CELSO GUSTAVO STALL SIKORA

SCHEDULING PROJECTS IN OPERATING SYSTEMS: AN APPLICATION ON ASSEMBLY LINE BALANCING

Dissertation presented to Graduate Program in Electrical and Computer Engineering from Federal University of Technology - Paraná as a parcial requirement to the attainment of the degree of "Master's degree in Science" Concentration Area: Automation and Systems.

Advisor: Prof. Dr. Leandro Magatão

CURITIBA

2017

Dados Internacionais de Catalogação na Publicação

S579s 2017

Sikora, Celso Gustavo Stall
Scheduling projects in operating systems : an application on assembly line balancing / Celso Gustavo Stall Sikora.-- 2017.
180 p. : il. ; 30 cm.

Texto em inglês, com resumo em inglês
Disponível também via World Wide Web
Dissertação (Mestrado) - Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Engenharia Elétrica e Informática Industrial, Curitiba, 2017
Bibliografia: p. 173-180

1. Programação linear. 2. Balanceamento de linha de montagem. 3. Automóveis – Projetos e construção. 4. Indústria automobilística – Inovações tecnológicas. 5. Métodos de linha de montagem. 6. Engenharia elétrica – Dissertações. I. Magatão, Leandro. II. Universidade Tecnológica Federal do Paraná. Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial. III. Título.

CDD: Ed. 22 – 621.3

Biblioteca Central da UTFPR, Câmpus Curitiba

UTFPR
CAMPUS CURITIBA

UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Curitiba

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E INFORMÁTICA INDUSTRIAL
CPGEI

Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial

Título da Dissertação Nº. ____

Scheduling Projects in Operating Systems: an Application on Assembly Line Balancing

por

Celso Gustavo Stall Sikora

Orientador: Prof. Dr. Leandro Magatão (UTFPR)

Esta dissertação foi apresentada como requisito parcial à obtenção do grau de MESTRE EM CIÊNCIAS – Área de Concentração: **ENGENHARIA DE AUTOMAÇÃO E SISTEMAS** do Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial – CPGEI – da Universidade Tecnológica Federal do Paraná – UTFPR, às 13**h30** do dia 19 de abril de 2017. O trabalho foi aprovado pela Banca Examinadora, composta pelos professores doutores:

Prof. Dr. Leandro Magatão
(Presidente – UTFPR)

Prof.ª Dr.ª Maristela Oliveira dos Santos
(USP – São Carlos)

Prof.ª Dr.ª Neida Maria Patias Volpi
(UFPR)

Visto da coordenação:

Prof. Jean Carlos Cardozo da Silva, Dr.
(Coordenador do CPGEI)

Agradecimentos

Fico muito contente em me deparar com a folha em branco para os agradecimentos e perceber que não é uma tarefa fácil elencar todos aqueles que me são importantes e de alguma forma fizeram parte desse processo de dois anos de mestrado. Para fazer inteligível a menção a diferentes pessoas que estão aqui vindo de diversas áreas, cidades e até países, pensei em agrupá-los pelos papéis que representaram para mim. Assim, na impossibilidade de citar o nome de todos os que considero importantes, ao menos agradeço em grupos.

Inicio com um muito obrigado aos meus colegas de mestrado, que trabalharam, estudaram, discutiram e dividiram experiências comigo. De forma alguma a caminhada foi solitária. Agradeço aqui em especial ao Thiago, colega de mais de sete anos, e para quem não bastou só fazer tanto a graduação quanto o mestrado junto, tinha também que ter os mesmos gostos para projetos, viagens e cervejas. Não sei como teríamos feito um mestrado mais coparticipativo que este. Obrigado pela companhia nas aulas e no trabalho, pelas discussões, ideias, competições sadias, revisões de trabalhos e de artigos, muitos em coautoria. O próximo da lista é o Rafael, que além da companhia nos ajudou muito nos projetos. Seu jeito diferente de ver o mundo com certeza trouxe novas ideias, risos, discussões, mas também um pouco de trabalho. Agradeço também ao Adalberto, outro que não se contentou a ser citado uma só vez. Sua entrada na equipe trouxe uma animação a mais para o nosso dia a dia. Fiquei muito contente em te auxiliar no início e mais ainda em ver que em pouco tempo já era você que nos explicava a parte do projeto que escolheu. Agradeço também ao Marcelo, Julio, Paulo, Guilherme, Lucas e outros colegas de turmas ou de laboratório pela parceria, dicas e até por tutorias realizadas.

Sigo agora por outros colegas de universidade, mas agora no papel de mestres. Agradeço a todos os meus professores, a escalada foi muito mais fácil com a orientação de vocês. Menciono em especial o professor Arinei, que já desde um pouco antes do mestrado nos mostrou que não importa o quanto você sabe ou fez, o que vale é o quanto se consegue evoluir, com muito esforço, é claro. Obrigado em dobro aos professores Ricardo, Heitor, Myriam, Cassius e Hugo pelos ensinamentos, projetos propostos e até uma coautoria. Obrigado também às professoras Maristela e Neida, pela presença na banca, a leitura e revisão do trabalho e os comentários que estão aqui presentes no texto. No âmbito institucional, estendo o agradecimento à Universidade Tecnológica Federal do Paraná e à Pós Graduação em Engenharia Elétrica e Informática Industrial pela estrutura, aulas e a oportunidade de realizar o mestrado. Obrigado também à Universidade Federal do

Paraná e ao Programa de Pós Graduação em Métodos Numéricos em Engenharia pelo acolhimento e aulas que complementaram a minha formação.

Uma frequente crítica quanto ao seguir um mestrado seria o isolamento entre o mundo acadêmico e o mundo empresarial. Este isolamento, porém, de forma alguma ocorreu. Esta dissertação deve muito à Fundação Araucária, que não apenas financiou parte da bolsa do projeto, mas principalmente propiciou a realização do projeto dentro de uma empresa. Agradeço à Renault, pela recepção, informações e a oportunidade de pesquisar diretamente no contexto de aplicação da pesquisa. Com certeza sem a participação da Renault a dissertação teria no mínimo uma outra cara. Trabalhar parte da carga horária na Renault me possibilitou conhecer grandes pessoas, a quem também devo agradecimentos. Em especial sou muito grato ao Daniel, quem propôs e coordenou o projeto. Não somente devo a minha presença na empresa ao Daniel, mas também ao auxílio dado ao projeto, sempre defendendo nossas ideias e buscando transmitir os resultados para outras esferas. Muito obrigado pelo voto de confiança e a autonomia que nos foi dada durante os dois anos de projeto. Agradeço também ao Robinson e ao Filipe por serem de certa forma mentores durante minha estadia na empresa. O entendimento do mundo Renault e o funcionamento dos robôs, das linhas, das relações com fornecedores e muitas outras coisas aconteceu em grande parte devido a vocês. Um obrigado também aos muitos outros colegas que tive, em especial ao João e ao Cotrin.

