possible to define analytically the required internal storage costs for both paced and unpaced lines, without any simulation requirement (ANUAR; BUKCHIN, 2006).

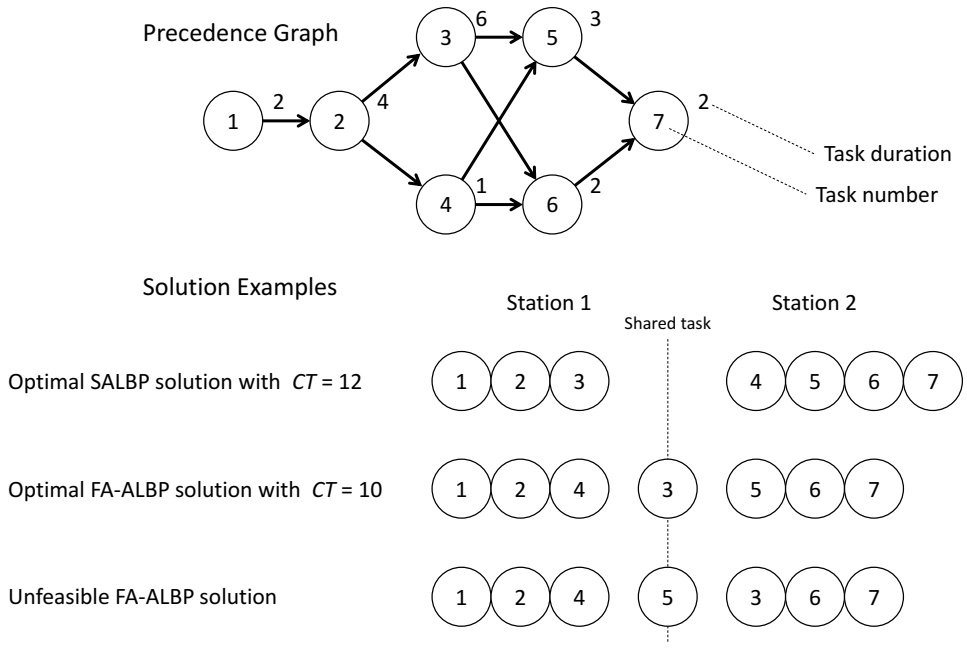
9.3 PROBLEM DESCRIPTION

The Fractional Allocation Assembly Line Balancing Problem (FA-ALBP) consists in assigning a set of tasks T (each with a given duration D_t) to a set of stations S, while respecting the precedence relations P_t for each task. Two problem goals are considered: first, to minimize the line's cycle time, second, to minimize the line's space requirements. In this chapter, these objectives are treated as hierarchical in nature, meaning that minimizing cycle time is considered strictly more important than minimizing space requirements. This leads to a two-stage approach: in the first stage, the line's cycle time is minimized; in the second, space requirements are minimized, while imposing as a constraint cycle time performance obtained by the first stage. In this chapter, the number of stations is considered given, hence line efficiency is measured in terms of the cycle time CT. Space costs are measured in terms of either line length or buffer

requirements, depending on the line control. While tasks are indivisible for each individual product, tasks can be fractionally assigned to stations in the following sense:

1. The fractional assignment value states the percentage of products for which the task will be performed at each station;
2. A task can only be split/shared by two adjacent stations;
3. Precedence relations must be respected for all products that flow through the line;
4. Only one task can be shared between two stations;
5. Each station can share tasks with either their upstream or downstream neighbor, but not both at once.
6. Both stations that share the task can perform it, and the task's duration is constant, regardless of the station that performs it.

Figure 30 – Precedence diagram example and solution examples



Source: Lopes *et al.* (2021)

Figure 30 presents a precedence diagram and illustrates the differences between SALBP and FA-ALBP, as well as some requirements of the latter. These definitions reflect a series of feasibility and practical considerations, such as producing a solution that can be straightforwardly

implemented. For instance, allowing stations to share tasks with both the upstream and downstream neighbor makes the problem easier to solve (Polynomial time for cycle time minimization) and allows maximum cycle time reduction (BUKCHIN; SOFER, 2011). However, this can also produce confusing scheduling solutions that would be very impractical to implement. Hence, for each station, fractional task allocations are limited to either the previous or the next one. This means that the line can be seen as a set of regular stations and task-sharing station blocks. The former are typical of traditional assembly lines, and always processes the current product within the cycle time. The latter consists of two adjacent stations that share a task, meaning the processing time of each station oscillates between a value higher than CT and another one lower than CT. In average, however, these blocks (and its stations) must process one product every cycle time, meaning its oscillations must be compensated by internal storage (buffers or line length). Cyclical scheduling can be used to validate the block's capacity to reach the required average cycle time with the available internal storage.

9.4 CYCLE TIME MINIMIZATION

This section presents the proposed mathematical model used to describe the studied problem. In this section, the model describes the cycle time minimization step. In order to represent this task-station allocation problem, three sets of decision variables are employed: $x_{t,s}$, $y_{t,s}$, and $z_{t,s}$. Each variable in the first one ($x_{t,s}$) is set to 1 when task t is fully assigned to station s, meaning it is performed in it for all products. The second one ($y_{t,s}$) is set to 1 when task t is shared between stations s and $s+1$. The third one ($z_{t,s}$) states the percentage of products for which task t is performed at station s, e.g. $z_{1,2} = 0.5$ means that for 50% of products, task 1 is performed at station 2. The proposed model for time efficiency maximization is presented by Expressions (117) to (128).

The problem's goal (Expression (117)) is to minimize the cycle time, as the number of stations is given by hypothesis. Equation (118) states that every task must be either always performed at a station or shared between a station pair. Constraint (119) states that the sum of fractional allocations for every task must add up to 100%, i.e. every task is performed for every product. Constraint (120) states the direct precedence relations requirements: tasks must be assigned or shared at a station pair prior to their direct successors. Inequality (121) also states precedence constraints in terms of the fractional decision variables. Constraint (122) states the

cycle time constraint: every station's average performance is a potential line bottleneck.

$$\text{Minimize} \quad CT \tag{117}$$

subject to:

$$\sum_{s \in S} (x_{t,s} + y_{t,s}) = 1 \qquad \forall\, t \in T \tag{118}$$

$$\sum_{s \in S} z_{t,s} = 1 \qquad \forall\, t \in T \tag{119}$$

$$\sum_{s \in S} s \cdot (x_{t_1,s} + y_{t_1,s}) \leq \sum_{s \in S} s \cdot (x_{t_2,s} + y_{t_2,s}) \qquad \forall\, (t_1,t_2) \in R \tag{120}$$

$$\sum_{s \in S} s \cdot z_{t_1,s} \leq \sum_{s \in S} s \cdot z_{t_2,s} \qquad \forall\, (t_1,t_2) \in R \tag{121}$$

$$\sum_{t \in T} D_t \cdot z_{t,s} \leq CT \qquad \forall\, s \in S \tag{122}$$

$$\sum_{t \in T} (y_{t,s-1} + y_{t,s}) \leq 1 \qquad \forall\, s \in S :\ s > 1 \tag{123}$$

$$z_{t,s} \geq x_{t,s} \qquad \forall\, t \in T, s \in S \tag{124}$$

$$z_{t,s} \leq x_{t,s} + y_{t,s} + y_{t,s-1} \qquad \forall\, t \in T, s \in S : s > 1 \tag{125}$$

$$z_{t,1} \leq x_{t,1} + y_{t,1} \qquad \forall\, t \in T \tag{126}$$

$$x_{t,s},\, y_{t,s} \in \{0,1\} \qquad \forall\, t \in T, s \in S \tag{127}$$

$$K \cdot z_{t,s} \in \mathbb{Z}^+ \qquad \forall\, t \in T, s \in S \tag{128}$$

Inequality (123) states that a station cannot simultaneously share tasks with both the previous and following neighbor, and that only one task can be shared per station. Constraints (124) to (126) logically tie the variable sets x and y to z. Expression (127) presents the binary requirements for variable sets x and y. Last, Expression (128) states an optional special integer requirement for the otherwise continuous variable set z: by adding it, z is allowed to be a fraction of a given integer denominator K. For instance, if $K = 4$, the fractional allocations variables $z_{t,s}$ can only assume the values 0, 0.25, 0.5, 0.75, and 1. Section 9.5 discusses the relevance of this factor for each assembly line type. Regarding implementation by general solvers, this constraint can be added using auxiliary integer variables. Furthermore, upon implementation, pre-processing can eliminate some variables, by fixing their values as zero: a valid upper bound on CT allows discarding some of the $x_{t,s}$ variables using natural adaptations of the earliest and latest stations concepts (GÖKÇEN; EREL, 1997). Similarly, the $y_{t,s}$ variables at the last station $s = |S|$ are always zero because there is no station $s + 1$ in this case. If, for any practical reason

a specific task t cannot be shared, all the associated $y_{t,s}$ variables should also be set to zero as well.

9.4.1 Problem's Complexity

The studied problem's complexity depends on the number of stations. For a fixed number of stations of at least 3, the problem is at least NP-complete in the weak sense. This is proven based on the fact that the bi-partition problem (HAYES, 2002) reduces to FA-ALBP with cycle time restriction. Given integers $a_1, \dots, a_n, b$ such that $\sum_j a_j = 2 \cdot b$, the following instance of FA-ALBP can be constructed with n tasks and 3 stations:

$$
\begin{aligned}
& CT = b \\
& D_1 = b + a_1 \qquad\qquad D_t = a_t \qquad \forall\, t \in \{2, \dots, n\} \\
& R = \{(1,t) : t \in \{2, \dots, n\}\}
\end{aligned}
$$

The first task precedes all others, therefore, it must be assigned to the first station. Because its size exceeds cycle time, it must also be shared between the first and second stations. This consumes all task-sharing possibilities, since the second station cannot simultaneously share tasks with both neighbors. The first task has duration $b + a_1$, which must be split as follows: b time units are performed at the first station, and a_1 at the second. The remaining tasks with durations $a_2, \dots, a_n$ must be partitioned between the second and third stations. Therefore, the FA-ALBP instance is feasible if and only if the bi-partition has a solution.

If the number of stations is part of the instance, FA-ALBP is NP-complete in the strong sense. This is proven based on the fact that the 3-Partition (GAREY; JOHNSON, 1979) problem reduces to FA-ALBP. Given integers $a_1, \dots, a_{3 \cdot n}, b$ such that $b/4 < a_j < b/2$ and $\sum_j a_j = n \cdot b$, the following instance of FA-ALBP can be constructed with $4 \cdot n$ tasks and $2 \cdot n$ stations:

$$
\begin{aligned}
& CT = 2 \cdot b \\
& D_t = a_t && \forall\, t \in \{1, \dots, 3 \cdot n\} \\
& D_t = 3 \cdot b && \forall\, t \in \{3 \cdot n + 1, \dots, 4 \cdot n\} \\
& R = \emptyset
\end{aligned}
$$

In order for this instance to be feasible, the n large tasks ($D_t = 3 \cdot b$) must all be shared: their duration exceeds cycle time. There are $2 \cdot n$ stations, each can only share one task with

one of its neighbors (either previous or next station). This means only n tasks can be shared, therefore, only the n large tasks are shared. This defines n blocks of two stations with b time available. The instance is feasible if and only if the first $3 \cdot n$ tasks (integers a_j) can be partitioned in blocks of sum b. That is, if and only if the 3-Partition has a solution.

9.5 INTERNAL STORAGE COST MINIMIZATION

The model presented in Section 9.4 allows to compute the best cycle time $\overline{CT}$ that can be obtained by adding the fractional task allocation flexibility. This value can differ from the optimal cycle time of the equivalent SALBP instance (CHEN; ASKIN, 2006; ANUAR; BUKCHIN, 2006). If that is the case, it is interesting to investigate the internal storage cost tied to this additional flexibility. Such storage is needed to compensate for production rate oscillations of stations in task-sharing blocks. This can be done by solving the balancing problem once again, with an additional constraint tied to cycle time performance (Expression (129)) and a new goal function that reflects the aforementioned internal storage costs.

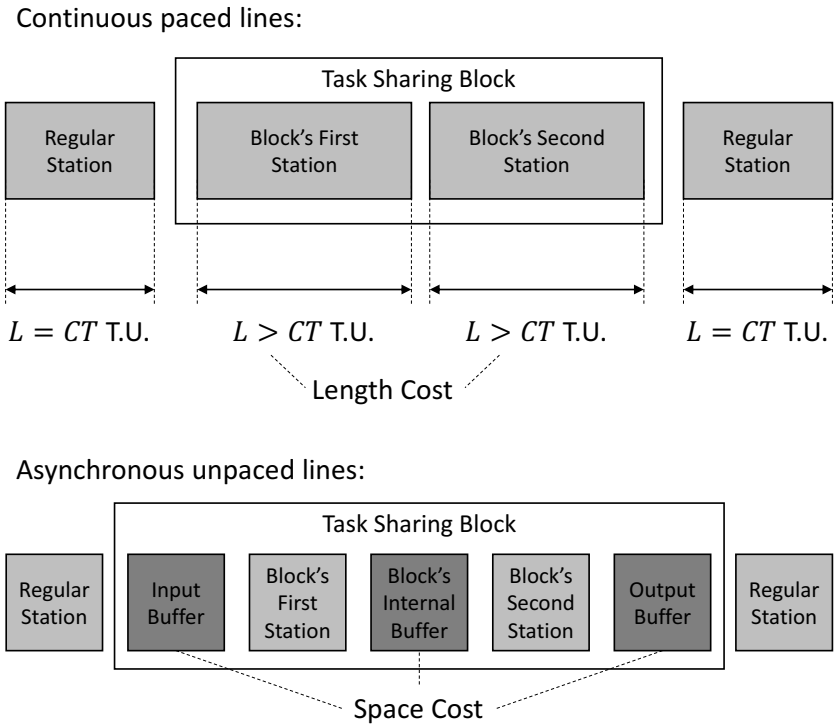
$$CT \leq \overline{CT} \tag{129}$$

These costs are dependent on the assembly line's control: for continuous paced lines they mean longer stations, hence a longer line length. These translate to greater internal storage because products enter the line at a constant rate, meaning longer lines (measured in time units) mean greater Work-in-Progress. Meanwhile, for unpaced asynchronous lines internal storage mean buffers between stations. These costs are conceptually illustrated in Figure 31. The following sections present goal functions that reflect worst-case bounds for these space costs for each of these line types.

9.5.1 Line Length Cost on Paced Lines

There are multiple variants of paced assembly lines from a scheduling perspective (BOYSEN *et al.*, 2009c). In this section, it is assumed that (*i*) products launch is rigidly disciplined (i.e. one product enters the line every CT time units), and (*ii*) that stations are rigidly separated (i.e. no station overlap is allowed). These hypotheses reflect "default" assumptions about paced assembly lines. Furthermore, considering them allows analyzing the impact of fractional allocations in isolation.

Figure 31 – Comparison of internal storage costs for different line controls (T.U. = time units)



Source: Lopes *et al.* (2021)

Under these considerations, it is possible to show that each task-sharing "block" can be studied independently: by placing boundaries between blocks and regular stations, it is clear that a product must cross each boundary every CT time units (hypothesis *(i)*). Furthermore, because station boundaries are rigid (hypothesis *(ii)*), the scheduling required within a block does not affect neighboring stations and blocks. This is illustrated by Figure 31. This implies that the total line length is the sum of lengths of all blocks and stations.

The length of regular stations are determined by their total processing time, which is given by the sum of tasks performed in them. This means it is only necessary to analyze blocks of adjacent stations that share a task. Let D state the duration of the task shared in the block, and B_1 and B_2 indicate the "base" time of each station in the block, i.e. the time required when they do not perform the shared task. In order to define a worst-case scenario for these blocks, it is assumed that the sum of the processing time of tasks assigned to these two stations is two cycle times ($B_1 + B_2 + D = 2 \cdot CT$). This means that there can be no idle time in these stations and that the percentage of time the shared task will be performed at each station is fixed when B_1, B_2 and D are given. Notice that Constraint (128) requires $z_{t,s}$ to be a fraction of denominator K, meaning that it is possible to define cyclical schedules based on K products. This can be done by defining residual problems of models developed to describe mixed-model paced lines

(DEFERSHA; MOHEBALIZADEHGASHTI, 2018). These residual problems will only have K binary variables, meaning they are tractable, and can be solved extensively for multiple values of z and D for the purposes of screening the required line length costs.

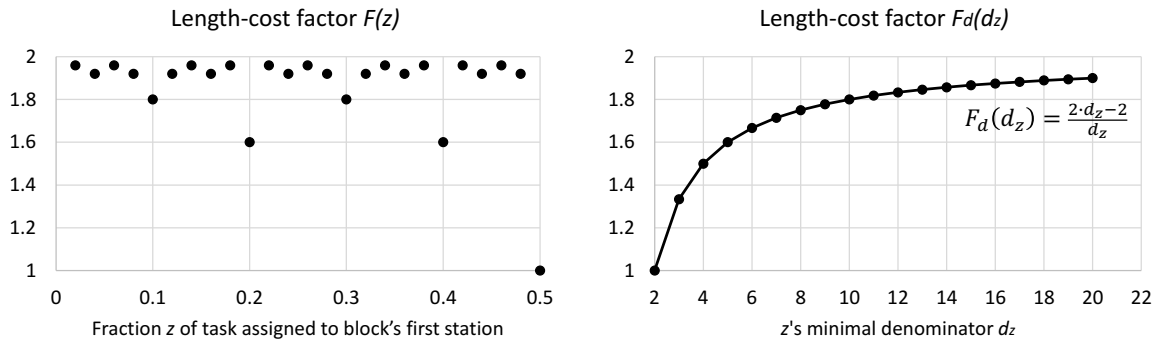
Preliminary tests indicated that the length of the block ($L = L_1 + L_2$) can be stated as twice the cycle time plus a cost that is proportional to the shared task's duration. This means that, by fixing the percentage of times (z) task are performed at each station, the line length increases linearly with the shared task's duration, as indicated by Equation (130).

$$L = 2 \cdot CT + F(z) \cdot D \tag{130}$$

The cost factor F is hence a function of the fraction z. Figure 32(a) presents the results of initial screenings with multiple values of z, which did not provide a clear growth or decline pattern. However, a more clear pattern, shown by Figure 32(b), is obtained by plotting F against the irreducible denominator d_z of the allocation fraction z. For instance, for $z = \frac{6}{8}$, $d_z = 4$. Notice that, because of Constraint (128), d_z is at most K.

Figure 32 – Length cost factor F behavior

(a) As a function of the fraction **(b) As a function of the denominator**

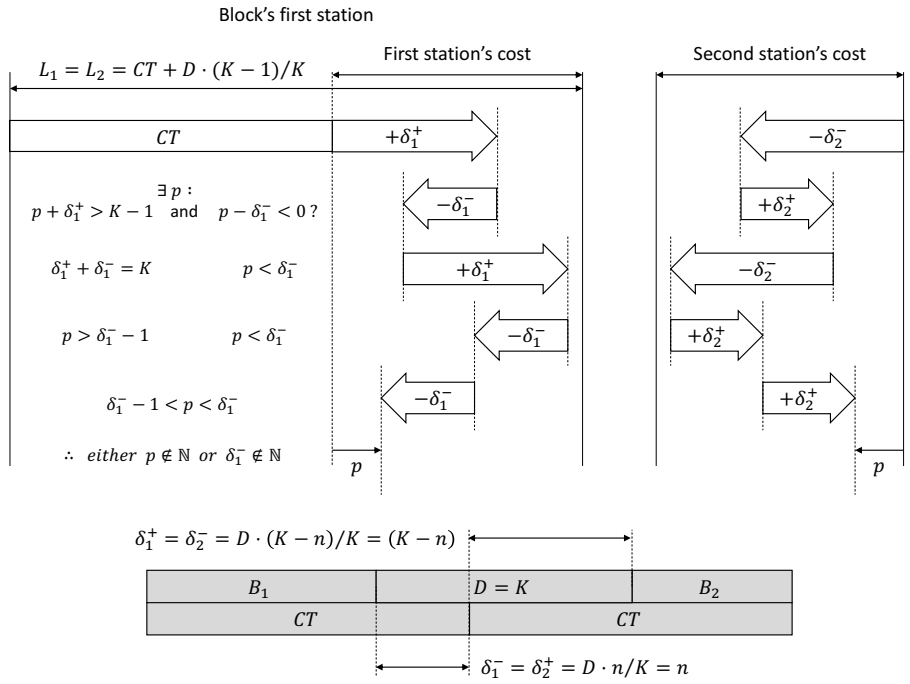


Source: Lopes *et al.* (2021)

In fact, the pattern observed in Figure 32(b) matches the expression $F_d = (2 \cdot d_z - 2)/d_z$. This can be proven by tracking the position p in which the worker finishes processing a product on the block's first station. Figure 33 serves as visual support for this proof. After each product, the paced nature of the line means the worker walks CT time units upstream to reach the next product. If by doing that he walks out of his station boundaries, then idle time occurs. However, this is a worst-case analysis in which no idle time is allowed: idle time would imply violating cycle time because $B_1 + B_2 + D = 2 \cdot CT$. This means that the finishing position can never be less than CT time units, which can serve as a zero reference point for this analysis. The finishing

position changes for each product by the difference (δ) between cycle time and the processing time of that product, as illustrated by Figure 33. Furthermore, positive differences for the block's first station translate into negative ones for the second station, and vice-versa. This imposes a symmetry in which the length of both station is the same and equal to the cycle time plus a cost tied to the variable part. As suggested by Figure 32(b), this cost should be $(K-1)/K$ for each station (or $(2 \cdot K - 2)/K$ for the block), where K is z's irreducible integer denominator. This is proven hereafter:

Figure 33 – Scheduling scheme for minimal block length



Source: Lopes *et al.* (2021)

First, consider the simpler case in which the shared task processing time D is equal to K and cycle time CT is an integer. In this case, if the task is assigned n out of K times to the block's first station, then n, K and $K-n$ are relative primes because K is an irreducible denominator by hypothesis. If the screening is correct, the cost should equal $K-1$ for the simpler case. This is true if and only if the optimal cyclical schedule of K products matches that value. By tracking the position in which the last product's processing was completed, the cyclical schedule defines a walk in integer values such that the difference between two consecutive positions is either $+\delta^+$ or $-\delta^-$. The highest position equals the station's length cost, which should be $K-1$, based on the preliminary screenings. The sufficiency of this value is proven first, by contradiction: suppose that there is a position p in the walk such that both $p + \delta^+ > K - 1$ and $p - \delta^- < 0$.

Because $\delta^+ + \delta^- = K$, those inequalities can be re-arranged to show that $\delta^- - 1 < p < \delta^-$, meaning either p or δ^- is not an integer (Figure 33). This is impossible because both differences are integer by hypothesis and the walk is a walk on integer numbers. Therefore, it is always possible to perform a step and stay within the range $[0,\, K-1]$. This means that $K-1$ should be sufficient to define a walk of arbitrary length. Its necessity is proven next: Consider that the starting position of the walk is 0. After K steps it is possible to return to the initial position. This is true because there are K positions in the range and after K steps, a repeated position must be reached (Pigeonhole principle). If that repeated position is 0, the result is true. Otherwise, the repetition occurred in less than K steps. This would imply that there is a smaller walk that defines a cyclical schedule with fewer than K products. However, this is impossible because the relative prime nature of the integers differences (δ^+ and δ^-) means their minimal common divisor is their product which can only be reached with n steps forward ($+\delta^+$) and $(K-n)$ steps backward ($-\delta^-$), which total K steps. This is relevant because a cyclical schedule means a walk in integer values such that the sum of movements forward equals the sum of movements backward. Therefore, $K-1$ is both sufficient and necessary for the cyclical schedule. This result can be easily generalized for other values of D and fractional values of cycle time. The former follows from observing that, for the same allocation fraction z, the δ steps are proportional to D (Figure 33). The latter reduces to a case with integer cycle time by multiplying all task durations by the cycle time's denominator.

This leads to two conclusions, first, that having lower denominators in the fractional task allocations lead to lower costs, which might motivate using a low value for K, such as 2 or 4. Second, the length cost of the fractional task allocations is bounded between one and two times the duration of shared tasks. This means that Expression (131) presents a worst-case MILP formulation for line length cost minimization step of FA-ALBP in the case of paced lines. If only fractions of up to a denominator K are used, then Expression (132) can be used instead.

$$\text{Minimize} \quad 2 \cdot \sum_{t \in T} D_t \cdot \sum_{s \in S} y_{t,s} \tag{131}$$

subject to: (118)-(129)

$$\text{Minimize} \quad F_d(K) \cdot \sum_{t \in T} D_t \cdot \sum_{s \in S} y_{t,s} \tag{132}$$

subject to: (118)-(129)

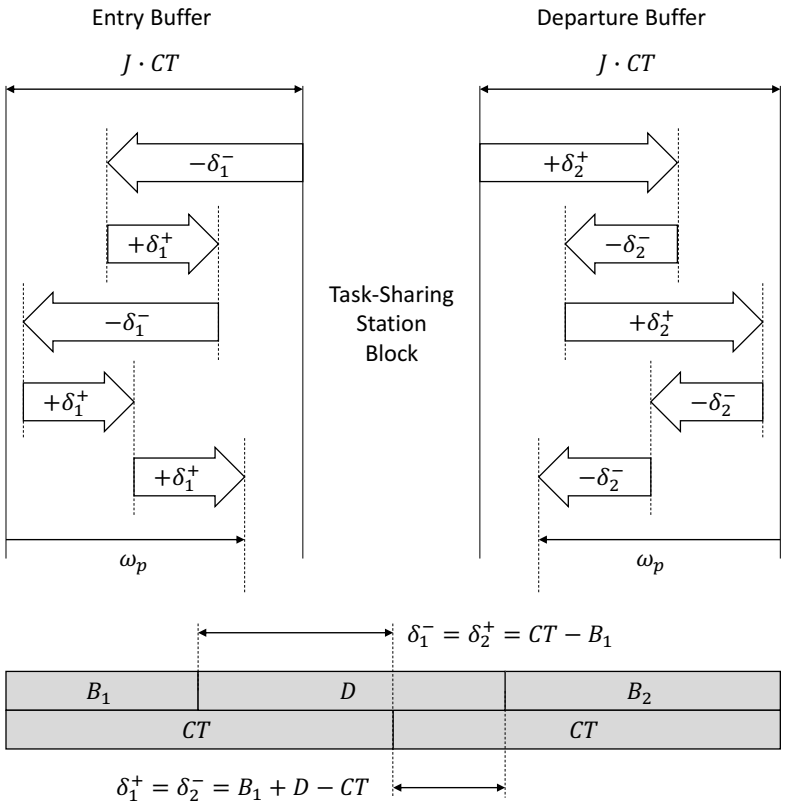
9.5.2 Buffer Cost on Asynchronous Lines

In unpaced asynchronous lines, internal storage (buffers) are commonly used to compensate for temporary disturbances in processing times such as mixed-model production (BOYSEN *et al.*, 2008). In this section, the worst-case for the minimum number of buffers required by a given fractional task allocation is analyzed. Once again, in order to reach a target cycle time CT, a cyclical schedule based on K products is used. Similar to paced lines, this analysis is based on defining the worst-case for each block of task-sharing stations. In order to do so, no intra-station idle time is allowed: Neighboring regular stations have processing times of CT time units, and the sum of processing times of station pairs in blocks is $2 \cdot CT$. This implies that the duration D of the shared task is necessarily smaller than twice the cycle time. The analysis is presented in two parts: first on buffer costs between regular stations and blocks, second on buffer costs within and between blocks.

Given a balancing solution, for a task-sharing block, the minimal buffer requirements can be derived by defining a cyclical scheduling and buffer allocation residual problem. This can be viewed as residual version of a combined balancing and buffer allocation problem discussed in Chapter 5.

Consider the entry buffer: by hypothesis, it is preceded by a regular upstream station. Hence, one product must enter every CT time units (T.U.), while the block's first workstation will take more or less than CT T.U. depending on whether it has performed the shared task. Naturally, both stations of the block process, on average, one product every CT T.U.. While these oscillations imply that at least one unitary buffer is necessary, it is also possible to show that is sufficient under a specific condition ($D \leq CT$). The proof of this result is visually supported by Figure 34, and is based on measuring the time each product stays in the entry buffer: the difference between the processing time in the block's first station is compensated by the buffer. If processing is longer than CT (share task is performed by the block's first station) the difference equals the increment of time spent in the buffer for the next product. Conversely, lower processing times than CT (shared task not performed by the block's first station) will demand products earlier from the buffer, meaning the next product will spend less time spent in the buffer.

Figure 34 – Scheduling scheme for time products stay in buffers



Source: Lopes *et al.* (2021)

Let δ^+ and δ^- as the differences between CT and the long and short processing times, respectively. Let w_p indicate the time product p spent in the buffer. These values must be positive, as negative values would indicate insufficient buffer capacity. Furthermore, their maximum value dictates the minimal buffer capacity requirements: because products are feed by the upstream station every CT T.U., if the maximum time products stay in the station is $J \cdot CT$ (J is an integer) then a buffer with capacity J is sufficient. Expression (133) states how w_{p+1} can be computed in function of w_p and whether the first station performed the shared task or not. From this scheduling rule, it is easy to infer the feasibility condition expressed by Expression (134).

$$w_{p+1} = w_p - \delta^- \text{ or } w_{p+1} = w_p + \delta^+ \tag{133}$$

$$w_p - \delta^- \geq 0 \text{ or } w_p + \delta^+ \leq J \cdot CT \tag{134}$$

In order to determine the worst-case requirements of buffer capacity, we try to violate this feasibility condition, as stated by Expression (135). The first part can be turned into an equality, by adding a positive value ϵ, as stated by Expression (136). This allows w_p to be replaced

in the second part (Expression (137)). Further replacing the values of δ^+ and δ^- by functions of CT, D and the base time of the block's first station (B_1) leads to Expression (138). Its terms can be rearranged into the condition that implies J buffers are insufficient, i.e. Expression (139).

$$w_p - \delta^- < 0 \text{ and } w_p + \delta^+ > J \cdot CT \tag{135}$$

$$w_p = \delta^- - \epsilon \tag{136}$$

$$\delta^- - \epsilon + \delta^+ > J \cdot CT \tag{137}$$

$$CT - B_1 - \epsilon + B_1 + D - CT > J \cdot CT \tag{138}$$

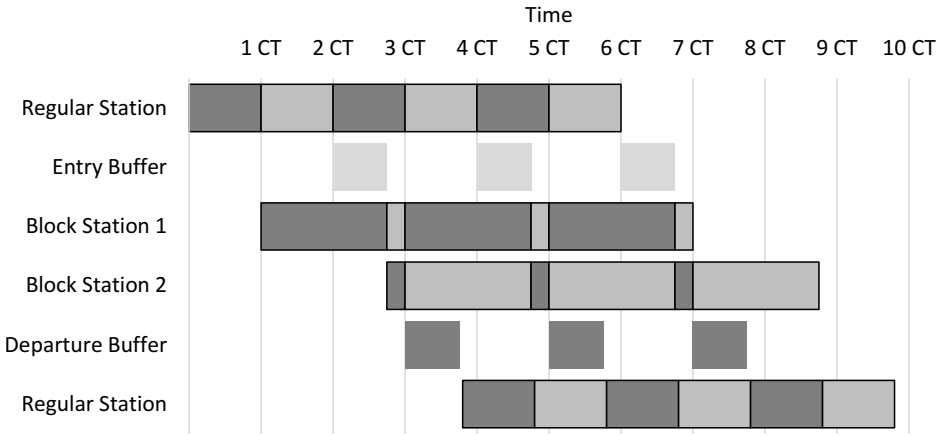
$$D > J \cdot CT + \epsilon \tag{139}$$

This means tasks shorter or equal to the cycle time ($D \leq CT$) require one entry buffer to be shared, and those bigger ($D > CT$) may require two. It is not possible to require more than two since $D \leq 2 \cdot CT$. A symmetrical argument, visually supported by Figure 34, proves the same requirement for departure buffers between blocks and regular stations. These are worst-case analysis, meaning for some cases they might not be required.

Once the worst-case requirements for buffer capacity between a regular station and a block station is defined, it is easy to determine the other two cases: buffer requirements between stations of a block and between blocks. Both follow the same argument, hereafter presented for the intra-block case: Add a dummy regular station between each station of the block. In the worst-case, this imposes a buffer requirement of capacity J between the regular station and each of the block's stations. This means that $2 \cdot J$ buffers are sufficient within the block: any schedule defined using the dummy regular station remains feasible when it is removed because it moves products regularly every CT T.U., meaning when one departs it, the next one enters. As mentioned before J is either one or two, depending on the tasks duration when compared to CT. In the particular case of two adjacent blocks, this argument leads to the conclusion that the worst-case requirements for buffers between them is the sum of two requirements: that between the first block and a regular station and that between a regular station and the second block. Hence, the cost of adjacent blocks add up in the worst-case.

A particular case warrants further exploring, namely that in which a task is evenly shared between stations, i.e. the 50-50 case. Because of its particular regularity, it is always possible to define schedules that require fewer buffers than the other fractional allocations do. Figure 35 presents an example of a cyclical scheduling between a block and two adjacent regular stations that (i) has a long task ($D > CT$), (ii) does not require intra-block buffers, and (iii) requires only one buffer between each regular station and the block ($J = 1$).

Figure 35 – Cyclic scheduling for a task sharing block between two regular stations



Source: Lopes *et al.* (2021)

Lastly, if a task is shared between the first and second station in the line, then that block does not require an input buffer. The same argument holds for the last two stations of the line and an output buffer. This is namely because it is commonly considered that the first station is never starved and the last one is never blocked (SCHOLL, 1999). This means that Expression (140) presents a worst-case MILP formulation for line buffer cost minimization step of FA-ALBP in the case of unpaced lines. In it, the cost parameter $G_{t,s}$ assumes the values presented in Table 34.

$$\text{Minimize} \quad \sum_{t \in T} \sum_{s \in S} G_{t,s} \cdot y_{t,s} \tag{140}$$

subject to: (118)-(129)

Table 34 – Values of $G_{t,s}$ in function of station s and task t

Fraction	General Case		50-50 Case	
Station s	middle	begin/end	middle	begin/end
$D_t \leq CT$	4	3	2	1
$D_t > CT$	8	6	2	1

9.6 EXPERIMENTS AND RESULTS

The proposed models were tested in three complementary experiments: First, Section 9.6.1 presents an illustrative and motivating example for a specific instance of the classical assembly line balancing dataset (SCHOLL, 1999). Second, Section 9.6.2 presents a screening on 1050 instances of a more recent extended dataset Otto *et al.* (2013). Lastly, Section 9.6.3 presents a case study in which the core studied concepts are applied to industrial data. The first experiment discusses in detail the advantages and trade-offs of the flexibility afforded by the fractional allocations. The second inquires on how specific instance parameters affect the aforementioned trade-offs. The third one discusses how to adapt the formulation to mixed-model lines as well as insights on the relevance of demand stability for the solutions' performance. All MILP models were solved using Gurobi 9.0 on a Core i7-8700 CPU (3.2 GHz, 6 CPUs, 12 threads) with 32 GB RAM and a 300 second time limit.

9.6.1 Illustrative Example

The Gunter instance was chosen from the Scholl (1999) dataset for the following reasons: First, it's optimal SALBP-2 solutions differed significantly from the trivial lower bound ($\lceil \sum_t D_t/|S| \rceil$) for multiple numbers of stations. Second, it is an instance that originates from a practical case study, meaning it is representative rather than randomly generated.