Estes anos não foram só de aulas e projetos. Atividades ao ar livre, viagens, passeios e conversas foram imprescindíveis para descansar a cabeça das fórmulas e tabelas. É no parágrafo de amigos que a maioria dos nomes entrariam e, felizmente, não há como citar todos. Obrigado a todos os que me fizeram sorrir, me ensinaram coisas novas, me ouviram ou me acompanharam em alguma atividade, vocês foram muito importantes para o bom andamento do mestrado e por fazer esses dois anos um período feliz. Depois do agradecimento geral, gostaria de mencionar mais especificamente algumas pessoas ou grupos. Obrigado aos meus amigos e colegas da graduação, mesmo agora com cada um seguindo o seu caminho, é sempre bom revê-los: em especial aos colegas do Lacit, Arthur, Arturo, Carlos, Julio, Rubens e Sérgio, que a distância não atrapalhe em nada a amizade. Agradeço também aos colegas Maurício e Thomas, que, além da amizade, me indicaram a matéria de Pesquisa Operacional, que foi o início da jornada do mestrado. Ao Ivan e ao nosso grupo de montanhismo, muito obrigado pela iniciativa e a sua companhia, junto a belas paisagens em dias cansativos mas que ficarão guardados na memória. Agora repetindo nomes, obrigado aos meus colegas-sócios cervejeiros Julio, Adalberto e Thiago. É uma sensação ótima provar o produto final e ter orgulho de dizer 'fui eu quem fiz'. Mesmo assim, a cerveja pode ser detalhe, bom mesmo são os encontros e os dias juntos que passamos para comprar equipamento, construí-los, buscar insumos e, principalmente, produzir: com vocês o processo se torna um evento de amigos. Obrigado aos meus companheiros planejados ou espontâneos das viagens que tive a oportunidade de fazer. Foi ótimo ser tão bem recebido nos mais diversos lugares. Podem ter certeza que vou continuar lhes visitando e reitero o meu convite para virem me ver. Não posso deixar

passar uma menção especial aos meus colegas de intercâmbio, que, mesmo sendo anterior ao mestrado, me trouxe amigos que guardo e vejo até hoje. Obrigado aos amigos da Alemanha e de Aachen, em especial aos grandes amigos Antonio, Arthur, Christian, Eric, Fernando, Guilherme, Hamilton, Irineu, Jeferson, João, Leonardo, Talita e Thais que foram colegas de intercâmbio e também à Carla e aos amigos que conheci através de vocês e agora fazem parte do grupo. Obrigado pelos encontros, passeios e viagens, mas também pelo apoio e incentivo. Vocês com certeza me inspiram e são parte da razão do meu esforço para alcançar meus objetivos. Sem uma melhor tradução, "You set the bar high".

Devo um enorme agradecimento minha família. Aos meus irmãos Rodrigo e Mariana obrigado pelo interesse. Sempre me animou poder contar as novidades e falar dos meus planos e vontades. Agradeço principalmente aos meus pais: Neide e Celso. Obrigado por me propiciar o estudo, sempre da melhor forma que podiam. Obrigado pelos cursos extras, esportes, aulas de línguas. Agradeço ainda mais pelo incentivo, muito mais do que me propiciar os cursos que eu quis fazer vocês me ensinaram a importância de estudar. Agradeço ao ambiente tranquilo, a autonomia de poder decidir o que eu gostaria de fazer e estudar. Muitos pais desejam dar aos filhos o que não tiveram. Eu posso dizer que quero dar aos meus exatamente o que vocês nos deram, muito obrigado.

Para finalizar, devo não só um agradecimento, mas em grande parte a razão do meu mestrado ao meu orientador Leandro Magatão, cujo nome completo é aqui homenagem pela sempre presente formalidade nas orientações e trocas de e-mail. Primeiramente eu lhe agradeço como professor. Acredito que um pré-requisito para gostar de uma área ou matéria é entender o assunto, o que foi fácil com a sua dedicação às aulas e ao material de apoio. A aula de Pesquisa Operacional foi uma mudança importante, entrei sem certeza alguma de qual caminho seguir profissionalmente e saí convicto de que tinha encontrado minha vocação. Fora ser o motivo de eu ter me inscrito no mestrado, lhe admiro ainda mais como orientador. De início devo inúmeros agradecimentos quanto às condições propiciadas para a pesquisa. Sejam as bolsas de graduação, mestrado, ajuda com o doutorado, congressos ou projetos, você sempre buscou dar a melhor condição possível para que realizássemos nosso trabalho. Muito além da parte burocrática e financeira, devo lhe agradecer pela orientação e instrução em si dada durante estes quase três anos juntos. Não houve um texto não lido, uma equação não revisada, uma orientação que faltou. Professor, nós notamos e reconhecemos o esforço despendido, a qualidade dos trabalhos reflete em muito a sua dedicação. Agradeço ainda a autonomia que me foi depositada. Acredito que tive a liberdade de procurar e testar mais para escolher no que eu gostaria de contribuir. Sei que dei trabalho, não peguei o caminho mais fácil em questão de tema (ou troca de temas) ou de prazo de execução. Fico feliz com o resultado que estou entregando e ciente de que o documento muito lhe deve pelo incentivo, orientação e incansável dedicação.

Em resumo este agradecimento tem como função reconhecer os papéis de grandes pilares que tive durante o mestrado. Sejam colegas de mestrado, mestres, colegas de em-

presa, instituições, amigos, familiares ou meu orientador, todos tem sua parcela relevante para não só a dissertação, mas também um mestrado no qual eu fui feliz. Mais uma vez, obrigado!

Sponsorship

The author of this master thesis acknowledges the financial support from Fundação Araucária and Renault do Brasil through the Agreement 141/2015 (FA - UTFPR - RENAULT).

Resumo

A Pesquisa Operacional investiga as (melhores) formas de se configurar e coordenar sistemas ou operações usando técnicas de otimização. Geralmente, a otimização de um sistema é modelado com base no estado final almejado. Porém, como atingir ou implementar tal estado final em sistemas é pouco retratado na literatura. Esta dissertação de mestrado propõe uma nova classe de problema de otimização: a programação das operações entre o estado inicial e o final de um sistema, o Problema de Implementação. A programação das operações é especialmente importante para linhas de montagem. A indústria automobilística é fortemente baseada em linhas de produção que podem ser usadas até 24 horas por dia. Assim, as oportunidades de intervenções para mudar ou otimizar o sistema produtivo são poucas. As condições de implementação aplicadas ao balanceamento de linhas produtivas são discutidas, e as características observadas resultam no proposto Problema de Implementação de Linhas de Montagem (PILM). Na dissertação, um guia de modelagem baseado em Programação Linear Inteira Mista (PLIM) é desenvolvido para a formulação de diversas variações do Problema de Implementação. As instruções de modelagem são usadas para desenvolver um conjunto de modelos PLIM para o Problema de Implementação de Linhas de Montagem. Para a obtenção de resultados, um conjunto de instâncias é proposto. Assim, uma análise de sensibilidade em função de cada um dos parâmetros formadores das instâncias é realizada. As formulações são comparadas, junto com as diferentes formas de apresentar e resolver o problema. Ademais, um método de decomposição baseado no modelo matemático é usado para resolver um problema industrial real. A modelagem mostra-se correta para a divisão da implementação de mudanças em linhas de montagem. Os resultados mostram que a divisão do esforço de implementação resulta em apenas poucas mudanças a mais (cerca de 7% para os casos pequenos e médios) comparadas com a implementação em uma fase. A possibilidade de programar a implementação em etapas menores aumenta a aplicabilidade de projetos, que, de outra forma, requeririam grande paradas de produção.

PALAVRAS CHAVES

Programação Linear Inteira Mista; Problema do Balanceamento de Linhas de Montagem; Programação de Projetos; Problema de Implementação; Programação de Implementação; Problema de Implementação de Linhas de Montagem

Abstract

Operations Research investigates the (best) ways to configure and coordinate systems or operations with optimization procedures. Usually, the optimization of a system is modeled based on the aimed final configuration. However, little is published about how to reach or implement such optimal configurations in the systems. This master thesis proposes a new class of optimization problem: a scheduling of operations between initial and final states of a system, the Implementation Problem. The scheduling of operations is especially important to assembly lines. The automotive industry strongly relies on production lines that can operate 24 hours a day. Thus, the intervention opportunities to modify or optimize the production system are very few. The implementation conditions of balancing on assembly lines are discussed, and the observed characteristics result in the proposal of the Assembly Line Implementation Problem (ALIP). The master thesis proposes a Mixed-Integer Linear Programming (MILP) modeling guide for the formulation of several variations of Implementation Problems. The modeling instructions are used to develop a set of MILP models for the Assembly Line Implementation Problem. For the results, a dataset is proposed and a sensitivity analysis on each of the consistent parameters of the dataset is performed. The proposed formulations are compared, along with the different forms of presenting and solving the problem. Furthermore, a model based decomposition method is used to solve a real-world industrial problem. The modeling correctly represents the division of the implementation of changes in assembly lines. The results show that the division of the effort in multiple stages only need a few more changes (around 7% for the small and medium cases) comparing to a straightforward implementation. The possibility of scheduling the implementation in smaller steps increases the applicability of projects that otherwise would require a large system's stoppage time.

KEYWORDS

Mixed-Integer Linear Programming; Assembly Line Balancing Problem; Project Scheduling; Implementation Problem; Implementation Scheduling; Assembly Line Implementation Problem

List of Figures

List of Tables

List of Labels

ALBP	Assembly Line Balancing Problem
ALIP	Assembly Line Implementation Problem
ALWABL	Assembly Line Worker Assignment Balancing Problem
ARALBP	Assignment Restricted Assembly Line Balancing Problem
BB&R	Branch-Bound and Remember
BI	Bimodal
BN	Bottleneck
CH	Chains
CT	Cycle Time
GALBP	General Assembly Line Balancing Problem
MILP	Mixed-Integer Linear Programming
MX	Mixed
NP	Number of Periods
NS	Number of Stations
OS	Order Strength
PB	Peak at the Bottom
PD	Precedence Diagram
PM	Peak in the Middle
RALBP	Robotic Assembly Line Balancing Problem
SALBP	Simple Assembly Line Balancing Problem
SH	Simplification Hypothesis
TD	Process Time Distribution
TSALBP	Time and Spaced Constrained Assembly Line Balancing Problem
TWALBP	Traveling Worker Assembly Line Balancing Problem

Contents

Chapter 1

Introduction

1.1 Motivation and Overview

Operations research is a science that aims for the investigation of the (best) way to configure and coordinate systems or operations. Every system in which choices can be made can be treated by some optimization method. To do so, models are used to formally represent real world problems in mathematical problems to be solved by a computational method.

Optimization problems are based on the problem data and boundary conditions and usually aim at finding the optimal configuration for a given system. The best configuration of a system is theoretically the most profitable operational form that respects the available amount of resources. Usually, the optimal configuration does not depend on present conditions or how the systems operate in the present. There is, still, a gap between knowing the best configuration and being able to implement it. The more different the optimal answer is to the actual system, the more effort is required to reach the desired configuration.

The implementation of a project is not guaranteed just because an operational research approach reaches an optimal condition. Companies usually perform several activities simultaneously and resources are limited. A manager frequently has to decide between several projects and invariably some of them must be discarded or put on hold.

The implementation of projects itself can be subjected to optimization procedures.

The Project Scheduling Problem, for instance, consists in scheduling the operations that are necessary to carry an implementation. The problem consists in determining the order of the operations that should be perform to minimize the necessary time for the project's conclusion. The project scheduling is useful for optimizing non-operative systems, such as the construction of a building. In these cases, the system just works after the implementation phase.

The industrial ambiance is dynamic and competitive. Projects have tight deadlines and it is not uncommon that a feasible, fast, and easy solution is usually applied rather than a more detailed and time intensive approach. Optimization processes depend on data gathering, modeling, and solving, requiring a study that can span for weeks or months depending on the application. Meanwhile, the real process modeled could have been modified in order to adapt a product to the market, introduce new models, reduce costs, etc. Along with the possibility of an outdated optimal answer, there are also implementation issues. Implementing a new configuration requires capital, resources, and possibly the stoppage of the system, resulting in potential lost revenue.

Assembly lines in automotive industry, which is the focus of this master thesis application, are severely restricted in terms of availability for changes. The automotive production is strongly based on flow shop arrangements, whose advantages are the low running costs and the proper to mass production. This design also has negative aspects: once the operations are based on production lines, any interruption in a line may affect the output of the entire production system. Buffers are usually used to absorb small production disruptions, however, they are usually limited to some production hours.

Despite emergency maintenances or machine breakdowns, alterations on production systems are limited to non-operative hours, such as, vacations, weekends, holidays, or inoperative night shifts. The production engineer usually studies and plans the implementations or improvements during commercial hours, but the application itself is faded to be performed overnight or during weekends.

As the possible time windows for alterations in assembly lines are short, large alterations aiming a global optimal performance for a line are usually impracticable. The routinely proceeding consists in small adaptations focusing in a specific problem or directly in the bottleneck. As it will be discussed in the literature review (Chapter 2), it is

noted a strong focus on modeling and solution methods, while nothing is implied about how to implement the outcome of the research effort. Almost every survey on assembly line balancing point out a large gap between the academic literature and the industrial practice. Nonetheless, reports on the implementation of projects are seldom reported or even mentioned.

In order to contribute with the applicability of optimization approaches in the industry, this master thesis proposes a scheduling for the implementation of new balancing in assembly lines considering the presented time window difficulties. The main idea consists in dividing the implementation in smaller steps that could be performed during regular stoppage times (overnight or weekends), as exemplified in Figure 1.1. An important operational constraint is related to the intermediary production phases. After a weekend intervention, for instance, the system must be ready to perform at acceptable levels in Monday morning. More generally, operative systems may present further restrictions on how implementations can be performed. The scheduling should also consider the intermediary configurations from which the system passes during its transformation. This way, a new class of problems arise when one considers the system's output during the implementation phases: the Implementation Problem and the specific Assembly Line Implementation Problem (ALIP), both proposed throughout the master thesis.

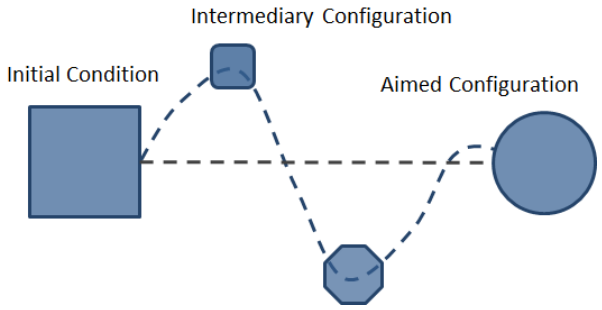

Figure 1.1: Example of intermediary stages in an implementation.