The FA-ALBP models were applied to this instance with all benchmark number's of stations (7-14) and with three values of K (2, 12, and ∞). Table 35 presents the optimal cycle time value ($\overline{CT}$) for each instance, and compares it to the trivial lower-bound ($LB = \lceil \sum_t D_t/|S| \rceil$). The internal storage costs required to realize the cycle time differences to the SALBP solutions is presented for both paced and unpaced lines as L-cost and B-cost, respectively. B-costs state the number of required buffers, a straightforward cost. In order to more adequately compare paced to unpaced lines, L-cost states the normalized line length cost: Products enter the line every CT time units, meaning this is the temporal length of a regular workstation (THOMOPOULOS, 1968). This means that dividing the line length cost (which is measured in time units by Expression (132)) by CT informs a normalized line length cost (e.g "1.5 products") that can be directly compared to the buffer cost (e.g. "buffers with capacity for 2 products"). For each number of stations and value of K, three MILP models are solved: The first minimizes cycle time and defines the performance requirement value $\overline{CT}$ for the latter two. The second minimizes

the line length cost (L-cost) required by a paced line to meet the performance requirement. The third minimizes the buffer cost (B-cost) required by an asynchronous line to meet the same performance requirement.

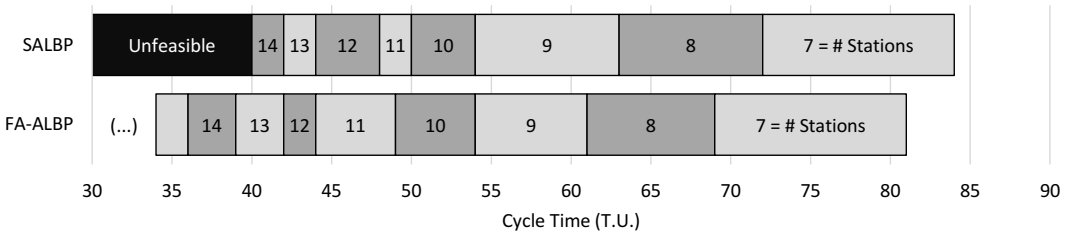
Table 35 – Results for the Gunther instance

	SALBP		FA-ALBP $K = 2$			FA-ALBP $K = 12$			FA-ALBP $K \to \infty$		
Stations	$\overline{CT}$	LB	$\overline{CT}$	L-cost	B-cost	$\overline{CT}$	L-cost	B-cost	$\overline{CT}$	L-cost	B-cost
7	72	69	70	0.57	1	70	1.14	4	69	1.71	7
8	63	61	62	0.37	2	61	1.31	3	61	0.75	3
9	54	54	54	0	0	54	0	0	54	0	0
10	50	49	49	0.1	2	49	0.2	3	49	0.2	3
11	48	44	45	0.42	1	45	0.84	3	44	1.73	11
12	44	41	42	0.33	1	42	0.66	3	42	0.66	3
13	42	38	40	1.55	3	40	3.1	10	39	6.77	25
14	40	35	38	3.84	8	37	3.11	22	36	7.66	30

Table 35 shows that the flexibility FA-ALBP allows is often capable of realizing all the improvement potential of the associated SALBP instance, i.e. the difference between its $\overline{CT}$ and the previously mentioned trivial lower bound, LB. For some combinations of parameters, however, this potential will be small or inexistent (9 stations). Furthermore, it is noticeable that higher values of K allow greater cycle time reduction for some cases, although they also require higher internal storage costs. Lastly, paced lines seem to require generally lower space costs to realize the cycle time improvement potential afforded by the fractional allocations.

Figure 36 presents a visual comparison between SALBP and FA-ALBP in regards to feasible values of cycle time. Notice that for some numbers of stations, the FA-ALBP performs as well as (and even better than) SALBP with one full additional station. This is particularly interesting as CT reductions and requiring fewer stations to reach the same CT translate to continuous monetary returns, while the line-length and buffer costs that are required for the FA-ALBP solution are essentially one-time investments.

Figure 36 – Feasible cycle time comparison between SALBP and FA-ALBP for the Gunther instance



Source: Lopes *et al.* (2021)

9.6.2 Screening

Additional tests on 1050 instances were performed in order to verify more general trends about the problem and the relevance of its parameters, in particular ordering strength (how restrictive precedence relations are) and task duration distributions. Otto *et al.* (2013) provided systematically generated instances for simple assembly line balancing, which were used for this screening. Instances were first solved for SALBP-2 with a fixed number of stations. The equivalent FA-ALBP instances were then solved with a cycle time minimization objective (Expressions (117)-(128)) so that the obtained FA-ALBP cycle times can be compared to SALBP ones. Two further executions of the FA-ALBP model used the obtained cycle time as a constraint, and employed the described internal storage minimization goals for paced (Expressions (132) and (118)-(128)) and unpaced lines (Expressions (140) and (118)-(128)). All FA-ALBP instances set $K = 2$. Furthermore, the CT minimization FA-ALBP run used SALBP's answer for warmstarts, and the space cost minimization FA-ALBP runs used the CT minimization FA-ALBP one.

Tables 36 and 37 present the results for small and medium instances, respectively: average time required (in seconds) per instance is informed. The cycle time differences are described in three terms: the average theoretical potential (avg. Theo.) compares the obtained SALBP cycle time to the lower bound ($\lceil \sum_t D_t / |S| \rceil$); the average realized potential (avg. Real.) means the average difference between cycle times obtained for SALBP and FA-ALBP as a percentage of the SALBP's cycle time. The best case column (Best Real.) presents the highest such percentage value. Internal storage costs are informed as average increases in line length compared to the line's base length (7 for small instances and 15 for large ones). For paced lines, the line length cost (L-cost) is normalized by dividing it by the cycle time (THOMOPOULOS, 1968), so that the increase can be measured in number of stations. For unpaced lines, buffers are considered to be equivalent in size to stations for the purposes of the buffer cost (B-cost) measure.

In each line of Table 36 the results are presented for the subset of instances with the specified parameter (OTTO *et al.*, 2013). The first three lines divide instances in terms of order strength (OS), i.e. how restrictive the precedence diagram is. The following three separates instances in terms of their task duration distributions (dist(D_t)): peak at the bottom (PB), peak in the middle (PM), and bimodal (BM). It is apparent that higher OS translates to both higher theoretical (Theo. column) and realized (Real. column) cycle time differences averages. However,

this is a general tendency and not a guarantee as illustrated by the 15.1% best realized difference (Best column) with OS=0.2. The task duration distribution, however, appears to be less relevant for the difference between SALBP and FA-ALBP cycle times.

Table 36 – Screening results for small instances (20 tasks 7 stations; CPU times in seconds)

Parameter	SALBP	FA-ALBP			CT Difference			Avg. Costs (%)	
OS/dist(D_t)	time	CT time	L time	B time	Theo.	Real.	Best	L-cost	B-cost
0.2	0.8	26.5	33.9	28	2.0%	1.4%	15.1%	9.9%	42.9%
0.6	0.1	3.9	2.1	1.8	3.6%	2.1%	9.9%	9.4%	40.1%
0.9	0.1	0.9	0.1	0.1	8.2%	3.9%	17.2%	8.1%	33.7%
PB	0.3	7.6	9.3	7.6	3.4%	1.9%	9.9%	9.4%	39.6%
PM	0.7	26	30.7	23.8	3.5%	1.9%	7.9%	8.9%	39.4%
BM	0.2	5.9	6.3	7.1	3.9%	2.3%	17.2%	9.6%	40.3%

It is also noticeable that FA-ALBP is more computationally costly than SALPB, as shown by the average time required to solve the small instances to optimality. Furthermore, the average space cost associated to the cycle time reduction does not seem proportional to it and its average behavior seems insensitive to instance parameters.

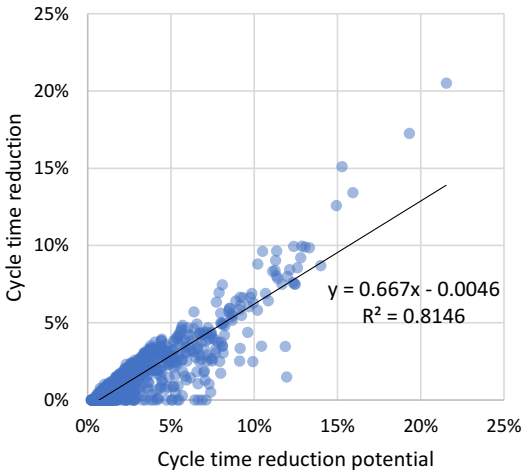
Table 37 – Screening results for medium instances (50 tasks, 15 stations; CPU times in seconds)

Parameter	SALBP	FA-ALBP			CT Difference			Avg. Costs (%)	
OS/dist(D_t)	time	CT time	L time	B time	Theo.	Real.	Best	L-cost	B-cost
0.2	298.7	300	14.3	14.7	0.7%	0.0%	3.4%	0.2%	0.9%
0.6	197.9	300	103.1	100.6	1.4%	0.3%	20.5%	2.4%	14.1%
0.9	8.1	64.2	68.2	55.2	7.0%	3.7%	9.2%	1.7%	8.6%
PB	202.4	262	47.9	48.3	1.9%	0.7%	20.5%	1.7%	8.6%
PM	248.7	276	60.1	54.1	1.8%	0.5%	5.7%	1.1%	6.7%
BM	191	261	72	69.6	2.0%	0.8%	9.2%	2.4%	12.4%

The medium cases are noticeably more difficult, with whole subsets of instances failing to reach optimality proofs within the time limit. It is also noticeable that, in average, the cycle time reduction was smaller with many instances reporting no difference (similarly to the case study with 9 stations, see Figure 36). This is mostly attributable to the lower observed potential, which seems to emerge from the greater flexibility of medium instances (50 tasks) when compared to smaller ones (20 tasks). However, under high precedence constraints (OS=0.9), similar results to the small cases were obtained. Furthermore, the best case for medium instances presented higher realized CT difference (20.5%) than the small ones (17.2%). Task distributions do not appear to be any more relevant for medium instances than they were for previously, and average space costs also seem mostly unaffected by parameters, except for low OS. However, this is easily explained: medium instances with low OS often produced solutions with the same value of CT as the SALBP equivalent, meaning that no buffers or additional line lengths in comparison to SALBP is required.

The relationship between realized potential and theoretical potential is further elucidated by Figure 37, in which each instance is translated to a semi-transparent point. Naturally, the realized cycle time reduction is bounded by its potential. The figure suggests that (for $K = 2$) around two thirds ($y = \frac{2}{3}x$, $R^2 = 0.81$) of the available potential (difference between SALBP's CT and a perfect division) is realized. Furthermore, this benefit seems easier to extract for SALPB instances with higher potential.

Figure 37 – Realized cycle time reduction versus available potential

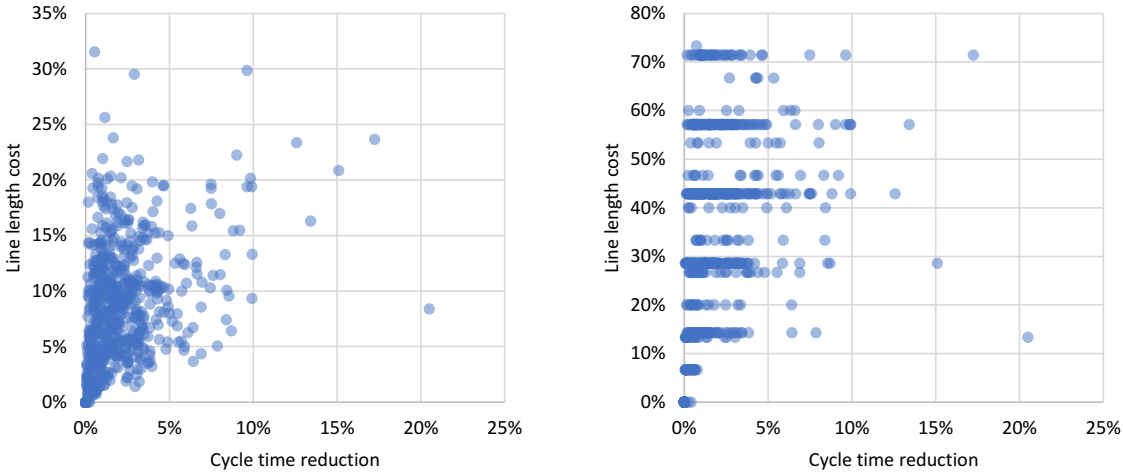


Source: Lopes *et al.* (2021)

Lastly, Figure 38 illustrates the distribution of internal storage costs of the realized cycle time reductions for paced lines (Figure 38(a)) and unpaced ones (Figure 38(b)). It is apparent that, while higher cycle time reductions tend to imply higher costs, the correlation is rather weak. It is also important to notice that storage costs are mostly one-time investments and only part of the total costs of running the assembly line, while cycle time reductions reflect on continuous performance improvements, meaning that a 20% longer line for a 10% cycle time reduction can be a very favorable trade-off. In terms of line pace comparison, paced lines tend to require much smaller costs than unpaced ones to realize the same cycle time reductions. This, however, depends on the assumption that buffers are equal stations in size, which is reasonable for final lines in the automobile industry, but may not be the case for smaller products such as electronics devices - for which (even "infinite") buffers can be substantially smaller than stations.

Figure 38 – Realized cycle time versus required internal storage cost

(a) Paced Lines **(b) Unpaced Lines**



Source: Lopes *et al.* (2021)

9.6.3 Application to Practical Data

This section describes the results obtained by the proposed mathematical formulation when applied to data of a practical industrial case. The instance data describes a car seat assembly line dedicated to the manufacture of two product models: a simpler high-demand one, and a more complex one with lower demand. The line is asynchronous unpaced, has seven workstations as well as physical space available for two unitary buffers ($B_{max} = 2$). Chapter 2 discusses this line in the context of cyclical scheduling - which requires stable demands. The present chapter described fractional allocations for single model assembly lines, therefore, some adaptations were required for the present case study. First, each product model m is modeled as having its own cycle time CT_m. Second, models have expected relative demand O_m, which must sum to 100%. This case study discusses what happens if these demands are not stable. Third, the throughput of a line with fractional task allocations is measured as a weighted average of these model-wise cycle times. Finally, task allocations are required to be the same for both product models, in order to maintain the benefits of worker specialization. Data required to reproduce this case study is made available as this Lopes *et al.* (2021)'s supplementary material.

This allows a mixed-model version of the proposed formulation to be defined using most of the constraints of Section 9.4's model, as presented by Expressions (141)-(143). In it, M is the set of product models (in this case, $M = \{1, 2\}$), Constraint (142) is the adapted version of Constraint (122): each task t is modeled as having a specific duration $D_{t,m}$ for each product

model m. If a task does not exist for a product model, its duration is set as zero. Constraint (143) states the buffer resource limit. In this case study, $K = 2$ was used because it offers a reasonable trade-off between cycle time reduction, space requirements and implementation simplicity. In this case study, expected relative demands are, approximately, of 83% and 17% for product models 1 and 2, respectively - 5 units of the first product model per unit of the second one, similarly to Chapter 2.

$$\text{Minimize} \quad \sum_{m \in M} O_m \cdot CT_m \tag{141}$$

subject to:

Constraints (118)-(121), (123)-(128)

$$\sum_{t \in T} D_{t,m} \cdot z_{t,s} \leq CT_m \qquad \forall\, s \in S, m \in M \tag{142}$$

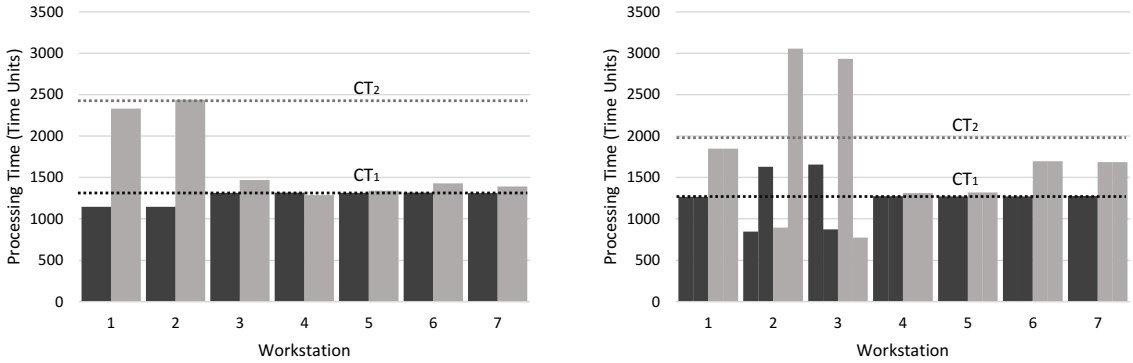
$$\sum_{s \in S} \sum_{t \in T} G_{t,s} \cdot y_{t,s} \leq B_{max} \tag{143}$$

The balancing solution obtained using this formulation is compared to that obtained by the simultaneous balancing-sequencing-buffer allocation one, described in Chapter 5, using the same buffer limit. This benchmark is hereafter referred to as “Cyclical Scheduling”. These model-wise processing ties of these solutions are presented by Figure 39. In it, the darker bars represent product model one, and the lighter ones represent product model two. In Figure 39(b), the second and third stations have have two processing times for each product model - representing the cases in which the shared task is and is not performed at that station. For those stations, the model-wise cycle time is bounded by the average between the high and low values. Consequently, Figure 39(b) presents lower model-wise cycle time values than Figure 39(a). However, this better performance is associated to processing time oscillations that require a model-wise cyclical scheduling to be realized. The buffer allocations for each solution also differ: for the benchmark, single unitary buffers are optimally assigned after stations one and two. Meanwhile the fractional task allocations solution assigns buffers to after stations one and three.

The solutions’ performances are compared by considering multiple realized demand scenarios that differ from the expected scenario (83%-17%). For both formulations, the balancing solution is fixed and its average cycle time (Avg. CT) is computed for each realized demand scenario. For the fractional allocations solution, the performance is measured as a weighted average of the model-wise cycle times. For the cyclical scheduling benchmark, an additional

Figure 39 – Processing times for each balancing solution

(a) Cyclical Scheduling Benchmark **(b) Fractional Allocation**



Source: Lopes *et al.* (2021)

residual cyclical sequencing and scheduling problem is solved. Table 38 compares the performance of these formulations in terms of their average cycle time values (Avg. CT), as well as their theoretical efficiency (Theo. Eff.), i.e. comparison to a simple division of total task time by the number of stations. Lastly, the solutions' relative performance is computed by dividing their average cycle times.