1.2 Optimization Steps

According to Hillier and Lieberman (2010), the application of Operations Research can be divided in six steps:

1. Define the problem and collect data;
2. Formulate a mathematical model to represent the problem;
3. Develop a computational procedure to solve the problem based on the model;
4. Test the model and adjust/improve it if necessary;
5. Prepare the organization to the continuous application of the model;
6. Implement the suggested model solution.

These steps are addressed to a commercial practice in which Operations Research professionals serve as consultants in the implementation of an optimization system. The academic research, however, usually do not go through all the steps. As it can be seen in Chapter 2 (at least for the research on assembly lines), most of the publications focus on the third item of Hillier and Lieberman (2010)'s list: the development of solution procedures. Due to the different agendas of the academic and industrial worlds, a gap on the procedures and practices is formed.

This master thesis enunciates a new class of optimization problem that can be directly linked to the sixth item on the list: the implementation. Adapting systems to assume a different configuration may be costly and time consuming. Assembly lines, the application focus in this document, present limited availability to interventions or rearrangements due to the almost continuous production applied in the industry. These difficulties, however, can be seen as restrictions in an implementation scheduling problem. The definition and modeling of the best form to implement changes in an operative system is the focus of the next chapters.

This work defines the problem (item 1), formulates the mathematical model (item 2), and uses a universal solver to solve, adjust, and validate the model (items 3 and 4). An industrial case study is presented as an example of the model's application. Although a real study is performed and presented, the implementation of the answers themselves is part of further works. Nevertheless, the development and investigation of the proposed problem contributes to reduce or optimize the implementation effort (item 6) of projects (mainly for assembly lines).

1.3 Objectives and Document Outline

The main objective of this master thesis is to **introduce, define, and model the problem of the implementation scheduling of new balancing in assembly lines**. The specific objectives are detailed and stressed in bold along with the presentation of the document outline and an introduction of each of the following chapters.

The content of this document is strongly linked to the Assembly Line Balancing Problem (ALBP). Chapter 2 brings the definition of this optimization problem, as well as the simplification hypotheses and assumptions used in the simple version of the problem (Simple ALBP, or SALBP). A cluster of articles is detailed to introduce the most important research on the theme. The solution methods are described along with ALBP variations and extensions. Section 2.7 brings a review on the application of solution methods in the industry. It becomes evident that, although much have been published in respect to solution methods, little mention to the implementation and application of the results is given in the literature. Furthermore, a review on the datasets used for SALBP is presented. In general, Chapter 2 is important to **investigate ALBP contributions and identify the gap between the developed models and the industrial implementation of the obtained answers**.

In Chapter 3, the scheduling of implementation projects is discussed. **A new class of problem, the Implementation Problem, is identified and explained** as a variation of the Project Scheduling. More specifically for ALBP, **the context and conditions for the implementation of a new balancing on real assembly lines is presented and discussed**. These considerations are used as boundary conditions for the **definition of the Assembly Line Implementation Problem (ALIP)**. The ALIP concerns the stepwise planning or scheduling of the operations and changes in an assembly line in order to transform its balancing into a better or optimal one.

The modeling tool elected to describe mathematically the treated problem is Mathematical Programming, more specifically the Mixed-Integer Linear Programming (MILP). Chapter 4 contains a **modeling guide for the formulation of the Implementation Problem**. The guide scope is intended to not only be relevant to assembly line's related problems, but it is built in a generic form. Expressions and directions are given for several

possible variations that the Implementation Problem may present.

Once the chapter is a guide for the formulation of generic implementation problems, for the .

Chapter 5 presents the **modeling**, based on the model guide, **for the Assembly Line Implementation Problem**. MILP models are detailed for the standard problem and some variations, as well as the different possible objective functions.

A dataset for the ALIP is presented in Chapter 6. This chapter also presents the modeling results. The different forms of defining the problem and the multiple models are tested and compared. The instances characteristics (or parameters used to build the dataset) are analyzed in order to **measure the contribution of each of the parameters used to build the dataset in the computational solving difficulty**. Furthermore, **an industrial based case study is presented** and solved with **proposed decomposition procedures**.

Finally, Chapter 7 contains the concluding remarks. A discussion of every chapter is given, along with possible directions for future works that are not the scope of this document.

1.4 Author's related works

This section is reserved to a brief introduction of the author's published or proposed works coauthored during the course of the master studies. The contributions are divided in three topics: Balancing Manual Production Lines Under Low Production Rate; Optimization and Unification of Car Body Welding Procedures; and Other Works.

Balancing Manual Production Lines Under Low Production Rate

The study of the balancing configurations under low production rate spans from the end of the undergraduate period to the beginning of the master's studies. Due to the economic crisis, the automotive industry sales and therefore its production levels were severely reduced. The main impact was observed in the reduction of the labor force, although the machinery and equipment were maintained. Thus, it could be observed that

in some lines there were a surplus of workstations compared to the workers.

The greater amount of workstations provoked some difficulties in the balancing of tasks among a reduced number of workers. The equipment was fixed and distributed among workstations, so that some workers had to be responsible for more than one workstation. The travel time between workstations, however, must be considered in the worker cycle time.

The first contribution on this topic is a congress paper (Sikora et al., 2015) that contained a Mixed-Integer Linear Programming (MILP) model for the balancing problem with an approximation for the movement time. A further development (Sikora, Lopes and Magatão, 2017), published in the European Journal of Operational Research, improved the MILP formulation to include an exact calculation for the travel time based on an integrated Traveling Salesman Problem (TSP) model. Both papers proposed variable reduction techniques to eliminate non-potentially optimal variables. The journal publication proposed real industrial case studies that could only be solved by the application of the reduction techniques.

Another contribution to the theme is due to Sikora, Lopes, Lopes and Magatão (2016), who propose a genetic algorithm for the assembly line balancing problem with assignment restrictions. Once pieces of equipment are frequently considered fixed to workstations, the genetic algorithm considered the allocation possibilities in the solution procedure. Reduction in the search space and focus on more promising regions were also used in the building of the algorithm.

Optimization and Unification of Car Body Welding Procedures

The main master project was developed in cooperation with Fundação Araucária and Renault do Brasil, in the outskirts of Curitiba, under the agreement 141/2015.

The automotive industry had plans to adapt the car body production of all vehicle models into a unified production system, named mono-flux. The project consists of optimization potentials in terms of balancing, vehicle sequencing, implementation scheduling, and line design.

The car body is usually assembled by joining sheets of metal using resistance spot

welding procedures. The spot welding procedures are very short in duration and they can be divided in groups of similar points with the same properties. Sikora, Lopes, Schibelbain and Magatão (2017) proposed a MILP formulation to treat multiple identical tasks with integer variables. The approach gathered the information of similar tasks into a simplified formulation that proved to be easier to solve for welding lines.