Table 38 – Performance comparison for multiple demand scenarios

Demand	Cyclical Scheduling		Fractional Allocations		Relative
Scenario	Avg. CT	Theo. Eff.	Avg. CT	Theo. Eff.	Performance
100%-0%	1316	96.2%	**1276**	99.2%	103.1%
90%-10%	1349.5	96.8%	**1345.9**	97.1%	100.3%
83%-17%	***1372***	*97.2%*	*1392.5*	*95.7%*	*98.5%*
80%-20%	**1412.6**	95.3%	1415.8	95.1%	99.8%
70%-30%	1537.9	90.2%	**1485.7**	93.3%	103.5%
60%-40%	1667.2	85.6%	**1555.6**	91.7%	107.2%
50%-50%	1792.5	81.9%	**1625.5**	90.3%	110.3%
40%-60%	1921.8	78.5%	**1695.4**	88.9%	113.4%
30%-70%	2051.1	75.5%	**1765.3**	87.7%	116.2%
20%-80%	2180.4	72.8%	**1835.2**	86.6%	118.8%
10%-90%	2309.7	70.5%	**1905.1**	85.5%	121.2%
0%-100%	2439	68.4%	**1975**	84.5%	123.5%

Notice that, on the one hand, the benchmark outperforms the fractional task allocations solution for the original or "expected scenario" (highlighted in italic), as well as for the neighboring 80%-20% one. On the other hand, the cyclical scheduling benchmark is outperformed for both product models individually (100%-0% and 0%-100% scenarios), as well as all other demand scenarios. Indeed, the benchmark's solution performance is contingent on the stability of demands. This leads to increasingly poorer performance the more realized demands differ from the expected ones. Furthermore, the worst relative performance by the fractional allocations formulation was 98.5% (1.5% worse), while its highest relative performance was over 20%

better than the benchmark. This suggests that fractional task allocations can lead to more robust solutions in the context of mixed-model lines.

9.7 DISCUSSIONS AND CONCLUSIONS

This chapter questions the assumption of the binary nature of task-station assignments in assembly line balancing problems. While other works had addressed some aspects of what amounts to adjacent stations "sharing tasks", none had analytically investigated the internal storage costs required to realize the cycle time reduction potential. A mathematical model for the Fractional Allocation Assembly Line Balancing Problem (FA-ALBP) is presented, with three variants: one that seeks to minimize the line's cycle time, and two that seek to minimize the internal storage required to realize a given cycle time value.

Line length is the cost metric for paced lines; on the other hand, buffer requirements is the cost metric for asynchronous unpaced ones. These costs are determined using worst-case analysis for both types of line control. In both cases, these costs are shown to be functions of the shared task's duration and of the allocation fraction, in particular its irreducible integer denominator (contrary to previously hypothesized by Bukchin and Sofer (2011)). For paced lines, line costs are continuous and proportional to the shared task's duration. For unpaced ones, the buffer requirements are discrete, they depend on task's duration relative to the cycle time, on whether the allocation fraction is 1/2 ($K = 2$) or not, and on whether the assignment happens on the first/last stations or not.

This chapter shows that for specific instances, fractional allocations allow lower cycle times with the same number of stations or the same cycle time with fewer stations than what is possible with SALBP. Furthermore, contrary to the base problem, the longest task is not a bound on cycle time, meaning that lower values of cycle time are possible for FA-ALBP than SALBP even when the latter has unlimited stations. Increasing the fractional allocation's maximum integer denominator K can allow greater reductions, but is also associated with higher internal storage costs and more complex cyclical scheduling. The most adequate value for K can, therefore, depend on instance and context-specific managerial preferences, although the smallest value of $K = 2$ seems to offer a simple and effective compromise.

This difference in cycle time is not, however, guaranteed to exist. Screening on a 1050-instance dataset showed that for many instances, SALBP optimal solutions are close to a perfect division of tasks' durations amongst stations. However, some instances lead to cycle time

reductions as high as 20%. Experiments suggest that this difference tends to be larger for more restrictive instances (precedence diagrams with high order strength), and that the distributions of task durations is not an as relevant factor. In terms of the internal storage cost required to realize these cycle time differences, paced lines tend to outperform their unpaced counterparts assuming that buffers are similar in size to stations for the latter. The required storage costs vary significantly on an instance by instance basis. However, these are mostly one-time investments and cycle time reductions represent continuous gains. This suggests fractional allocations can be an interesting alternative to increase performance: contrary to increasing the number of workstations, internal storage usually has relatively low continuous operating costs. Such indirect costs tied to internal storage such as Work in Progress (WIP) and Time in System (TIS) can be analyzed for both line types: In the case of paced lines, these costs are trivially proportional (WIP) or equal (TIS) to the line length cost measured in time units. For unpaced lines, the WIP and TIS costs can easily be shown to equal those of paced lines with the same balancing solution.

With minor adaptations, the proposed methodology was also applied to industrial data of a mixed-model assembly line. The resulting solution offered better cycle time performance than a previous cyclical scheduling benchmark solution for both product models. The resulting line balancing also lead to better performance the more realized demands differ from the expected ones. That was achieved with the simplest possible sharing (50-50%) of a single task.

The following key managerial insights can be drawn from this Chapter: fractional allocations are a performance enhancing possibility tied to an internal storage cost trade-off. They can be seen as an alternative to parallel stations with some advantages tied to the fact that only one task is shared: Regarding worker training, they require less and are also tied to less specialization losses. Regarding physical structure, they require fewer layout changes, as well as less tool duplication costs, specially for asynchronous lines. The main disadvantage of fractional allocations are the internal storage costs, which are tied to greater work-in-progress and time-in-system than the parallel stations alternative. Furthermore, while this chapter focuses on single model assembly lines, the industrial data case study suggests that fractional allocations can also be useful for mixed-model lines. Indeed, the case's balancing solutions was remarkably more robust regarding demand fluctuations.

Finally, FA-ALBP is shown to be at least either weakly or strongly NP-complete, depending on how the number of stations scales with instance size. This means that larger instances might be intractable for generic solvers: experiments suggest that just the cycle time

minimization effort tends to be significantly harder than SALBP. Indeed, a drawback of the proposed mathematical model is its difficulty in solving larger size instances. This drawback is increase by the fact that a follow-up step of internal storage cost minimization is also desirable, in order to achieve the cycle time reduction with minimum internal storage cost.

10 THE MULTI-OBJECTIVE PACED LINE CASE

This chapter is an adapted version of a conference paper presented in the XXI ROADEF (LOPES *et al.*, 2020a), held in Montpelier (France) in 2020. That work shared the prize of best student paper presented in the conference. In it, a method to solve multi-objective problems with mixed-integer linear Pareto fronts is presented. The method has been developed as a means to offer a more meaningful solution for the paced assembly line balancing-sequencing problem. As suggested by Chapters 8 and 9, this class of problems can display a multi-objective dispute between line length and cycle time.

10.1 CONTEXT

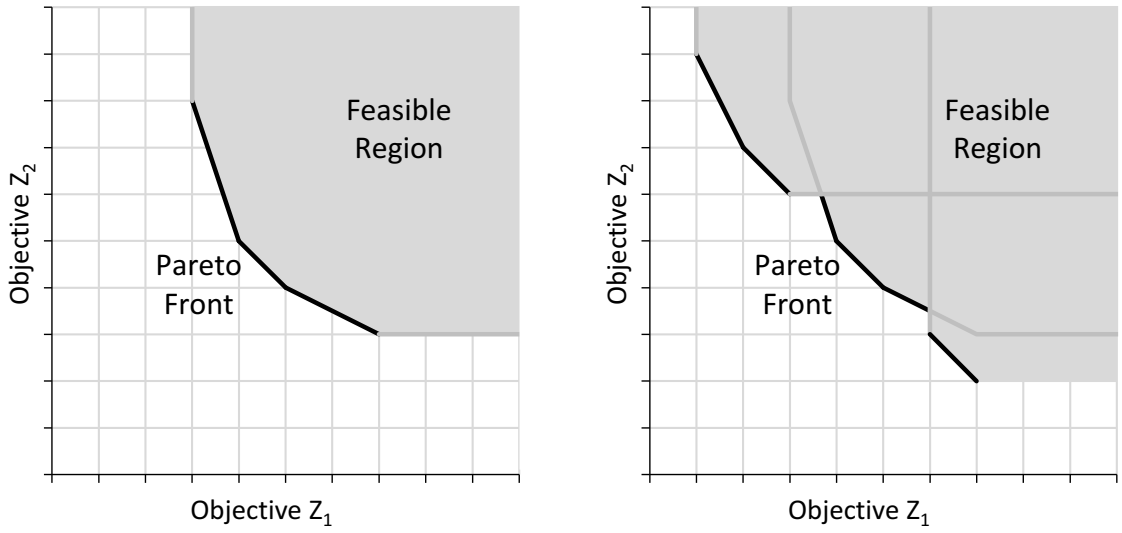
Frequently, in various practical environments, decision makers have to consider multiple objectives at the same time. These are often in conflict, meaning it is not possible to simultaneously reach the best score in all of them. This context defines multi-objective optimization, which commonly employs one of two main strategies: (i) using weight factors for each objective, transforming the problem back into a single objective one; and (ii) obtaining the set of Pareto efficient solutions and allowing the decision maker to choose *a posteriori* (KELLER, 2017; RANGAIAH, 2009). This chapter focuses on the latter case. For the sake of simplicity, this chapter considers a problem definition with two minimization objectives.

Multi-objective problems can relate to integer programming, which implies each integer solution leads to one value for each objective. Furthermore, solution methods are commonly problem-specific (UNGULU; TEGHEM, 1994). However, in the case of mixed-integer linear programming (MILP) problems, fixing the integer variables and optimizing the continuous ones of a solution can lead to a Pareto front made of continuous segments, as illustrated by Figure 40(a). By combining the Pareto fronts of various integer solutions, a part-wise continuous Pareto front is obtained, as depicted by Figure 40(b).

Chapter 8 provides an example of the aforementioned multi-objective disputes in the context of assembly line balancing problems (SCHOLL, 1999). In particular, cycle time (tied to production rate) and line length (tied to implementation costs) display a mixed-integer linear multi-objective dispute, when treated as minimization objectives. A recent work (DEFERSHA; MOHEBALIZADEHGASHTI, 2018) on simultaneous line balancing-sequencing presents a

Figure 40 – Combining feasible regions of integer solutions

(a) Single Integer Solution **(b) Multiple Integer Solutions**



mathematical model that can be easily adapted to test that dispute.

While some methods have been developed to solve multi-objective MILP problems, they are rather limited in terms of the size of problems that are treatable. Recent branch and bound methods (MAVROTAS; DIAKOULAKI, 2005; VINCENT *et al.*, 2013), for instance, were only tested on instances with up to 70 variables and constraints, which were considered "large instances". More commonly (KELLER, 2017; RANGAIAH, 2009), two main approaches are applied: the multiple-weighted sums method and the epsilon-constrained method. The former consists on solving the problem using weighted sums for each objective, gradually changing the weights to reach new solutions in the Pareto front. The later consists in minimizing one objective and using bounds on the others. These limits can then be gradually increased in further executions to generate the Pareto front.

These two methods share the strategy of solving multiple single objective MILP problems to solve a multi-objective one. The inherent advantage of this approach is that the multi-objective problem is tractable whenever multiple executions of the single objective one is. Given the increasing strength of commercial MILP solvers, it is reasonable to expect that exact solutions are attainable for problems ranging from hundreds to a few thousands variables and constraints. However, there are some limitations of these methods: the weighted goal functions only allows some extreme point solutions to be found; on the other hand, the epsilon method might find the same integer solution multiple times, leading to redundancies. The aforementioned capacity to

solve MILP instances does not address these limitations, which motivates the development of a new method that overcomes them.

This chapter is structured as follows: Section 10.2 presents an overview of the class of studied problems; Section 10.3 presents the proposed method to solve them; Section 10.4 describes a mathematical model for balancing and cyclically sequencing paced mixed-model assembly lines. Section 10.5 presents a example that illustrates why this balancing-sequencing problem belongs to the class of studied problems. Section 10.6 revisits Chapter 2's case study data and compares pace to unpaced line controls. Finally, Section 10.7 summarizes this chapter's contributions.

10.2 PROBLEM DEFINITION

Consider a mixed-integer linear problem with two minimization objectives Z_1 and Z_2, as stated in Expression (144). Let these goals be bounded by linear functions of two sets of decision variables **x** and **y**, as presented in Expression (145) and (146). Finally, let **x** represent integer/binary decisions and **y** be the set of continuous decision variables, as stated in Expression (148).

$$\text{Minimize } Z_1 \text{ and } Z_2 \tag{144}$$

Subject to:

$$Z_1 \geq \mathbf{f_1}(\mathbf{x},\mathbf{y}) \tag{145}$$

$$Z_2 \geq \mathbf{f_2}(\mathbf{x},\mathbf{y}) \tag{146}$$

$$A \cdot \mathbf{x} + B \cdot \mathbf{y} \geq \mathbf{C} \tag{147}$$

and:

$$\mathbf{x} \in \mathbb{Z}, \mathbf{y} \in \mathbb{R} \tag{148}$$

By fixing all the integer variables **x** to a specific set $\bar{\mathbf{x}}$, an integer solution is defined. Notice that the residual problem defined by the **y** variables is linear, meaning its feasible region is convex. By defining a plane such that the goals Z_1 and Z_2 correspond to its axis, it is possible to project integer solutions to it, defining convex feasible regions as illustrated by Figure 40(a): an area bounded by Pareto-efficient continuous solution segments and lines for the minimum possible values of each objective. Naturally, by combining multiple integer solutions, a global

part-wise continuous Pareto front is defined for the studied problem (Figure 40(b)).

10.3 PROPOSED METHOD

In order to define the optimal Pareto front for this problem, a method composed of two parts is presented hereafter: First, it is necessary to extract the full continuous set of solutions for a given integer solution. This is achieved using a duality-based method described in Section 10.3.1. Second, a procedure for generating an initial global Pareto front and then refining it to optimality is required to determine which set of integer solutions is Pareto efficient. This is done by exploiting a second MILP-based model, and is described in Section 10.3.2. In these sections, an overline is used to distinguish variables/goal functions from reference values they can be set to. For instance, $\mathbf{x}$ represents the variables, and $\overline{\mathbf{x}}$ refers to values they can be set to.

10.3.1 Extracting the Pareto front of each Integer Solution

Consider a given integer solution $\mathbf{x} = \overline{\mathbf{x}}$. The residual linear problem ties the goals Z_1 and Z_2 to the set of continuous variables $\mathbf{y}$. This defines a feasible region $\Omega(\overline{\mathbf{x}})$ as stated by Equation (149):

$$\Omega(\overline{\mathbf{x}}) = \{(\mathbf{y},Z_1,Z_2) : Z_1 \geq \mathbf{f_1}(\overline{\mathbf{x}},\mathbf{y}),\, Z_2 \geq \mathbf{f_2}(\overline{\mathbf{x}},\mathbf{y}),\, A \cdot \overline{\mathbf{x}} + B \cdot \mathbf{y} \geq \mathbf{C},\, \mathbf{y} \in \mathbb{R}\} \tag{149}$$

Algorithm 7 describes how to define the Pareto front of this feasible region $\Omega(\overline{\mathbf{x}})$ by solving a set of linear programming problems. Similarly to other works (IGNIZIO, 1985; POURKARIMI; ZAREPISHEH, 2007) on multi-objective problems, Algorithm 7 exploits duality theory to explore the feasible region of the residual linear problem. However, contrary for instance to reference (ECKER; KOUADA, 1978), the analysis is based on the objective space (Z_1-Z_2 plane) rather than the decision space (defined by the $\mathbf{y}$ variables).

This algorithm solves linear programming models in its three main steps (A, B, C), which are illustrated in the criterion-space by Figures 41(a) and 41(b). The step A is the most straightforward, and consists in simply using Z_2 as the only goal function and solving the linear problem to optimality. The optimal value of the goal function in this step is used to define a resource constraint, as stated in lines 2-4.

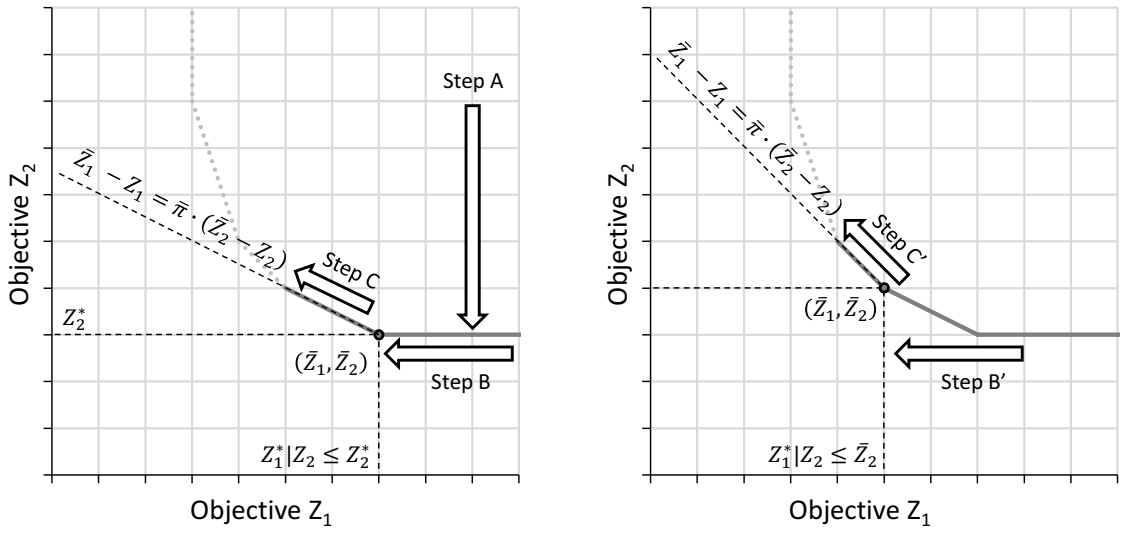
In Step B, a model that incorporates this resource constraint is solved, and its objective

Algorithm 7 – ParetoGiven($\overline{\mathbf{x}}$)

1: $\overline{Z}_2 \leftarrow \min\ Z_2 : (\mathbf{y}, Z_1, Z_2)\ \in\ \Omega(\overline{\mathbf{x}})$ ▷ Step A
2: Constraint (I) $\leftarrow Z_2 \leq \overline{Z}_2 + \varepsilon$ ▷ ε is a small number
3: $\pi \leftarrow$ dual variable associated to Constraint (I) ▷ $\overline{\pi}$ states the variable's value
4: $\Omega^1(\overline{\mathbf{x}}) \leftarrow \Omega(\overline{\mathbf{x}}) \cap \{(\mathbf{y}, Z_1, Z_2)$: Constraint (I) is satisfied$\}$
5: $\overline{Z}_1 \leftarrow \min Z_1 : (\mathbf{y}, Z_1, Z_2)\ \in\ \Omega^1(\overline{\mathbf{x}})$ ▷ Step B - save value of $\overline{\pi}$
6: Store $(\overline{Z}_1, \overline{Z}_2)$
7: **if** $\overline{\pi} < 0$ **then**
8: $\quad \Omega^2(\overline{\mathbf{x}}) \leftarrow \Omega(\overline{\mathbf{x}}) \cap \left\{(\mathbf{y}, Z_1, Z_2) : \overline{Z}_1 - Z_1 = \overline{\pi} \cdot (\overline{Z}_2 - Z_2)\right\}$
9: $\quad \overline{Z}_1 \leftarrow \min Z_1 : (\mathbf{y}, Z_1, Z_2)\ \in\ \Omega^2(\overline{\mathbf{x}})$ ▷ Step C
10: $\quad \overline{Z}_2 \leftarrow \overline{Z}_2 - (\overline{Z}_1 - Z_1)/\overline{\pi}$ ▷ Update $\overline{Z}_2$ for Constraint (I)
11: $\quad$ Go to Line 2
12: **end if**
13: **return** stored values of $(\overline{Z}_1, \overline{Z}_2)$

Figure 41 – Generating the feasible region of an integer solution

(a) First Iteration **(b) Following Iterations**



is set to minimize Z_1. The dual variable π stores the shadow price of $\overline{Z}_2$ as a resource, i.e. what is the relative reduction of Z_1 enabled by a marginal increase of $\overline{Z}_2$. A small value (ε) can be added to the RHS (Right Hand Side) of this constraint in order to ensure that its slack is zero and that a valid shadow price π is generated. Adequately defining this ε value may require problem-specific knowledge. The minimal value of Z_1 obtained in this step defines the first Pareto efficient point for the integer solution, namely ($\overline{Z}_1$, $\overline{Z}_2$).

Assuming a non-zero shadow price π, its value can be combined with that information to define the line that contains this point and its associated Pareto efficient segment, as stated in Algorithm 7's line 8. This leads to step C, in which Z_1 is minimized again. The resulting value of Z_2 is then used to update $\overline{Z}_2$ and iterate step B. This process iterates until $\overline{\pi}$ is set to

zero, meaning a vertical line is reached in the feasible region and no further reduction of Z_1 is possible.

10.3.2 Defining and refining the global Pareto front

The previous section detailed how to extract the Pareto front associated to an integer solution $\mathbf{x} = \overline{\mathbf{x}}$. Algorithm 8 determines the full set of efficient $\overline{\mathbf{x}}$ solutions, as well as the global Pareto front. Algorithm 8 first steps determine the global minimal values of Z_1 and Z_2. It is possible, however, that their combined Pareto fronts do not cover all possible ranges of Z_1. Figure 42(a) illustrates how to tie them using artificial Pareto segments such that all potentially efficient values of Z_1 are covered. The resulting initial front contains segments that cover all potentially relevant values of Z_1, but is not guaranteed to contain solutions of the problem's true Pareto front.

Algorithm 8 – Defining the global Pareto front

```
 1: x̄1 ← x : Z1 is minimized
 2: Points1 ← ParetoGiven(x̄1)
 3: x̄2 ← x : Z2 is minimized
 4: Points2 ← ParetoGiven(x̄2)
 5: if min Z̄1 ∈ Points2 > max Z̄1 ∈ Points1 then        ▷ Not all ranges of Z1 are covered
 6:     Points1.Add((min Z̄1 ∈ Points2, min Z̄2 ∈ Points1))        ▷ Add artificial segment
 7: end if
 8: IncFront ← Generate Initial Front by Combining Adjacent (Z̄1, Z̄2) in Points1 and Points2
 9: repeat
10:     for all seg ∈ IncFront: seg.TruePareto=false do
11:         Δ ← max Δ : Δ =distance(seg, (Z1,Z2)) with (x,y, Z1, Z2) ∈ Ωseg
12:         Store x̄ values for x variables in previous step
13:         if Δ = 0 then
14:             seg.TruePareto=true
15:         else
16:             Points ← ParetoGiven (x̄)
17:             Combine new solution's Pareto front with IncFront
18:             Discard fully dominated segments in IncFront
19:             exit for
20:         end if
21:     end for
22: until all segments in IncFront are flagged as TruePareto
23: return IncFront
```

Algorithm 8 then tests each segment in the incumbent Pareto set using a maximization model (Line 11). This model tests each segment seg by considering the region Ω_{seg} (defined by Equation (150)), under the segment. The test maximizes Δ, a continuous variable that measures

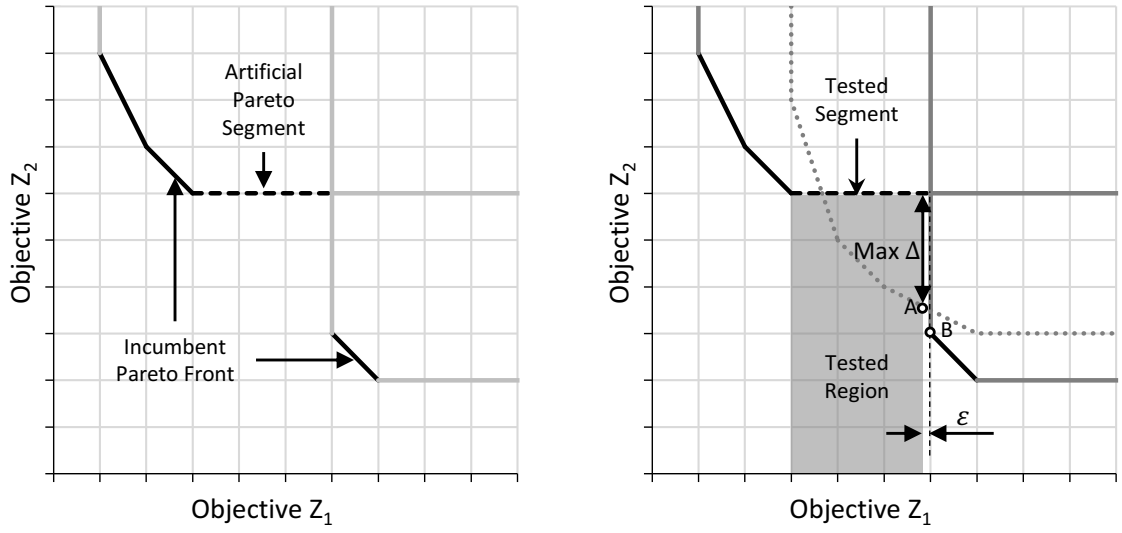
how much bellow the tested segment a given solution is.

$$\Omega_{seg} = \{(\mathbf{x},\mathbf{y},Z_1,Z_2) : (145) - (148),\, p_1.Z_1 \leq Z_1 \leq p_2.Z_1 - \varepsilon\} \tag{150}$$

The maximum value of Δ defines an optimality test for the line segment: if the resulting maximum value of Δ is zero, then there is no solution bellow the line segment and it is a Pareto efficient. Otherwise, a new integer solution is found and guaranteed to partially dominate the segment for at least one value of Z_1. Figure 42(b) illustrates such optimality test for a line segment. It highlights a particularity: The small value ε must be subtracted from Z_1^+ in Expression (150) so that the model does not consider as feasible a solution that belongs to the next line segment. That is, so that the best solution found under the tested region is the one marked with an "A" rather than the one marked with a "B" in Figure 42(b). Naturally, adequately defining the ε value may require problem specific knowledge.

Figure 42 – Determining the optimal Pareto front

(a) Initial incumbent Pareto front **(b) Delta test for new solutions**



10.4 A MODEL FOR PACED LINES

This section presents a mixed-integer programming model to describe continuous paced lines. It is similar to the model presented in Chapter 3, but contains more scheduling variables to reflect the paced nature of the line: in unpaced lines scheduling variables existed for entry (Tin), processing (Tx) and departure ($Tout$) times at each station. Meanwhile, in paced lines,

products may physically enter (Tin states entry time, Lin states entry position) the station before processing start ($Tstart$) as well as conclude processing before leaving it ($Tend$ states moment processing is complete, $Lend$ states position in which it occurs), meaning more scheduling variables are required. Furthermore, to measure line length additional scheduling variables are added to define positions (Lin and $Lend$) as well as station boundaries ($Lmin$ and $Lmax$).

Minimize CT and L (151)

subject to:

$$\sum_{s \in S} x_{t,s} = 1 \qquad \forall\, t \in T \qquad (152)$$

$$\sum_{s \in S} s \cdot x_{t_1,s} \leq \sum_{s \in S} s \cdot x_{t_2,s} \qquad \forall\, (t_1,t_2) \in R \qquad (153)$$

$$\sum_{m \in M} y_{p,m} = 1 \qquad \forall\, p \in P \qquad (154)$$

$$\sum_{p \in P} y_{p,m} = N_m \qquad \forall\, m \in M \qquad (155)$$

$$Tx_{p,s} \geq \sum_{t \in T} D_{t,m} \cdot x_{t,s} - (1 - y_{p,m}) \cdot H \qquad \forall\, p \in P, s \in S, m \in M \qquad (156)$$

$$\sum_{p \in P} Tx_{p,s} = \sum_{t \in T,\, m \in M} N_m \cdot D_{t,m} \cdot x_{t,s} \qquad \forall\, s \in S \qquad (157)$$

$$Tstart_{p,s} \geq Tin_{p,s} \qquad \forall\, p \in P, s \in S \qquad (158)$$

$$Tend_{p,s} = Tstart_{p,s} + Tx_{p,s} \qquad \forall\, p \in P, s \in S \qquad (159)$$

$$Tend_{p,s-1} \leq Tin_{p,s} \qquad \forall\, p \in P, s \in S : s > 1 \qquad (160)$$

$$Tend_{p-1,s} \leq Tstart_{p,s} \qquad \forall\, p \in P : p > 1, s \in S \qquad (161)$$

$$CT \cdot |P| \geq Tend_{|P|,s} - Tstart_{1,s} \qquad \forall\, s \in S \qquad (162)$$

$$Lin_{p,s} = (Tin_{p,s} - Tin_{p,1}) \cdot V \qquad \forall\, p \in P, s \in S \qquad (163)$$

$$Lend_{p,s} = (Tend_{p,s} - Tin_{p,1}) \cdot V \qquad \forall\, p \in P, s \in S \qquad (164)$$

$$Lin_{p,s} \geq Lmin_s \qquad \forall\, p \in P, s \in S \qquad (165)$$

$$Lend_{p,s} \leq Lmax_s \qquad \forall\, p \in P, s \in S \qquad (166)$$

$$L = Lmax_{|S|} \qquad (167)$$

$$Lmin_s \geq Lmax_{s-1} \qquad \forall\, s \in S : s > 1 \qquad (168)$$

$$Tin_{p,1} - Tin_{p-1,1} = CT \qquad \forall\, p \in P : p > 1 \qquad (169)$$

Expression (151) states the minimization objectives (cycle time and line length). Constraint (152) states that every task must be assigned to a station. Precedence relations between tasks are imposed by Constraint (153). Constraints (154) and (155) define the basic requirements of mixed-model sequencing. Constraint (156) combines balancing and sequencing to determine processing times of each product at each station. Equation (157) is a cut that improves the model's linear relaxation. The following constraints describe the basic cyclical scheduling rules: Products must enter each station before processing can start (Constraint (158)); they must be fully processed (Equation (159)) before entering the next station (Constraint (160)) and before the next product can start being processed (Constraint (161)) at the current one. Constraint (162) ties the time between the first and last products to the line's cycle time. Products are also scheduled in terms of their position in the line. Here, it is usually assumed that the line speed V equals one length unit per time unit. This means that L also denotes the time it takes for a product to move through the line, i.e. the system's flow-time or lead-time. Equations (163) and (164) measure the positions in which products enter and complete processing at each station, respectively. These positions are bounded by each station's boundaries, as stated by Constraints (165) and (166). Finally, Equation (167) defines the last boundary of the last station as the line length. The two last constraints reflect practical concerns: Stations are not allowed to overlap (Constraint (168)), and products are evenly spaced (Constraint (169)).

10.5 AN ILLUSTRATIVE EXAMPLE

The following example demonstrates the possible multi-objective nature of the relationship between line length and cycle time. It also allows the discussion of how to take into account line speed in a simplified context. Consider the problem data presented by Table 39 and an assembly line with four workstations.

Table 39 – Illustrative example: Task durations and precedence relations

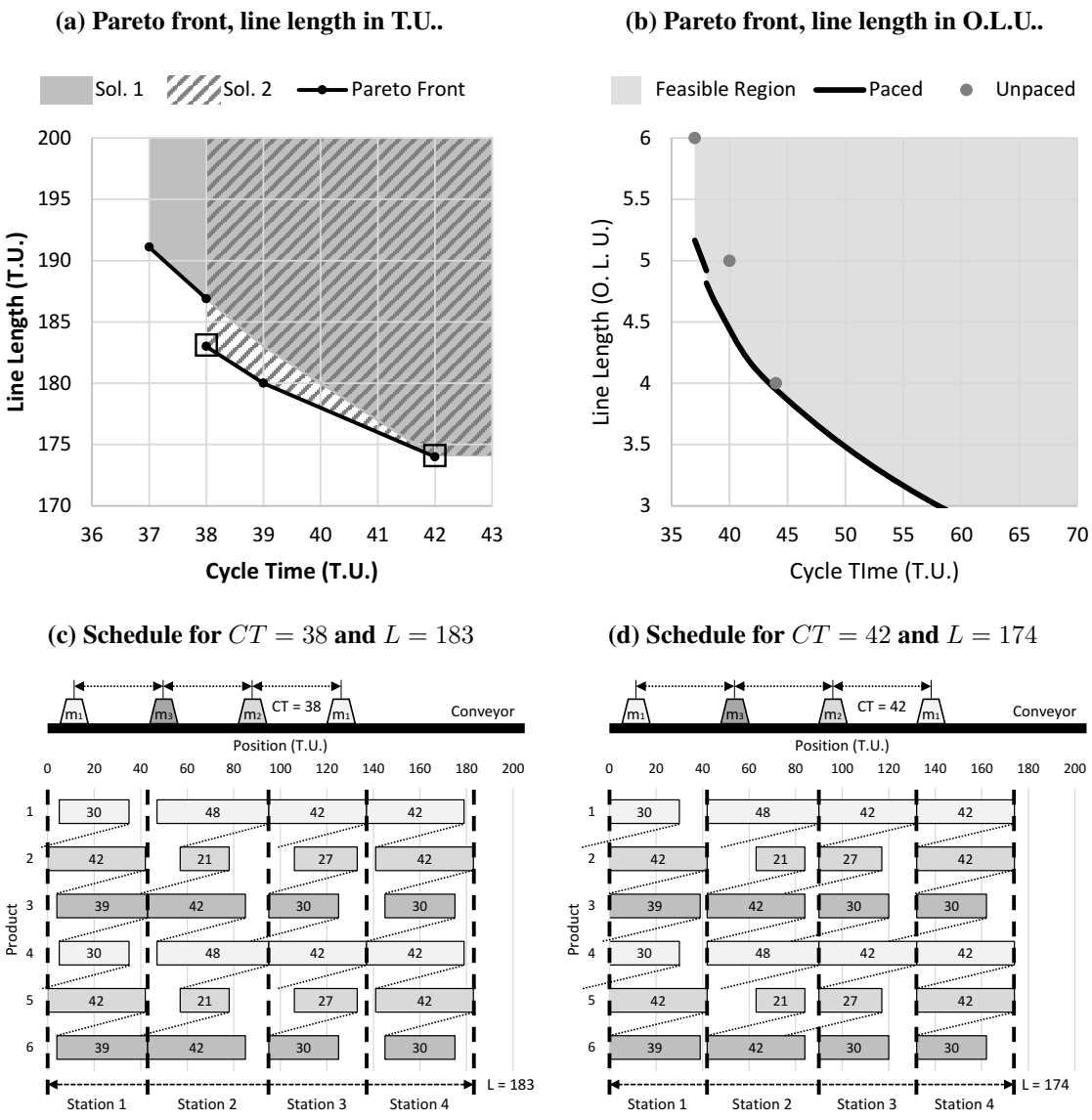
Task	1	2	3	4	5	6	7
Model 1	21	9	36	12	42	0	42
Model 2	42	0	21	0	27	0	42
Model 3	39	0	42	0	30	12	18
Predecessor	-	1	1	3	3	5	5

Figure 43(a) presents the Pareto front resulting from applying this Chapter's proposed method to the model described in Section 10.4. Two of its solutions (highlighted with squares in Figure 43(a)) are represented in detail by Figures 43(c) and 43(d): in them, the vertical dashed

lines represent station boundaries, each horizontal line represents a product being manufactured, the numbers within blocks represent their processing time, the blocks' colors represent the product model, and the diagonal dotted lines represent the worker's upstream movement after finishing work on a product.

Notice that the size and sequence of processing time blocks is the same in both Figure 43(c) and 43(d). This happens because the solutions share all binary decisions (task assignments and product sequence) but differ in terms of their continuous (scheduling) decision variables, leading to different performance in both objectives. This confirms Chapter 8's assertion that mixed-integer linear multi-objective disputes can occur between line length and cycle time.