Using the points grouping strategy, a balancing model for the robotic spot welding manufacturing line problem is proposed in Lopes, Sikora, Molina, Schibelbain, Rodrigues and Magatão (2017). According to the paper, most of the published papers on assembly line balancing problems relate to the final assembly phase, when the car structure (body) is already assembled. The welding procedures presented substantial differences such as the accessibility of electrodes to the welding goal points, the measurement of movement time between two welding points, and the interference caused by multiple robots allocated in the same workstation. A methodology proposed to the optimization of robotic weld spot manufacturing lines was deposited as a patent (Schibelbain et al., 2017).

The balancing model optimal answer implied that several tasks should be reallocated in order to achieve the optimal cycle time. In fact, about half of the tasks' assignments would have to change to reach the model's optimal configuration. Such implementation would require more than a month of line stoppage, during which no production would be possible. Therefore, dividing the implementation in smaller phases was necessary. Small implementations could be performed during weekends or holidays, making it possible to implement greater line changes. With this view, Sikora, Lopes and Magatão (2016) proposed a scheduling model to minimize the multi-phase implementation effort. This congress paper introduced the Assembly Line Implementation Problem, which is the main topic of this master thesis.

Up to now, the presented models suppose a fixed production layout. That is, the stations, robot positions, and subassemblies are already defined and fixed. By relaxing this hypothesis we have a line design problem. A congress publication (Lopes, Michels, Molina, Sikora and Magatão, 2016) considers the necessary tasks and plans the line choosing the number of stations and robots and their locations. At the same time, the station paralleling possibilities are considered to reduce the piece's handling time in a cost-based model. An industrial case study and a sensitive analysis on the model's parameters is

also present in a journal paper under review (Michels et al., 2017).

Finally, the production of several models in the same assembly line is related to a vehicle sequencing problem. Different models may require similar, but different operations, usually with different production times. An effective balancing should consider the sequence and the allocation of buffers. A balancing model aware of the buffer layout and product sequence is proposed in a congress paper by Lopes, Sikora and Magatão (2016), and further expanded to a journal publication under review (Lopes, Sikora, Michels and Magatão, 2017b). An integrated approach can produce even better results by treating the balancing, sequencing, and buffer allocation problem together, as it is proposed in the journal publication under review by Lopes, Sikora, Michels and Magatão (2017a).

Other Works

Along with the projects, other non-related works were also presented in conferences. In Michels et al. (2016) a linear programming model is proposed for the design of integrated renewable energy systems for small cities in Brazil. Furthermore, simulation tools were used to find bottlenecks and measure the effect of investment changes in a door manufacture cell (Meira et al., 2016).

List of Publications

A list of publications is proposed to summarize the contributions related to the master's study:

Published journal papers:

- (Sikora, Lopes and Magatão, 2017) - **European Journal of Operational Research** - *Traveling worker assembly line (re)balancing problem: Model, reduction techniques, and real case studies*;
- (Sikora, Lopes, Schibelbain and Magatão, 2017) - **Computers and Industrial Engineering** - *Integer based formulation for the simple assembly line balancing problem with multiple identical tasks.*

Published congress papers:

- (Sikora et al., 2015) - **XLVII Simpósio de Brasileiro de Pesquisa Operacional** - *Variable Sets Reduction for Assembly Line Balancing Problem: MILP Model and Case Studies*;

- (Sikora, Lopes, Lopes and Magatão, 2016) - **Latin-America Congress on Computational Intelligence, LA-CCI 2015** - *Genetic algorithm for type-2 assembly line balancing*;

- (Sikora, Lopes and Magatão, 2016) - **XLVIII Simpósio de Brasileiro de Pesquisa Operacional** - *Implementing Changes in the Balancing of Assembly Lines: a Sequencing Problem Addressed by MILP*;

- (Lopes, Michels, Molina, Sikora and Magatão, 2016) - **XLVIII Simpósio de Brasileiro de Pesquisa Operacional** - *An MILP formulation for robotic assembly line design with parallel stations, dead time and equipment selection*;

- (Lopes, Sikora and Magatão, 2016) - **XLVIII Simpósio de Brasileiro de Pesquisa Operacional** - *Buffer and Cyclical Product Sequence Aware Assembly Line Balancing Problem: Model and Steady-State Balancing Case Study*;

- (Michels et al., 2016) - **XLVIII Simpósio de Brasileiro de Pesquisa Operacional** - *Using a LP Model for the Design Analysis of an Integrated Renewable Energy System*;

- (Meira et al., 2016) - **25th SAE BRASIL International Congress and Display** - *Optimização do Setup de uma Célula de Manufatura de Portas.*

Deposited patents:

- (Schibelbain et al., 2017) - **Institut national de la propriété industrielle (INPI-France)** - *Procédé de réduction de temps de cycles d'une ligne automatisée de soudage et installation de soudage correspondante.*

Journal publications under review:

- (Lopes, Sikora, Molina, Schibelbain, Rodrigues and Magatão, 2017) - *Balancing a Robotic Welding Manufacturing Line: Problem, Model and Case Study*;

- (Michels et al., 2017) - *The Robotic Assembly Line Design (RALD) Problem: Model and case studies with practical extensions*;
- (Lopes, Sikora, Michels and Magatão, 2017a) - *Balancing, Cyclical Scheduling and Buffer Allocation Mixed-Model Assembly Line Problem: Model and Case Studies*;
- (Lopes, Sikora, Michels and Magatão, 2017b) - *Mixed-Model Assembly Lines Balancing with Given Buffers and Product Sequence: Model, Formulation Comparisons and Case Study.*

Chapter 2

ALBP Review

This chapter presents a definition for Assembly Line Balancing Problems. A review on the simplification hypotheses, solution methods, and problems variations give an extensive panorama of the research that have been published in the last decades. The industrial applicability of the scientific development is discussed, exhibiting a lack of academic concern in the industrial implementation. Finally, the datasets concerning assembly line balancing problems are presented.

2.1 Production Layout

Production systems can be organized in several forms. Depending on the variety, volume, and complexity produced, same layout patterns are known to be efficient. Krajewski et al. (2013) describe four usual forms of layout design used in production systems: job-shop, flow-shop, hybrid, and fixed position.

The job-shop or functional design consists in grouping similar activities in the same physical space. An example would be gathering milling machines at one side of the layout and drilling machines at the other side. This way, the layout is divided by the function of the machines or employees. The job-shop design is an efficient configuration for low production volumes or an unpredictable demand. The layout provides a high flexibility for the production of personalized products. Several different products can be produced simultaneously in the same system. Therefore, the machines present more general use

and the installing costs are said less intensive.

The job-shop presents disadvantages in its complex flow of products. Since the functional design usually works with a wider mix of products, the different order of the operations may make the controlling and handling of pieces difficult to perform. Two important optimization problems arise from the design of job-shops: the positioning of machines and the scheduling of products.

The flow-shop production system or product-based design consists in allocating the machines and resources around the product flow. In other words, the production system is organized as production stages of a product. The flow-shop configuration is well adapted for the mass production of homogeneous products. Assembly or production lines are built specifically to perform operations for a given product. The high utilization of the equipment, regular material flow, high throughput, low setup time, and easy material handling are some of the advantages of this production layout. An assembly line, on the other hand, requires more intensive investment. The maintenance and repair are also more drastic operations. If one station is stopped, the entire system must be put on hold. Because of the high costs and high utilization of the line, designing a flow-shop is a medium to long-term decision. One of the important optimization problems that arise is how to divide the tasks into the system's stations.