Figure 43 – Illustrative example: Pareto front and solutions

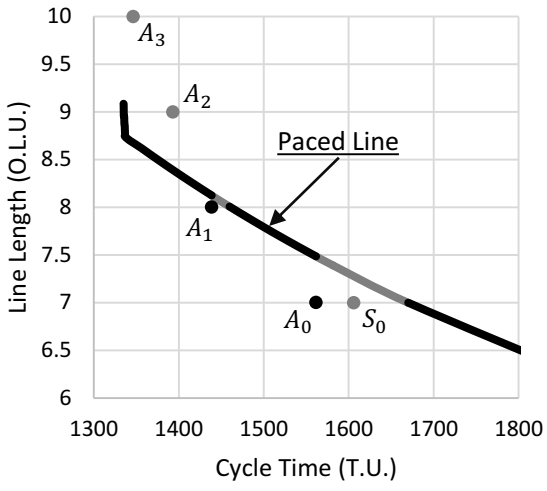


Finally, the model described in Section 10.4 effectively describes line length in Time Units (T.U.) which is a common simplification that follows from considering that the line speed is constant (DEFERSHA; MOHEBALIZADEHGASHTI, 2018). However, this obscures comparisons to unpaced lines, whose lengths are measures in discrete terms (i.e. 4 stations). In order to make "apples-to-apples" comparisons to lines of discrete pace, the line length measured in T.U. can be divided by the cycle time, defining a length metric in Operational Length Units (O.L.U.), i.e. one that relates to the size of the product manufactured in the line. A similar strategy was applied in Chapter 9 regarding line length costs.

10.6 REVISITING A CASE STUDY

The method presented in Section 10.3 is here applied to practical data from the assembly line previously studied in Chapter 2. The main research question is: how does the current line control (Asynchronous unpaced) compare to a continuous paced alternative? In this case study, the number of workstations is fixed as seven for both line types. Therefore, the optimization objectives are to minimize cycle time and internal storage requirements. As demonstrated in Section 10.5, these objectives can naturally conflict, defining a bi-objective problem. Furthermore, in the case of paced lines, fractional values for both objectives are admissible. By normalizing the paced line length as Operational Length Units (O.L.U), as described in Section 10.5, the resulting Pareto front for this practical case is presented in Figure 44. The performance of (unbuffered) synchronous and (buffered) asynchronous lines are denoted by S_0 and A_b (b states the number of buffers). These were obtained using the MILP models described in Chapters 3 and 5.

Figure 44 – Practical Case's global Pareto front for both line control types



Notice that each line type is dominant for a specific range of cycle time values. In this particular case, the asynchronous line control is efficient in the 1438-1460 Time Units (T.U.) and 1560-1670 T.U. ranges. The paced line control is efficient for the remaining ones. Furthermore, for both line controls, the cycle time reduction offered by increased internal storage is subject to diminishing returns. However, these seem to be less penalized for paced lines than for its unpaced counterparts. One explanation for this fact is that internal storage can be continuously distributed for paced lines, but only discretely distributed for asynchronous ones, i.e. there cannot be fractional buffers between stations. Lastly, paced lines allow arbitrary increases to CT to reduce line length (measured in O.L.U.) to less than seven (the number of stations). This normally represents too high cycle time values, but is an useful flexibility in cases with lower demand and in which line length is extremely costly.

10.7 CONCLUSIONS AND PERSPECTIVES

The newly introduced method allows to optimally solve multi-objective mixed-integer linear programming problems with part-wise continuous Pareto fronts. This is achieved by solving a finite number of single objective linear and mixed-integer linear problems. The former are employed to obtain the Pareto front of the residual problem defined by fixing the integer variables $\mathbf{x}$, and the later is used to verify whether incumbent segments in the objective space are part of the true Pareto front. By doing so, the proposed method converts the exact solution of this class of multi-objective MILP problems to multiple, but finitely many, solutions single-objective MILP instances of similar size.

This method addresses some of the major drawbacks of common alternatives that are based on reductions to single objective problems: the weighted sum and ε-constrained methods. The former tends not to find all efficient solutions and the latter might find the same solution multiple times without being able to guarantee that the Pareto front has been fully defined. Assuming the instance is tractable, the proposed method is guaranteed to require a finite number of single-objective MILP model solutions to find the full Pareto front.

Applying the proposed method to industrial data, lead to a few inferences: First, the multi-objective dispute between line lenght and cycle time is confirmed. Second, it is possible to compare the performance of different line paces using that framework: in the case study, paced lines offered more efficient compromises between space and throughput for lower ranges of cycle time, and were outperformed by their unpaced counterparts in the higher ranges.

11 CONCLUDING REMARKS

This chapter highlights the thesis main contributions and presents its final considerations, i.e. specific comments on the thesis objectives, listed in Chapter 1. Lastly, this chapter lists possible directions for further works presented in the publications that based this thesis. Some of the originally suggested directions have already been followed and became chapters in this document. Others are suggestions as well as ongoing works that have not yet been published.

11.1 SUMMARY OF CONTRIBUTIONS

This thesis addressed variants of the assembly line balancing problem with cyclical scheduling aspects and proposed mathematical models and solution methods to address them. Chapter 2 presents a mathematical formulation to balance mixed-model asynchronous assembly lines under a cyclical product sequence. This formulation is extended by the following chapters to incorporate product sequencing and other line paces (Chapter 3), parallel workstations (Chapter 4) and finally buffer allocation (Chapter 5). Chapter 5 is also the first one to depart from solving a mathematical model to proposing a solution method - an iterative decomposition. Chapter 6 furthers that departure by presenting a solution method independent of mathematical models (a simheuristic) to solve a stochastic variant of the problem. In it, specific contributions to the bowl phenomenon and a fast algorithm for asynchronous line simulation are also presented. Chapter 7 presents a contribution for the multi-manned assembly line balancing literature in the form of corrections to a previous paper. Chapter 8 introduces a paradigm shift that allows significantly shorter multi-manned lines. Chapter 8 also describes a potential multi-objective dispute between line length and cycle time, which can be both continuous and discrete. Chapter 9 investigates fractional task allocations in assembly lines and describes the internal storage costs required to achieve the cycle time reductions that they offer. Chapter 10 presents a method to solve general mixed-integer linear multi-objective problems and shows how the balancing-sequencing of paced lines define such a problem - which was shown by Chapter 8.

In multiple chapters, the proposed formulations and methods are compared to literature benchmarks and shown to significantly outperform them, both in the deterministic problem classes (Chapter 2 and 3) and in stochastic contexts (Chapter 6). Chapter 4 offers a particularly strong comparison as the tested instances were introduced by the benchmark article itself.

Chapters 2-5 can be seen as continuations of one another - gradually incorporating more decision variables and features in the formulation. Chapter 6 discusses how to balance lines from Chapter 2 when product sequence is beyond the decision maker's control. Chapters 7-9 are somewhat independent of the previous five, as they address different problem variants. However, the multi-objective relationship between line length and cycle time is shown to also occur for the balancing-sequencing class of problems studied in Chapter 10.

Multiple Operational Research techniques were employed in this thesis. The most common throughout its chapters is mathematical modelling with mixed-integer linear programming. However, other solution methods were also applied: Chapter 5 presents an iterative decomposition; Chapter 6 presents a simheuristic; Chapter 7 and 8 present hierarchical decompositions; Lastly, Chapter 10 presents a criterion-space multi-objective method. Simulations were also applied (Chapters 2 and 6), as well as graph-based lower bound algorithms (Chapter 8), and complexity proof by reduction (Chapter 9). During his doctorate, the author also collaborated in works that applied combinatorial Bender's Cuts and graph-based feasibility cuts/valid constraints (MICHELS *et al.*, 2019; MICHELS *et al.*, 2020).

11.2 FINAL CONSIDERATIONS

This section refers back to the thesis' objectives listed in Section 1.2. In short, these were: (1) to develop and (2) adapt mathematical models to describe problem variants; (3) to describe the role of line control; and (4) to develop efficient solution methods. The following paragraphs highlight how the thesis' contributions accomplish these objectives.

Regarding the objectives of developing and adapting mathematical models, this thesis: presents a mixed-integer programming model tying mixed-model line balancing to cyclical scheduling and extends it to incorporate mixed-model sequencing, parallel stations and buffer allocation. Mathematical models are also presented for multi-manned lines, tied to two solution methods and a paradigm shift regarding paced line controls. A mathematical formulation is also presented in the context of fractional task allocations, in which cyclical scheduling allows a description of the internal storage costs tied to this additional flexibility for line balancing. Lastly, the original mixed-model line balancing formulation is adapted to describe paced lines in a bi-objective context.

Regarding the objective of describing the role of line control (i.e. type of conveyor system), this thesis reached several conclusions: the distinction between synchronous and asyn-

chronous ones was mathematically described and tied to a trade-off between investment cost and line efficiency: Asynchronous lines are more costly, but can be more efficient and also allow buffers to play a more important role. Comparisons between paced and unpaced lines are more delicate and were first addressed in this thesis in the context of multi-manned lines: paced line controls allow significantly shorter multi-manned lines lengths than their unpaced counterparts, but require a more flexible definition for station boundaries. A possible mixed-integer linear multi-objective dispute between cycle time and line length was shown in this particular context. This dispute was shown to also happen in paced mixed-model lines. Chapter 10's case study suggests that paced lines outperform unpaced lines for lower ranges of cycle time, and that paced lines are outperformed in higher cycle time ranges by unbuffered asynchronous lines.

Regarding the objective of developing solution methods, those presented in this thesis have ranged from solving mathematical programming models to decompositions and a simheuristic. It is apparent that the cyclical scheduling component adds significant computational challenges to these problems, specially when linked to decision variables involving Big-M constraint relaxations. Hence, the solution methods appear to be more efficient the more the scheduling (e.g. sequencing) decisions can be hierarchically separated from the balancing decisions, while still maintaining a higher order model capable of estimating performance without them. In a particular stochastic case, the simheuristic presented in Chapter 6 applies that concept by using and algorithmic simulator only for balancing solutions, whose lower bound on performance is better than a reference solution in the local search algorithm. While this hierarchical separation may be desirable, it may not always be possible, as shown in the multi-objective dispute between line length and cycle time: there, each balancing-sequencing solution is tied to a whole Pareto front of efficient values of cycle time and line length, making the tie between the line balancing and cyclical scheduling decision variables rather inextricable.

11.3 FURTHER WORKS

Out of the possible extensions of the formulation presented in Chapter 2, multi-model and U-shaped lines remain the most significant unexplored opportunities. Chapter 3's discussion regarding synchronous and asynchronous line controls can be extended by incorporating economic pricing of transport systems. Further works can apply Chapter 4's boundary-wise formulation to other scheduling problems with parallelism such as job-shops and flow-shops. Chapter 5's combination of degrees of freedom lead to major performance challenges regarding

lower bounds. Hence, stronger lower bounds are very desirable for that class of problems. The contributions described in Chapter 6 can be enhanced by extending the proposed Cycle Time Simulator to include other assembly line features. Another direction lies in employing exact performance evaluations, such as the Markov-Chain formulation described in Lopes *et al.* (2019), and analytical lower bounds to produce an exact method for this stochastic problem. Chapter 7 was mainly a correction to a previous paper, but its proposed method has already been employed as a heuristic warm-start to solve a related multi-manned balancing problem (MICHELS *et al.*, 2020). Chapter 8's flexible multi-manned line formulation can be enhanced by incorporating space constraints in order to better describe potential collisions of workers and how to avoid them. Regarding Chapter 9's fractional task allocations, proposed directions for further works include adapting or developing effective (meta)-heuristic or exact solution methods for the problem. Furthermore, more research can investigate adequate throughput performance metrics and the explicit incorporation of mixed-model sequencing to lines that allow fractional allocations. Lastly, further works should thoroughly compare Chapter 10's multi-objective method to previously described strategies, as well as generalize it for problems with more objectives.

REFERENCES

AHN, Hyun-soo; RIGHTER, Rhonda. Dynamic load balancing with flexible workers. **Advances in Applied Probability**, n. September, p. 621–642, 2006.

AKPINAR, Sener; BAYKASOGLU, Adil. Modeling and solving mixed-model assembly line balancing problem with setups. Part I: A mixed integer linear programming model. **Journal of Manufacturing Systems**, v. 33, p. 177–187, 2014.

AKPINAR, Sener; BAYKASOGLU, Adil. Modeling and solving mixed-model assembly line balancing problem with setups. Part II: A multiple colony hybrid bees algorithm. **Journal of Manufacturing Systems**, v. 33, n. 4, p. 445–461, 2014.

AKPINAR, Sener; ELMI, Atabak; BEKTAS, Tolga. Combinatorial Benders cuts for assembly line balancing problems with setups. **European Journal of Operational Research**, v. 259, n. 2, p. 527–537, 2017.

ALEXANDER, David R.; PREMACHANDRA, I. M.; KIMURA, Toshikazu. Transient and asymptotic behavior of synchronization processes in assembly-like queues. **Annals of Operations Research**, v. 181, n. 1, p. 641–659, 2010.

ALGHAZI, Anas; KURZ, Mary E. Mixed model line balancing with parallel stations, zoning constraints, and ergonomics. **Constraints**, Constraints, v. 23, p. 123–153, 2018.

ÁLVAREZ-MIRANDA, Eduardo; CHACE, Sebastián; PEREIRA, Jordi. Assembly line balancing with parallel workstations. **International Journal of Production Research**, 2020.

ÁLVAREZ-MIRANDA, Eduardo; PEREIRA, Jordi. On the complexity of assembly line balancing problems. **Computers and Operations Research**, v. 108, p. 182–186, 2019.

ANUAR, Rouie; BUKCHIN, Yossi. Design and operation of dynamic assembly lines using work-sharing. **International Journal of Production Research**, v. 44, n. 18-19, p. 4043–4065, 2006.

BARD, Jonathan F.; DAR-ELJ, Ezey; SHTUB, Avraham. An analytic framework for sequencing mixed model assembly lines. **International Journal of Production Research**, v. 30, n. 1, p. 35–48, 1992.

BARTHOLDI, J. J. Balancing two-sided assembly lines: A case study. **International Journal of Production Research**, v. 31, n. 10, p. 2447–2461, 1993.

BATTAÏA, Olga; DOLGUI, Alexandre. A taxonomy of line balancing problems and their solution approaches. **International Journal of Production Economics**, v. 142, n. 2, p. 259–277, 2013.

BATTINI, Daria; FACCIO, Maurizio; PERSONA, Alessandro; SGARBOSSA, Fabio. Balancing - sequencing procedure for a mixed model assembly system in case of finite buffer capacity. **International Journal of Advanced Manufacturing Technology**, v. 44, n. 3-4, p. 345–359, 2009.

BAYBARS, Ilker. Survey of exact algorithms for the simple assembly line balancing problem. **Management Science**, v. 32, n. 8, p. 909–932, 1986.

BECKER, Christian; SCHOLL, Armin. A survey on problems and methods in generalized assembly line balancing. **European Journal of Operational Research**, v. 168, n. 3, p. 694–715, 2006.

BECKER, Christian; SCHOLL, Armin. Balancing assembly lines with variable parallel workplaces: Problem definition and effective solution procedure. **European Journal of Operational Research**, Elsevier B.V., v. 199, n. 2, p. 359–374, 2009.

BIELE, Alexander; MÖNCH, Lars. Hybrid approaches to optimize mixed-model assembly lines in low-volume manufacturing. **Journal of Heuristics**, Springer US, v. 24, n. 1, p. 49–81, 2018.

BOYSEN, Nils; FLIEDNER, Malte; SCHOLL, Armin. A classification of assembly line balancing problems. **European Journal of Operational Research**, v. 183, n. 2, p. 674–693, 2007.

BOYSEN, Nils; FLIEDNER, Malte; SCHOLL, Armin. Assembly line balancing: Which model to use when? **International Journal of Production Economics**, v. 111, n. 2, p. 509–528, 2008.

BOYSEN, Nils; FLIEDNER, Malte; SCHOLL, Armin. Assembly line balancing: Joint precedence graphs under high product variety. **IIE Transactions**, v. 41, n. 3, p. 183–193, 2009.

BOYSEN, Nils; FLIEDNER, Malte; SCHOLL, Armin. Production planning of mixed-model assembly lines: Overview and extensions. **Production Planning and Control: The Management of Operations**, v. 20, n. 5, p. 455–471, 2009.

BOYSEN, Nils; FLIEDNER, Malte; SCHOLL, Armin. Sequencing mixed-model assembly lines: Survey, classification and model critique. **European Journal of Operational Research**, v. 192, n. 2, p. 349–373, 2009.

BOZEJKO, Wojciech; UCHROŃSKI, Mariusz; WODECKI, Mieczysław. Parallel metaheuristics for the cyclic flow shop scheduling problem. **Computers and Industrial Engineering**, Elsevier Ltd, v. 95, p. 156–163, 2016.

BRAUNER, Nadia. Identical part production in cyclic robotic cells: Concepts, overview and open questions. **Discrete Applied Mathematics**, v. 156, n. 13, p. 2480–2492, 2008.

BRUCKER, Peter; KAMPMEYER, Thomas. A general model for cyclic machine scheduling problems. **Discrete Applied Mathematics**, v. 156, n. 13, p. 2561–2572, 2008.

BUKCHIN, Joseph. A comparative study of performance measures for throughput of a mixed model assembly line in a JIT environment. **International Journal of Production Research**, v. 36, n. 10, p. 2669–2685, 1998.

BUKCHIN, Joseph; DAR-EL, Ezey M; RUBINOVITZ, Jacob. Mixed model assembly line design in a make-to-order environment. **Computers and Industrial Engineering**, v. 41, p. 405–421, 2002.

BUKCHIN, Joseph; RUBINOVITZ, Jacob. A weighted approach for assembly line design with station paralleling and equipment selection. **IIE Transactions**, v. 35, n. 1, p. 73–85, 2003.

BUKCHIN, Yossi; SOFER, Tal. Bi-directional work sharing in assembly lines with strict and flexible assembly sequences. **International Journal of Production Research**, v. 49, n. 14, p. 4377–4395, 2011.

BUKCHIN, Yossi; WEXLER, Efrat. The Effect of Buffers and Work-Sharing on Makespan Improvement of Small Batches in Assembly Lines under Learning Effects. **IIE Transactions**, v. 48, n. 5, p. 403–414, 2015.

CHANG, Horng-Jinh; CHANG, Tung-Meng. Simultaneous Perspective-Based Mixed-Model Assembly Line Balancing Problem. **Tamkang Journal of Science and Engineering**, v. 13, n. 3, p. 327–336, 2010.