Assembly lines present a low flexibility if compared to job-shop layouts. Either a single product or a family of similar products are adequate for this production layout. The operation with multiple products results in sequencing and scheduling problems whose solution may be important to assure high levels of productivity.

Hybrid layouts are a middle term between the functional and product designs. They can be used to enjoy the advantages of both systems. The formation of production cells, a grouping with two or more workstations, is common in hybrid designs. The hybrid layout is useful when there is a high variety of products, but they can still be gathered in groups of similar products.

The last production design is the fixed position layout. In this configuration, workers or machines move around the product, which is maintained in a fixed position. Layouts with fixed positions are useful for products of large dimensions: the construction of buildings, ships, and airplanes can be used as examples. This layout minimizes the costs of

moving pieces or is the only feasible option when the product cannot be transported. The optimization problem that arises from the fixed position layout is how to coordinate the workers around the product. Project scheduling may be useful to assure the correct timing and avoid interference between different tasks.

2.2 Assembly Line Balancing

Assembly lines are product-oriented production systems. The necessary production equipment is organized in a serial matter so that the product flow is simple and unidirectional. The unfinished products, or **(work)pieces**, are usually moved using conveyor belts or other mechanical handling equipment. With these characteristics, the system is configured to the mass producing of standardized products.

The operations or activities that must be performed are called **tasks**. Each task is considered to be indivisible and requires an amount of time to be performed, named **processing time**. The physical locations, in which tasks are performed, are called **(work)stations**. A workstation can be both manual or robotic and consists of a segment of the assembly line, equipment and machinery.

In serial lines, workpieces are moved along the workstations. The average time interval between the production time of two adjacent workpieces is called **cycle time** (Scholl, 1999). The line's output is determined by the cycle time: the faster pieces are produced, the higher is the production rate and the lower must be the cycle time.

In flow lines, each product passes through all workstations in the same sequence. A piece is only produced correctly when all operations are performed. Therefore, a workstation could only release a product when all its tasks are finished. Each workstation have its own cycle time, which affects the line's global productivity. A workstation that has finished its operations but have not yet received another piece is said to be starved. On the contrary, a workstation is blocked if its operations are finished, but the next station is not yet free to receive another product piece.

A **balancing** is a feasible task division among the workstations. Every task assignment results in different workloads or cycle times for each workstation. Due to starvation and blocking, a line's global cycle time is bounded by the most loaded workstation (also

called **bottleneck**). Hence, a balancing that produces a smooth division of workload results in more efficient assembly lines. A perfect balancing is one in which all workstations produce at exactly the same cycle time. The difference of a station's cycle time and the line's global cycle time is called **idle time**.

Figure 2.1 can exemplify the concepts presented in the previous paragraphs. In the left part of the graph, every column represent a station and its workload distribution. The height of the bar represent the processing time that has been assigned to the station. Note that the processing time of the stations are uneven. If a line operates in such configuration, the steady-flow behavior would be represented by the right of the figure: the most loaded station is the bottleneck, while the other stations present idle time. The bottleneck (station 1) states the production rate and, therefore, the cycle time of the line.

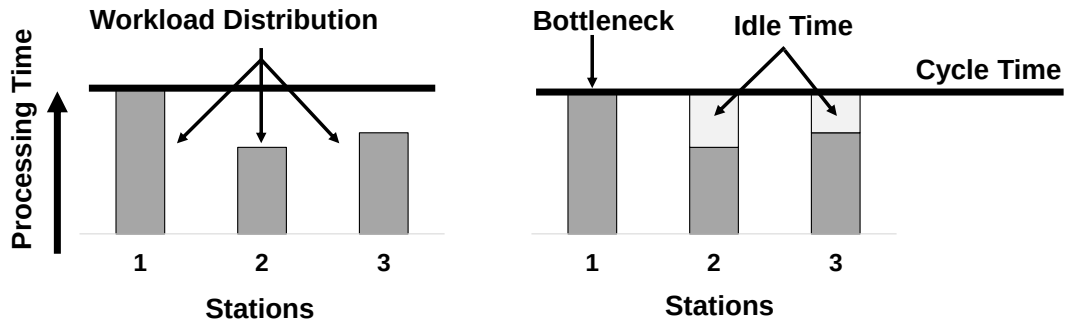

Figure 2.1: Example of line balancing for three stations.

Balance assembly lines is not just divide the total processing time by the number of workstations. Some practical restrictions should also be observed and can result in unavoidable idle time. The first common assumption of the assembly line balancing is that tasks are **not preemptive**. That is, tasks cannot be divided and performed in different stations. This way, balancing an assembly line can be seen as a generalized Bin Packing Problem (Wee and Magazine, 1982). Tasks could be represented by objects that should be fitted in boxes or bins representing the workstations. Each object presents the duration time as an one dimensional characteristic and the size of a bin is also measured in time units. Thus, as the Bin Packing is a NP-Hard problem, so does the Assembly Line Balancing Problem (Wee and Magazine, 1982).

Another form of restriction the balancing of tasks must respect is **precedence re-**

lations between operations. There are operations that must be performed before others. As the flow of products is serial, the order of the stations is determined by their position in the assembly line. It would not make sense to pack an electronic product in the first station if the following stations must perform tasks on the component, for instance. Therefore, the precedence relations affects the order in which tasks must be performed.

The precedence relations can be represent by a **precedence graph** or diagram. An example of such precedence graph can be seen in Figure 2.2. Each node or circle represents a task, containing its identification number on the center and the processing time on the upper right side. Each directed arc represents an explicit precedence relation. For instance, the arc between nodes 1 and 2 indicates that Task 1 must be performed before Task 2. Therefore, task 1 must be assigned to a workstation positioned earlier in the assembly line or, at its limit, at the same workstation. Indirect precedence relations are transitive precedence relations. For instance, there is no direct arc from Task 3 to Task 7. However, once Task 3 precedes Task 5, and Task 5 precedes Task 7, Task 3 must also precede Task 7. Unrelated tasks, such as Tasks 4 and 7, can be assigned in any order. Task 3 is said a direct predecessor of Task 5 and an indirect predecessor of Task 7. On the other hand, Task 7 is a direct follower of Task 5 and an indirect follower of Task 3.

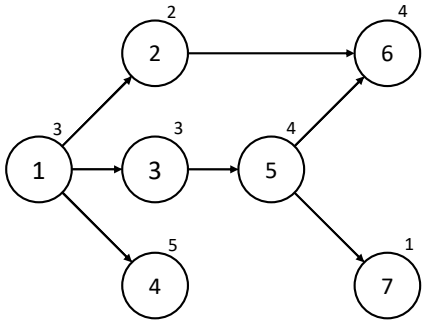

Figure 2.2: Example of precedent graph with task number inside the circle and processing time on the right side.