CHEN, Jiaqiong; ASKIN, Ronald G. Throughput maximization in serial production lines with worksharing. **International Journal of Production Economics**, v. 99, p. 88–101, 2006.

CHEN, Yin Yann. A hybrid algorithm for allocating tasks, operators, and workstations in multi-manned assembly lines. **Journal of Manufacturing Systems**, The Society of Manufacturing Engineers, v. 42, p. 196–209, 2017.

CHIANG, Wen-Chyuan Chyuan; URBAN, Timothy L.; XU, Xiaojing. A bi-objective metaheuristic approach to unpaced synchronous production line-balancing problems. **International Journal of Production Research**, v. 50, n. 1, p. 293–306, 2012.

CHICA, Manuel; BAUTISTA, Joaquín; CORDÓN, Óscar; DAMAS, Sergio. A multiobjective model and evolutionary algorithms for robust time and space assembly line balancing under uncertain demand. **Omega**, Elsevier, v. 58, p. 55–68, 2016.

CHICA, Manuel; CORDÓN, Óscar; DAMAS, Sergio; BAUTISTA, Joaquín. Multiobjective constructive heuristics for the 1/3 variant of the time and space assembly line balancing problem: ACO and random greedy search. **Information Sciences**, Elsevier Inc., v. 180, n. 18, p. 3465–3487, 2010.

CHICA, Manuel; JUAN, Angel A.; CORDON, Oscar; KELTON, David. Why Simheuristics? Benefits, Limitations, and Best Practices When Combining Metaheuristics with Simulation. **SSRN**, 2017. Available at: https://doi.org/10.2139/ssrn.2919208.

CODATO, Gianni; FISCHETTI, Matteo. Combinatorial Benders' Cuts for Mixed-Integer Linear Programming. **Operations Research**, v. 54, n. 4, p. 756–766, 2006.

COROMINAS, Albert; PASTOR, Rafael; PLANS, Joan. Balancing assembly line with skilled and unskilled workers. **Omega**, v. 36, n. 6, p. 1126–1132, 2008.

DAGKAKIS, Georgios; ROTONDO, Anna; HEAVEY, Cathal. Embedding optimization with deterministic discrete event simulation for assignment of cross-trained operators : An assembly line case study. **Computers and Operations Research**, Elsevier Ltd, v. 111, p. 99–115, 2019.

DAR-EL, E M; HERER, Y T; MASIN, M. CONWIP-based production lines with multiple bottlenecks:performance and design implications. **IIE Transactions**, v. 31, p. 99–111, 1999.

DECKER, Maria. Capacity smoothing and sequencing for mixed-model lines. **International Journal of Production Economics**, v. 30-31, p. 31–42, 1993.

DEFERSHA, Fantahun M; MOHEBALIZADEHGASHTI, Fatemeh. Simultaneous balancing, sequencing, and workstation planning for a mixed model manual assembly line using hybrid genetic algorithm. **Computers and Industrial Engineering**, Elsevier, v. 119, p. 370–387, 2018.

DELICE, Yilmaz; AYDOGAN, Emel Kizilkaya; ÖZCAN, Ugur; ILKAY, Mehmet Sitki. A modified particle swarm optimization algorithm to mixed-model two-sided assembly line balancing. **Journal of Intelligent Manufacturing**, v. 28, p. 23–36, 2017.

DEMIR, Leyla; TUNALI, Semra; ELIIYI, Deniz Tursel. The state of the art on buffer allocation problem: a comprehensive survey. **Journal of Intelligent Manufacturing**, v. 25, n. 3, p. 371–392, 2014.

DIMITRIADIS, Sotirios G. Assembly line balancing and group working: A heuristic procedure for workers' groups operating on the same product and workstation. **Computers and Operations Research**, v. 33, p. 2757–2774, 2006.

DINECHIN, Benoît Dupont de; KORDON, Alix Munier. Converging to periodic schedules for cyclic scheduling problems with resources and deadlines. **Computers and Operations Research**, Elsevier, v. 51, p. 227–236, 2014.

DOERR, Kenneth H.; KLASTORIN, Theodore D.; MAGAZINE, Michael J. Synchronous Unpaced Flow Lines with Worker Differences and Overtime Cost. **Management Science**, v. 46, n. 3, p. 421–435, 2000.

DONG, Jietao; ZHANG, Linxuan; XIAO, Tianyuan. A hybrid PSO / SA algorithm for bi-criteria stochastic line balancing with flexible task times and zoning constraints. **Journal of Intelligent Manufacturing**, Springer US, v. 29, n. 4, p. 737–751, 2018.

ECKER, J G; KOUADA, I A. Finding all efficient extreme points for multiple objective linear programs. **Mathematical Programming**, v. 14, p. 249–261, 1978.

ELMI, Atabak; TOPALOGLU, Seyda. Cyclic job shop robotic cell scheduling problem: Ant colony optimization. **Computers and Industrial Engineering**, v. 111, p. 417–432, 2017.

FATTAHI, Parviz; ROSHANI, Abdolreza; ROSHANI, Abdolhassan. A mathematical model and ant colony algorithm for multi-manned assembly line balancing problem. **International Journal of Advanced Manufacturing Technology**, v. 53, p. 363–378, 2011.

GAREY, Michael R; JOHNSON, David S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. *[S.l.]*: W. H. Freeman, 1979. 338 p. ISBN 0-7167-1045-5.

GLOVER, Fred. Future Paths for Integer Programming and Links to Artificial Intelligence. **Computers and Operations Research**, v. 13, n. 5, p. 533–549, 1986.

GÖKÇEN, Hadi; EREL, Erdal. A goal programming approach to mixed-model assembly line balancing problem. **International Journal of Production Economics**, v. 48, p. 177–185, 1997.

GOLLE, Uli; ROTHLAUF, Franz; BOYSEN, Nils. Iterative beam search for car sequencing. **Annals of Operations Research**, v. 226, p. 239–254, 2015.

GONZALEZ-NEIRA, Eliana Maria; FERONE, Daniele; HATAMI, Sara; JUAN, Angel A. A biased-randomized simheuristic for the distributed assembly permutation flowshop problem with stochastic processing times. **Simulation Modelling Practice and Theory**, Elsevier B.V., v. 79, p. 23–36, 2017.

GRZECHCA, W; FOULDS, L R. The Assembly Line Balancing Problem with Task Splitting: A Case Study. **IFAC-PapersOnLine**, Elsevier Ltd., v. 48, n. 3, p. 2002–2008, 2015. Available at: http://dx.doi.org/10.1016/j.ifacol.2015.06.382.

GUIMARANS, Daniel; DOMINGUEZ, Oscar; PANADERO, Javier; JUAN, Angel A. A simheuristic approach for the two-dimensional vehicle routing problem with stochastic travel times. **Simulation Modelling Practice and Theory**, Elsevier B.V., v. 89, p. 1–14, 2018.

GULTEKIN, Hakan. Scheduling in flowshops with flexible operations: Throughput optimization and benefits of flexibility. **International Journal of Production Research**, Elsevier, v. 140, n. 2, p. 900–911, 2012.

GUREVSKY, Evgeny; BATTAÏA, Olga; DOLGUI, Alexandre. Balancing of simple assembly lines under variations of task processing times. **Annals of Operations Research**, v. 201, n. 1, p. 265–286, 2012.

GURGUR, Cigdem Z. Optimal configuration of a decentralized, market-driven production/inventory system. **Annals of Operations Research**, v. 209, n. 1, p. 139–157, 2013.

HAMZADAYI, Alper; YILDIZ, Gokalp. A genetic algorithm based approach for simultaneously balancing and sequencing of mixed-model U-lines with parallel workstations and zoning constraints. **Computers and Industrial Engineering**, Elsevier Ltd, v. 62, n. 1, p. 206–215, 2012.

HAMZADAYI, Alper; YILDIZ, Gokalp. A simulated annealing algorithm based approach for balancing and sequencing of mixed-model U-lines. **Computers and Industrial Engineering**, Elsevier Ltd, v. 66, n. 4, p. 1070–1084, 2013.

HANEN, Claire; MUNIER, Alix. Cyclic scheduling on parallel processors: an overview. *In*: CHRETIENNE, P; COFFMAN, E G; LENSTRA, J K; LIU, Z (Ed.). **Scheduling Theory and Its Applications**. *[S.l.]*: John Wiley & Sons LTd, 1994. chap. 4. ISBN 0166-218X.

HATAMI, Sara; CALVET, Laura; FERNÁNDEZ-VIAGAS, Victor; FRAMIÑÁN, José M; JUAN, Angel A. A simheuristic algorithm to set up starting times in the stochastic parallel flowshop problem. **Simulation Modelling Practice and Theory**, Elsevier, v. 86, p. 55–71, 2018.

HAYES, Brian. The Easiest Hard Problem. **American Scientist**, v. 90, n. 2, p. 113–117, 2002.

HEATH, Susan K.; BARD, Jonathan F.; MORRICE, Douglas J. A GRASP for simultaneously assigning and sequencing product families on flexible assembly lines. **Annals of Operations Research**, v. 203, n. 1, p. 295–323, 2013.

HELBER, Stefan; SAHLING, Florian. A fix-and-optimize approach for the multi-level capacitated lot sizing problem. **International Journal of Production Economics**, Elsevier, v. 123, n. 2, p. 247–256, 2010.

HILLIER, Frederick S.; BOILING, Ronald W. On the Optimal Allocation of Work in Symmetrically Unbalanced Production Line Systems with Variable Operation Times. **Management Science**, v. 25, n. 8, p. 721–728, 1979.

HILLIER, Frederick S.; LIEBERMAN, Gerald J. **Introduction to Operational Research**. 10. ed. New York, NY: McGrawHill, 2015. 1010 p. ISBN 978-0-07-352345-3.

HILLIER, Frederick S.; SO, Kut C. The Effect of the Coefficient of Variation of Operation Times on the Allocation of Storage Space in Production Line Systems. **IIE Transactions**, v. 23, n. 2, p. 198–206, 1991.

HILLIER, Frederick S; SO, Kut C. On the robustness of the bowl phenomenon. **European Journal of Operational Research**, v. 89, p. 496–515, 1996.

IGNIZIO, James P. An Algorithm for Solving the Linear Goal Programming Problem by Solving its Dual. **Journal of Operational Research Society**, v. 36, n. 6, p. 507–515, 1985.

JEONG, In-jae; JEON, Sumin. Balanceability of a work-sharing line using floating workers and its comparison with floating work strategy. **International Journal of Production Research**, Taylor & Francis, 2020. Available at: https://doi.org/10.1080/00207543.2020.1795291.

JOHNSON, Roger V. A Branch and Bound Algorithm for Assembly Line Balancing Problems with Formulation Irregularities. **Management Science**, v. 29, n. 11, p. 1309–1324, 1983.

JUAN, Angel A; FAULIN, Javier; GRASMAN, Scott E; RABE, Markus; FIGUEIRA, Gonçalo. A review of simheuristics: Extending metaheuristics to deal with stochastic combinatorial optimization problems. **Operations Research Perspectives**, Elsevier Ltd, v. 2, p. 62–72, 2015.

KARABATI, Selçuk; KOUVELIS, Panagiotis. The interface of buffer design and cyclic scheduling decisions in deterministic flow lines. **Annals of Operations Research**, v. 50, n. 1, p. 295–317, 1994.

KARABATI, Selçuk; KOUVELIS, Panagiotis. Cyclic Scheduling in Flow Lines: Modeling Observations, Effective Heuristics and a Cycle Time Minimization Procedure. **Naval Research Logistics**, v. 43, n. 5, p. 211–231, 1996.

KARABATI, Selçuk; KOUVELIS, Panagiotis; KIRAN, Ali S. Games , Critical Paths and Assignment Problems in Permutation Flow Shops and Cyclic Scheduling Flow Line Environments. **Journal of Operational Research Society**, v. 43, n. 3, p. 241–258, 1992.

KARABATI, Selçuk; SAYIN, Serpil. Assembly line balancing in a mixed-model sequencing environment with synchronous transfers. **European Journal of Operational Research**, v. 149, n. 2, p. 417–429, 2003.

KAZEMI, Abolfazl; SEDIGHI, Abdolhossein. A cost-oriented model for balancing mixed-model assembly lines with multi-manned workstations. **International Journal of Services and Operations Management**, v. 16, n. 3, p. 289–309, 2013.

KELLEGÖZ, Talip. Assembly line balancing problems with multi-manned stations: a new mathematical formulation and Gantt based heuristic method. **Annals of Operations Research**, Springer US, v. 253, n. 1, p. 377–404, 2017.

KELLEGÖZ, Talip; TOKLU, Bilal. An efficient branch and bound algorithm for assembly line balancing problems with parallel multi-manned workstations. **Computers and Operations Research**, v. 39, p. 3344–3360, 2012.

KELLEGÖZ, Talip; TOKLU, Bilal. A priority rule-based constructive heuristic and an improvement method for balancing assembly lines with parallel multi-manned workstations. **International Journal of Production Research**, v. 53, n. 3, p. 736–756, 2015.

KELLER, André A. **Multi-Objective Optimization in Theory and Practice: I. Classical Methods.** *[S.l.]*: Bentham Science Publishers Ltd, 2017. (Multi-objective Optimization in Theory and Practice, I, Classical methods). ISBN 9781681085692.

KIM, Yeongho Keun; KIM, Jae Yun; KIM, Yeongho Keun. A Coevolutionary Algorithm for Balancing and Sequencing in Mixed Model Assembly Lines. **Applied Intelligence**, v. 13, n. 3, p. 247–258, 2000.

KIM, Yeo Keun; KIM, Jae Yun; KIM, Yeongho Keun. An endosymbiotic evolutionary algorithm for the integration of balancing and sequencing in mixed-model U-lines. **European Journal of Operational Research**, v. 168, n. 3, p. 838–852, 2006.

KIM, Yeo Keun; KIM, Sun Jin; KIM, Jae Yun. Balancing and sequencing mixed-model U-lines with a co-evolutionary algorithm. **Production Planning and Control: The Management of Operations**, v. 11, n. 8, p. 754–764, 2000.

KOENIGSBERG, Ernest. Production Lines and Internal Storage – A Review. **Management Science**, v. 5, n. 4, p. 410–433, 1959.

KOUVELIS, Panos Panagiotis; KARABATI, Selcuk Selçuk. Cyclic scheduling in synchronous production lines. **IIE Transactions**, v. 31, n. 8, p. 709–719, 1999.

KUCUKKOC, Ibrahim; ZHANG, David Z. Mathematical model and agent based solution approach for the simultaneous balancing and sequencing of mixed-model parallel two-sided assembly lines. **International Journal of Production Economics**, Elsevier, v. 158, p. 314–333, 2014.

KUCUKKOC, Ibrahim; ZHANG, David Z. Balancing of parallel U-shaped assembly lines. **Computers and Operations Research**, Elsevier, v. 64, p. 233–244, 2015.

KUCUKKOC, Ibrahim; ZHANG, David Z. Integrating ant colony and genetic algorithms in the balancing and scheduling of complex assembly lines. **International Journal of Advanced Manufacturing Technology**, v. 82, p. 265–285, 2016.

LEU, Y Y; HUANG, P Y; RUSSELL, R S. Using beam search techniques for sequencing mixed-model assembly lines. **Annals of Operations Research**, v. 70, p. 379–397, 1997.

LEVNER, Eugene; KATS, Vladimir; PABLO, David Alcaide López de; CHENG, T. C. E. Complexity of cyclic scheduling problems: A state-of-the-art survey. **Computers and Industrial Engineering**, Elsevier Ltd, v. 59, n. 2, p. 352–361, 2010.

LI, Zixiang; JANARDHANAN, Mukund Nilakantan; TANG, Qiuhua; NIELSEN, Peter. Mathematical model and metaheuristics for simultaneous balancing and sequencing of a robotic mixed-model assembly line. **Engineering Optimization**, Taylor & Francis, v. 50, n. 5, p. 877–893, 2017.

LOPES, Thiago Cantos; BRAUNER, Nadia; MAGATÃO, Leandro. Assembly Line Balancing with Fractional Task Allocations. **International Journal of Production Research**, 2021. Available at: https://doi.org/10.1080/00207543.2020.1866224.

LOPES, Thiago Cantos; BRAUNER, Nadia; MAGATãO, Leandro. Optimally solving multi-objective MILP problems with part-wise continuous Pareto fronts. *In*: **Proceedings of the XXI ROADEF**. Montpellier: *[s.n.]*, 2020. Available at: https://easychair.org/publications/preprint/PJK8.

LOPES, Thiago Cantos; MICHELS, Adalberto Sato; LÜDERS, Ricardo; MAGATÃO, Leandro. A simheuristic approach for throughput maximization of asynchronous buffered stochastic mixed-model assembly lines. **Computers and Operations Research**, Elsevier Ltd, v. 115, p. 104863, 2020.

LOPES, Thiago Cantos; MICHELS, Adalberto Sato; MAGATÃO, Leandro. A note to: A hybrid algorithm for allocating tasks, operators, and workstations in multi-manned assembly lines. **Journal of Manufacturing Systems**, Elsevier, v. 52, p. 205–208, 2019.

LOPES, Thiago Cantos; MICHELS, Adalberto Sato; SIKORA, Celso Gustavo Stall; MOLINA, Rafael Gobbi; MAGATÃO, Leandro. Balancing and cyclically sequencing synchronous, asynchronous, and hybrid unpaced assembly lines. **International Journal of Production Economics**, v. 203, p. 216–224, 2018.

LOPES, Thiago Cantos; MICHELS, Adalberto Sato; SIKORA, Celso Gustavo Stall; MAGATÃO, Leandro. Balancing and cyclical scheduling of asynchronous mixed-model assembly lines with parallel stations. **Journal of Manufacturing Systems**, Elsevier, v. 50, p. 193–200, 2019.

LOPES, Thiago Cantos; PASTRE, Giuliano Vidal; MICHELS, Adalberto Sato; MAGATÃO, Leandro. Flexible multi-manned assembly line balancing problem: Model, heuristic procedure, and lower bounds for line length minimization. **Omega**, Elsevier Ltd, v. 95, p. 102063, 2020.

LOPES, Thiago Cantos; SIKORA, Celso Gustavo Stall; MAGATÃO, Leandro. Buffer and Cyclical Product Sequence Aware Assembly Line Balancing Problem: Model and Steady-State Balancing Case Study. *In*: **Annals of the XLVIII Brazilian Symposium of Operations Research**. Vitória-ES, Brazil: *[s.n.]*, 2016. p. 3458–3469.

LOPES, Thiago Cantos; SIKORA, Celso Gustavo Stall; MICHELS, Adalberto Sato; MAGATÃO, Leandro. A New Model for Simultaneous Balancing and Cyclical Sequencing of Asynchronous Mixed-Model Assembly Lines with Parallel Stations. *In*: **Annals of the XLIX Brazilian Symposium of Operations Research**. Blumenau: *[s.n.]*, 2017. p. 3570–3581.

LOPES, Thiago Cantos; SIKORA, Celso Gustavo Stall; MICHELS, Adalberto Sato; MAGATÃO, Leandro. A Matheuristic for Makespan Minimization of Asynchronous Stochastic Mixed-Model Assembly Lines. *In*: **Annals of the L Brazilian Symposium of Operations Research**. Rio de Janeiro-Brasil: *[s.n.]*, 2018. p. 83377.

LOPES, Thiago Cantos; SIKORA, Celso Gustavo Stall; MICHELS, Adalberto Sato; SILVA, Arinei Carlos Lindbeck da; MAGATÃO, Leandro. Modeling the Stochastic Steady-State of Mixed-Model Asynchronous Assembly Lines with Markov Chains. *In*: **Proceeding of the XIX Latin-Iberoamerican Conference on Operations Research, CLAIO**. Lima-Peru: *[s.n.]*, 2019. p. 183–189.

LOPES, Thiago Cantos; SIKORA, Celso Gustavo Stall; MICHELS, Adalberto Sato; MAGATÃO, Leandro. An iterative decomposition for asynchronous mixed-model assembly lines: combining balancing, sequencing, and buffer allocation. **International Journal of Production Research**, v. 58, n. 2, p. 615–630, 2020.

LOPES, Thiago Cantos; SIKORA, Celso Gustavo Stall; MICHELS, Adalberto Sato; MAGATÃO, Leandro. Mixed-Model Assembly Line Balancing with Given Buffers and Product Sequence: Model, Formulation Comparisons and Case Study. **Annals of Operations Research**, v. 286, p. 475–500, 2020.

LOPES, Thiago Cantos; SIKORA, Celso Gustavo Stall; MOLINA, Rafael Gobbi; SCHIBELBAIN, Daniel; RODRIGUES, L C A; MAGATÃO, Leandro. Balancing a robotic spot welding manufacturing line : An industrial case study. **European Journal of Operational Research**, Elsevier B.V., v. 263, n. 3, p. 1033–1048, 2017.

LOURENÇO, Helena R.; MARTIN, Olivier C.; STÜTZLE, Thomas. Iterated Local Search. *In*: GLOVER, Fred; KOCHENBERGER, W; GARY, A (Ed.). **Handbook of Metaheuristics**. 1. ed. *[S.l.]*: Springer US, 2003. chap. 11, p. 320–353. ISBN 978-0-306-48056-0.

LUSA, Amaia. A survey of the literature on the multiple or parallel assembly line balancing problem. **European Journal of Industrial Engineering**, v. 2, n. 1, p. 50–72, 2008.

MATANACHAI, Sittichai; YANO, Candace Arai. Balancing mixed-model assembly lines to reduce work overload. **IIE Transactions**, v. 33, n. 1, p. 29–42, 2001.

MAVROTAS, G; DIAKOULAKI, D. Multi-criteria branch and bound: A vector maximization algorithm for Mixed 0-1 Multiple Objective Linear Programming. **Applied Mathematics and Computation**, v. 171, p. 53–71, 2005.

MCCORMICK, S. Thomas; RAO, U. S. Some complexity results in cyclic scheduling. **Mathematical and Computer Modelling**, v. 20, n. 2, p. 107–122, 1994.

MCNAMARA, Tom; SHAABAN, Sabry; HUDSON, Sarah. Fifty years of the bowl phenomenon. **Journal of Manufacturing Systems**, The Society of Manufacturing Engineers, v. 41, p. 1–7, 2016.

MERENGO, C.; NAVA, F.; POZZETTI, A. Balancing and sequencing manual mixed-model assembly lines. **International Journal of Production Research**, v. 37, n. 12, p. 2835–2860, 1999.

MICHELS, Adalberto Sato; LOPES, Thiago Cantos; GUSTAVO, Celso; SIKORA, Stall; MAGATÃO, Leandro. The Robotic Assembly Line Design (RALD) problem: Model and case studies with practical extensions. **Computers and Industrial Engineering**, Elsevier, v. 120, p. 320–333, 2018.

MICHELS, Adalberto Sato; LOPES, Thiago Cantos; MAGATÃO, Leandro. An exact method with decomposition techniques and combinatorial Benders' cuts for the type-2 multi-manned assembly line balancing problem. **Operations Research Perspectives**, Elsevier, v. 7, p. 100163, 2020.

MICHELS, Adalberto Sato; LOPES, Thiago Cantos; SIKORA, Celso Gustavo Stall; MAGATÃO, Leandro. A Benders ' decomposition algorithm with combinatorial cuts for the

multi-manned assembly line balancing problem. **European Journal of Operational Research**, Elsevier B.V., v. 278, n. 3, p. 796–808, 2019.

MOON, Ilkyeong; LOGENDRAN, Rasaratnam; LEE, Jeonghun. Integrated assembly line balancing with resource restrictions. **International Journal of Production Research**, v. 47, n. 19, p. 5525–5541, 2009.

MOSADEGH, H.; ZANDIEH, M.; GHOMI, S. M. T. FATEMI. Simultaneous solving of balancing and sequencing problems with station-dependent assembly times for mixed-model assembly lines. **Applied Soft Computing**, Elsevier B.V., v. 12, p. 1359–1370, 2012.

NAKADE, Koichi; OHNO, Katsuhisa; SHANTHIKUMAR, J. George. Bounds and approximations for cycle times of a U-shaped production line. **Operations Research Letters**, v. 21, p. 191–200, 1997.

NILAKANTAN, J.M.; LI, Z.; TANG, Q.; NIELSEN, P. MILP models and metaheuristic for balancing and sequencing of mixed-model two-sided assembly lines. **European Journal of Industrial Engineering**, v. 11, n. 3, p. 353–379, 2017.

OTTO, Alena; OTTO, Christian; SCHOLL, Armin. Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. **European Journal of Operational Research**, v. 228, n. 1, p. 33–45, 2013.

ÖZCAN, Ugur. Balancing stochastic two-sided assembly lines: A chance-constrained, piecewise-linear, mixed integer program and a simulated annealing algorithm. **European Journal of Operational Research**, v. 205, n. 1, p. 81–97, 2010.

ÖZCAN, Ugur; ÇERÇIOGLU, Hakan; GÖKÇEN, Hadi; TOKLU, Bilal. Balancing and sequencing of parallel mixed-model assembly lines. **International Journal of Production Research**, v. 48, n. 17, p. 5089–5113, 2010.

ÖZCAN, Ugur; TOKLU, Bilal. Balancing of mixed-model two-sided assembly lines. **Computers and Industrial Engineering**, v. 57, p. 217–227, 2009.

ÖZTÜRK, Cemalettin. **Simultaneous Balancing and Scheduling of Flexible Mixed Model Assembly Lines**. 2013. 285 p. Phd Thesis (PhD Thesis) — Dokuz Eylül University, 2013.

ÖZTÜRK, Cemalettin; TUNALI, Semra; HNICH, Brahim; ÖRNEK, Arslan. Balancing and scheduling of flexible mixed model assembly lines with parallel stations. **International Journal of Advanced Manufacturing Technology**, v. 67, n. 9-12, p. 255–2591, 2012.

ÖZTÜRK, Cemalettin; TUNALI, Semra; HNICH, Brahim; ÖRNEK, M. Arslan. Balancing and scheduling of flexible mixed model assembly lines. **Constraints**, IFAC, v. 18, n. 3, p. 434–469, 2013.

ÖZTÜRK, Cemalettin; TUNALI, Semra; HNICH, Brahim; ÖRNEK, Arslan. Cyclic scheduling of flexible mixed model assembly lines with parallel stations. **Journal of Manufacturing Systems**, The Society of Manufacturing Engineers, v. 36, n. 3, p. 147–158, 2015.

PASTOR, Rafael; ANDRÉS, C.; DURAN, A.; PÉREZ, M. Tabu search algorithms for an industrial multi-product and multi-objective assembly line balancing problem, with reduction of the task dispersion. **Journal of the Operational Research Society**, v. 53, n. 12, p. 1317–1323, 2002.

PASUPA, Thanatat; SUZUKI, Sadami. Impact of work-sharing on the performance of production line with heterogeneous workers. **International Journal of Industrial Engineering and Management**, v. 10, n. 4, p. 284–302, 2019.

PEREIRA, Jordi; ÁLVAREZ-MIRANDA, Eduardo. An exact approach for the robust assembly line balancing problem. **Omega**, v. 78, p. 85–98, 2018.

PINEDO, Michael L. **Scheduling: Theory, algorithms, and systems**. 3. ed. *[S.l.]*: Springer, 2008. 662 p. ISBN 9780387789347.

POURKARIMI, L; ZAREPISHEH, M. A dual-based algorithm for solving lexicographic multiple objective programs. **European Journal of Operational Research**, v. 176, p. 1348–1356, 2007.

QUINTON, Félix; HAMAZ, Idir; HOUSSIN, Laurent. Algorithms for the Flexible Cyclic Jobshop Problem. *In*: **14th IEEE International Conference on Automation Science and Engineering (CASE 2018)**. Munich, Germany: *[s.n.]*, 2018. p. 5.

RANGAIAH, G. P. **Multi-Objective Optimization Techniques And Applications In Chemical Engineering**. *[S.l.]*: World Scientific, 2009. (Advances in Process Systems Engineering, 1). ISBN 9813148225.

REKIEK, Brahim; DOLGUI, Alexandre; DELCHAMBRE, Alain; BRATCU, Antoneta. State of art of optimization methods for assembly line design. **Annual Reviews in Control**, v. 26, n. 2, p. 163–174, 2002.

RIEZEBOS, Jan. Order release in synchronous manufacturing. **Production Planning and Control: The Management of Operations**, v. 21, n. 4, p. 347–358, 2010.

RIEZEBOS, Jan. Order sequencing and capacity balancing in synchronous manufacturing. **International Journal of Production Research**, v. 49, n. 2, p. 531–552, 2011.

ROSHANI, Abdolreza; GIGLIO, Davide. Simulated annealing algorithms for the multi-manned assembly line balancing problem: minimising cycle time. **International Journal of Production Research**, v. 55, n. 10, p. 2731–2751, 2016.

ROSHANI, A; NEZAMI, Farnaz Ghazi. Mixed-model multi-manned assembly line balancing problem: A mathematical model and a simulated annealing approach. **Assembly Automation**, v. 37, n. 1, p. 34–50, 2017.

SAIF, Ullah; GUAN, Zailin; LIU, Weiqi; WANG, Baoxi; ZHANG, Chaoyong. Multi-objective artificial bee colony algorithm for simultaneous sequencing and balancing of mixed model assembly line. **International Journal of Advanced Manufacturing Technology**, v. 75, p. 1809–1827, 2014.

SALVESON, M E. The assembly line balancing problem. **The Journal of Industrial Engineering**, v. 6, p. 18–25, 1955.

SARKER, B. R.; PAN, H. Designing a mixed-model, open-station assembly line using mixed-integer programming. **Journal of the Operational Research Society**, v. 52, n. 5, p. 545–558, 2001.

SAWIK, Tadeusz. Simultaneous versus sequential loading and scheduling of flexible assembly systems. **International Journal of Production Research**, v. 34, n. 14, p. 3267–3282, 2000.

SAWIK, Tadeusz. Monolithic vs. hierarchical balancing and scheduling of a flexible assembly line. **European Journal of Operational Research**, v. 143, n. 1, p. 115–124, 2002.

SAWIK, Tadeusz. Loading and scheduling of a flexible assembly system by mixed integer programming. **European Journal of Operational Research**, v. 154, n. 1, p. 1–19, 2004.

SAWIK, Tadeusz. Batch versus cyclic scheduling of flexible flow shops by mixed-integer programming. **International Journal of Production Research**, v. 50, n. 18, p. 5017–5034, 2012.

SCHOLL, Armin. **Balancing and sequencing assembly lines**. 2nd. ed. Heidelberg: Physica, 1999. ISBN 978-3-7908-1180-3.

SCHOLL, Armin; BECKER, Christian. State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. **European Journal of Operational Research**, v. 168, n. 3, p. 666–693, 2006.

SCHOLL, Armin; BOYSEN, Nils; FLIEDNER, Malte. Optimally solving the alternative subgraphs assembly line balancing problem. **Annals of Operations Research**, v. 172, n. 1, p. 243–258, 2009.

SCHOLL, Armin; BOYSEN, Nils; FLIEDNER, Malte. The assembly line balancing and scheduling problem with sequence-dependent setup times: Problem extension, model formulation and efficient heuristics. **OR Spectrum**, v. 35, n. 1, p. 291–320, 2013.

SCHOLL, Armin; KLEIN, Robert. SALOME: A bidirectional branch-and-bound procedure for assembly line balancing. **INFORMS Journal on Computing**, v. 9, n. 4, p. 319–334, 1997.

SCHONBERGER, Richard. **Japanese Manufacturing Techniques: Nine Hidden Lessons in Simplicity**. New York, NY: Free Press, 1982. 260 p. ISBN 0-02-929100-3.

SHTUB, Avraham. Increasing efficiency of assembly lines through computer integration, dynamic balancing and wage incentive. **Computer Integrated Manufacturing Systems**, n. 1, p. 273–277, 1993.

SIKORA, Celso Gustavo Stall; LOPES, Thiago Cantos; MAGATÃO, Leandro. Traveling worker assembly line (re)balancing problem: Model, reduction techniques, and real case studies. **European Journal of Operational Research**, v. 259, n. 3, p. 949–971, 2017.

SPINELLIS, Diomidis D.; PAPADOPOULOS, Chrissoleon T. A Simulated Annealing Approach for Buffer Allocation in Reliable Production Lines 1 Introduction and Literature Review. **Annals of Operations Research**, v. 93, p. 373–384, 2000.

STERNATZ, Johannes. The joint line balancing and material supply problem. **International Journal of Production Economics**, Elsevier, v. 159, p. 304–318, 2014.

THOMOPOULOS, Nick T. Line Balancing-Sequencing for Mixed-Model Assembly. **Management Science**, v. 14, n. 2, p. B–59–B–75, 1967.

THOMOPOULOS, Nick T. Some Analytical Approaches to Assembly Line Problems. **The Production Engineer**, v. 47, n. 7, p. 345–351, 1968.

THOMOPOULOS, Nick T. Mixed Model Line Balancing with Smoothed Station Assignments. **Management Science**, v. 16, n. 9, p. 593–603, 1970.

TIACCI, Lorenzo. Event and object oriented simulation to fast evaluate operational objectives of mixed model assembly lines problems. **Simulation Modelling Practice and Theory**, Elsevier B.V., v. 24, p. 35–48, 2012.

TIACCI, Lorenzo. Coupling a genetic algorithm approach and a discrete event simulator to design mixed-model un-paced assembly lines with parallel workstations and stochastic task times. **International Journal of Production Economics**, Elsevier, v. 159, p. 319–333, 2015.

TIACCI, Lorenzo. Simultaneous balancing and buffer allocation decisions for the design of mixed-model assembly lines with parallel workstations and stochastic task times. **International Journal of Production Economics**, Elsevier, v. 162, p. 201–215, 2015.

TIACCI, Lorenzo. Mixed-model U-shaped assembly lines: Balancing and comparing with straight lines with buffers and parallel workstations. **Journal of Manufacturing Systems**, The Society of Manufacturing Engineers, v. 45, p. 286–305, 2017.

TIACCI, Lorenzo; MIMMI, Mario. Integrating ergonomic risks evaluation through OCRA index and balancing/sequencing decisions for mixed model stochastic asynchronous assembly lines. **Omega**, Elsevier Ltd, v. 78, p. 112–138, 2018.

UNGULU, E L; TEGHEM, J. Multi-objective Combinatorial Optimization Problems : A Survey. **Journal of Multi-Criteria Decision Analysis**, v. 3, p. 83–104, 1994.

URBAN, Timothy L.; CHIANG, Wen Chyuan. Designing energy-efficient serial production lines: The unpaced synchronous line-balancing problem. **European Journal of Operational Research**, Elsevier Ltd., v. 248, n. 3, p. 789–801, 2016.

VENKATESH, J V L; DABADE, B M. Evaluation of performance measures for representing operational objectives of a mixed model assembly line balancing problem. **International Journal of Production Research**, v. 46, n. 22, p. 6367–6388, 2008.

VINCENT, Thomas; SEIPP, Florian; RUZIKA, Stefan; PRZYBYLSKI, Anthony; GANDIBLEUX, Xavier. Multiple objective branch and bound for mixed 0-1 linear programming: Corrections and improvements for the biobjective case. **Computers and Operation Research**, Elsevier, v. 40, n. 1, p. 498–509, 2013.

WALDHERR, Stefan; KNUST, Sigrid. Decomposition algorithms for synchronous flow shop problems with additional resources and setup times. **European Journal of Operational Research**, Elsevier B.V., v. 259, n. 3, p. 847–863, 2017.

YANG, Caijun; GAO, Jie; LI, Jinlin. Balancing mixed-model assembly lines with adjacent task duplication. **International Journal of Production Research**, Taylor & Francis, v. 52, n. 24, p. 7410–7427, 2014.

YILMAZ, Hamid; YILMAZ, Mustafa. A multi-manned assembly line balancing problem with classified teams: a new approach. **Assembly Automation**, v. 36, n. 1, p. 51–59, 2016.

ZHONG, Yu-guang. Hull mixed-model assembly line balancing using a multi-objective genetic algorithm simulated annealing optimization approach. **Concurrent Engineering: Research and Applications**, v. 25, n. 1, p. 30–40, 2017.