The relative amount of precedence relations on the precedence graph is measured by the **Order Strength (OS)**. The order strength varies from 0 to 1, being 0 equal to a precedence diagram "without precedence relations" and 1 equal to a strict ordering of tasks. OS is calculated based on the ratio of the number of existent direct and indirect precedence relations ($Prec$) and the maximal number of possible relations ($n(n-1)$) :

$OS = 2 \cdot |Prec|/(n(n-1))$ (Scholl, 1999).

The **Assembly Line Balancing Problem** (**ALBP**) is then the problem of finding the best task assignments for an assembly line, respecting its operational restrictions and precedence relations.

2.3 Simple Assembly Line Balancing Problem

The Assembly Line Balancing Problem has been of interest of researchers for more than 60 years. The first published paper on the issue is due to Salveson (1955). Although the study of task assignment and line's cycle time already existed, Salveson introduced the Assembly Line Balancing Problem as an optimization problem. However, the author presented a linear programming formulation that was unable to assure the indivisibility of tasks.

One year after Salveson's publication, Jackson (1956) presented an enumerative procedure that is able to solve ALBP considering both the indivisibility and precedence constraints. Jackson developed dominance criteria to eliminate several unpromising partial solutions that are still used in the modern solution methods. The first consistent mixed-integer linear programming model was proposed by Bowman (1960) and further improved by White (1961).

The Assembly Line Balancing Problem as a simple division of tasks respecting indivisibility and precedence relations contains several simplification hypotheses when compared to real assembly lines. Baybars (1986) introduced on his survey the concept of **Simple Assembly Line Balancing Problem** (**SALBP**). The SALBP contains the simplification assumptions that are needed to treat the problem as a Bin Packing Problem with precedence relations. Along with the precedence relations, the indivisibility of tasks and the requirement of assigning all tasks, the Simple Assembly Line Balancing Problem contains the following Simplification Hypotheses (SH):

- (SH-1) All stations are equally equipped and manned;
- (SH-2) Task processing time is independent of the workstation;
- (SH-3) Any station is able to perform any task;

- (SH-4) The assembly line is serial, no feeder or subassembly lines are considered;
- (SH-5) The assembly line is designed for the production of a single product;
- (SH-6) All problem's parameters are deterministic (processing time and precedence relations);
- (SH-7) The cycle time of every station is the sum of the assigned tasks' processing time.

Several problem variations arise relaxing one or more simplification hypotheses. Baybars (1986) defines these variations as **General Assembly Line Balancing Problem** (**GALBP**). GALBP refers to a rich variety of problems that are mostly based on SALBP, but models one or more characteristics that need further reasoning than the simplification limitations.

2.4 Problem Types

We have seen that Assembly Line Balancing consists in finding a feasible division of tasks into workstations. The Assembly Line Balancing Problem is defined as an optimization problem, whose objective is to find the best task assignment. The best division depends on some evaluation criteria, that can change depending on the optimization problem in hand.

In practice, the objective function of balancing problems would be either minimizing costs or maximizing output levels. The simple version of the problem allows the expression of these objectives in more simple measurable ideas. Once every station is equally equipped and manned, the cost of an assembly line is directly proportional to the number of stations (NS). The output levels are the inverse of the cycle time (CT). So, the number of stations and the cycle time are enough to express the optimization objectives.

According to Scholl (1999), SALBP problems can be classified in 4 problem versions: SALBP-F, SALBP-1, SALBP-2, and SALBP-E, as described in Table 2.1. The versions depend on whether the number of stations and the cycle time are given or if they should be minimized. In the simplest problem, both the number of stations and the required cycle

time are given. The problem consists in finding a feasible answer for the line's balancing. Such variation is named SALBP-F after the feasibility objective. When a specific cycle time is required and the number of stations is unknown, the resulting version is named SALBP-1. SALBP-1 is specially useful for the project design of assembly lines. The aimed cycle time is normally a managerial decision, and assembly line designers must find the minimal number of stations that is capable of achieving the necessary production levels. In the adaptation or redesign of assembly lines, the number of stations is usually given. The cycle time can then be minimized to optimize the output levels. The variation that minimizes the cycle time is named SALBP-2. Finally, when both the number of stations and the cycle time are variables, the SALBP-E is defined. In this problem, the objective is to find the number of stations along with the task assignment that produces with maximal efficiency. In other words, the objective of SALBP-E is the minimization of the line's idle time. The production levels are normally restricted to lower and upper bounds depending on the demand requirements. If no such bound exists, assigning all tasks to a single station would result in a line with no idle time, but a very high cycle time. This decision goes against the benefits of flow production designs, no task specialization would be possible.

Table 2.1: Classification of SALBP problems based on the optimization objective.

	Cycle Time (CT)	
	Given	Minimize
Number of Stations (NS)		
Given	SALBP-F	SALBP-2
Minimize	SALBP-1	SALBP-E

According to Scholl and Becker (2006), from the four problem versions, the most explored is SALBP-1, followed by SALBP-2. Several search procedures, however, solves multiple SALBP-F instances in order to optimize a SALBP-1 or SALBP-2. The forth variation, SALBP-E, is the most difficult problem. Minimizing the idle time can be also rewritten as minimizing the product $NS \cdot CT$, that is, SALBP-E is a non-linear problem (Scholl, 1999).

2.5 Solution Procedures

In this section we present a review of the solution procedures used for SALBP. Although GALBP requires other solution procedures, most of the algorithms are based on the simple version of the problem (Becker and Scholl, 2006). The employed solution methods can be classified in heuristics, metaheuristics or exact procedures.

2.5.1 Heuristic Methods

According to a survey on SALBP solution methods, the heuristic procedures can be classified in priority rules procedures and incomplete enumerative procedures (Scholl and Becker, 2006). Heuristics are, in simplified point of view, fast methods of obtaining feasible solutions. Some methods can obtain optimum or almost optimum answers for some instances, but no optimality is guaranteed.

In such priority rule heuristics, tasks are classified by some criteria and their assignment order depends on a construction scheme. These rules are usually based on the processing time and precedence relations of the tasks. Examples of priority rules are the task duration, number of followers, and ratios between the task time and the number of available stations. The first published article on priority rules is due to Helgeson and Birnie (1961) and used the *ranked position weight technique*. Helgeson and Birnie's rule is calculated as the sum of the duration of a task and all of its followers.

The solution construction can be based on two different schemes: the station-oriented and task-oriented. In the station-oriented construction, one station at a time is filled with tasks. For each considered station, the task with higher priority that obeys the cycle time and precedence relation is added to the station. Such procedure is repeated until no more tasks can be assigned. The considered station is said maximally loaded, and the process continues with the next station. On the other hand, in the task-oriented procedure, the tasks with higher priority are assigned to the first station that can support them, also respecting cycle time and precedence relations.

Talbot et al. (1986), Scholl and Voß (1996), and Scholl (1999) presented comparisons between the priority rules. No clear dominance can be drawn, since heuristics present variable results based on the instances. Scholl (1999) and Scholl and Becker (2006) defend,

however, that station-oriented construction schemes are stronger than task-oriented ones.

The second type of heuristics is the incomplete enumerative methods. These methods are mainly based on an exact method, but has a limited search space. The first work published with this method is due to Hoffmann (1963). Hoffmann's heuristic decomposed the problem in one subproblem per workstation. For each station, all combination of tasks are tested, the best solution in terms of idle time is selected and the tasks are removed from the problem. The procedure goes along the stations up to the end of the line.

In Talbot et al.'s survey, 25 years after the publication, Hoffmann's heuristic was still pointed as the best pure heuristic method for SALBP. Further developments on Hoffmann's algorithms were performed. Fleszar and Hindi (2003) adapted the heuristic to work bidirectionally. That is, the method can fill stations from the beginning of the line to the end and the opposite direction. Fleszar and Hindi called their method multi-Hoffmann, because several runs of the algorithm with all forms of filling stations from the beginning or from the end of the line. The heuristic works along with reduction techniques and lower bound calculations to prove the optimality of answers.

Bautista and Pereira (2009) constructed an incomplete enumerative method based on dynamic programming. The stations are filled one at a stage of the procedure. Instead of building all the sub-solutions, the algorithm choses a subset of the nodes to continue the enumeration, working along with bounds to prone less promising solutions. Blum (2008) proposes a similar method called beam search. The beam radius determine the amount of nodes that are evaluated and kept to the next level of the search. An ant colony optimization algorithm is used to choose the nodes that are explored.

Scholl and Becker (2006) also mention that any exact procedure running under time restrictions works as an heuristic procedure.

2.5.2 Metaheuristic Methods

Several metaheuristics have been adapted to solve balancing problems. The most used procedure is the genetic algorithm, although tabu search, simulated annealing, ant colony optimization and other methods have also been applied.

The definition of the fitness function is an important aspect on applying metaheuris-

tics to SALBP. SALBP-1 instances present several assignment possibilities with the same number of stations. Using simply the number of stations as a fitness function results in practically no search direction for the algorithm. For this reason, authors also consider the unbalance of workstations in the fitness function to produce meaningful information on the solution quality (Scholl and Becker, 2006). On the other hand, SALBP-2 presents several possible values for the cycle time, so that the fitness function can be easier formulated.

Starting with genetic algorithms, the greater difference between the proposed approaches relates to the encoding used to represent the task assignments. Scholl and Becker (2006) describe as the standard encoding the use of a station label for each task. That is, the chromosome has one gene per task. Each gene stores a value representing in which station the task is assigned. This encoding is easily prone to unfeasible balancing so that solutions are penalized (Anderson and Ferris, 1994) or rearranged using heuristics (Kim et al., 2000). Kim et al. (2000) uses Helgeson and Birnie (1961)'s heuristic to translate the chromosome information to build only feasible individuals.

Other intuitive form of encoding is based on the task's ordering (Ajenblit and Wainwright, 1998; Sabuncuoglu et al., 2000). A chromosome can be represented by a gene for each task. Each gene contains the number of a task. Then, the crossover and mutation operations can consist on ordering procedures. The chromosome can be translated into a balancing by assigning the tasks in order.

Falkenauer and Delchambre (1992) modeled the problem using a group encoding. This encoding has two chromosomes, one representing tasks and other for the stations. The task chromosomes works as the standard encoding, while the station chromosome orders the workstations. The genetic operations are applied to the station chromosome, indirectly affecting the task chromosome. In the crossover, the content of entire stations are copied to the task chromosome, resulting in repeated tasks. A correcting procedure excludes tasks that do not obey the precedence relations. Finally, an heuristic completes the solution after the excluding procedure. Although the algorithm consists in three parts and needs correction heuristics, the authors points to a greater search range. Due to the operations on the station chromosome, several task assignments are influenced in the genetic operations.

Other genetic algorithms presents an indirect encoding for the assignment of tasks. Gonçalves and De Almeida (2002) use a chromosome containing the priority values for the tasks. The assignment is then made using a priority rule. Further local improvement searches are applied to the individuals. Bautista et al. (2000) present an encoding of priority rules. The genes control the priority rules that are used in the decoding of the individuals. These indirect approaches have the advantage of easily producing feasible individuals.

Sikora, Lopes, Lopes and Magatão (2016) present an indirect encoding based on a feasibility list for each task. The chromosome presents one gene for each task, which stores a position in a list. A list of feasible assignment options is generated and actualized for each task dynamically. The translation is performed in the tasks' order. The assignment of each task influences the feasibility list of their followers, so that only feasible sequences are obtained. Further procedures are used to focus the search in promising regions and eliminate proved non-optimal assignments.

A second metaheuristic, tabu search improves answers by transforming solutions into similar neighbor configurations. The tabu search for SALBP is defined based on task's shifts or swaps. The procedure looks for promising solutions by reassigning or swapping tasks. When all possible swaps are considered before choosing the best, the algorithm is said best fit. The first fit, on the other hand, performs the first profitable swap or shift found. The tabu list management is made in terms of task-station assignment possibilities.

The difficulty of differentiate solutions from SALBP-1 also is present on tabu search. Scholl and Voß (1996) presents a metaheuristic for SALBP-2 and uses to solve SALBP-1 instances iteratively. Chiang (1998) and Lapierre et al. (2006) modeled the objective function for SALBP-1 also in terms of the maximally loaded stations. This way, the solutions with better packed stations are better evaluated during the procedure.

Other methods such as simulated annealing (Heinrici, 1994), ant colony optimization (Bautista and Pereira, 2002), differential evolution algorithm (Nearchou, 2007) are also used as solution procedures for ALBP.

2.5.3 Exact Methods

According to Scholl and Becker (2006), the most successful exact solution methods for SALBP are either dynamic programming or branch-and-bound procedures.

The first enumeration procedure based on dynamic programming is due to Jackson (1956) and was improved by Held et al. (1963). Jackson's algorithm is a breadth-first search that enumerates all the forms of assigning tasks to workstations. A dominance rule was used to eliminate partial solutions that were dominated by other assignments. The Jackson's dominance rule is still used in the more recent procedures.

The recent dynamic programming based methods also employ heuristic or meta-heuristic procedures in order to avoid the complete enumeration and have been mentioned in Section 2.5.1.

Just as the constructive heuristics, the exact procedures can be divided in station or task oriented. Johnson (1988)'s algorithm "FABLE" is an example of the task oriented method. Each node in the branch-and-bound tree represent a task, while the branches represent the options of stations in which the task can be assigned to. Johnson (1988) also developed and presented bounds and dominance rules applied during the procedure.

The station oriented schemes represent a station loading in each node. One clear advantage of this method is that all considered nodes consist in fully loaded stations. If there was an assignment with enough idle time to add another task, it could be eliminated, because it is dominated by another partial solution that added the task. Hoffmann (1992) presented a station oriented branch-and-bound procedure called "EUREKA". EUREKA had less resources comparing to FABLE, but the performance of both algorithms are similar.

Scholl and Klein (1997) combined the advantages of FABLE and EUREKA along with extra bounds and a more efficient search procedure into an algorithm called "SALOME". According to the computational studies of Scholl and Klein (1999), SALOME clearly outperformed the previously published methods. A SALOME version for the SALBP type-2 (SALOME-2) is due to Klein and Scholl (1996).

More recently, Sewell and Jacobson (2012) presented an algorithm called Branch-Bound-and-Remember (BB&R) for the SALBP-1. The algorithm solves a bin packing

problem for each node to obtain a lower bound for each of the subproblems. The Remember part of the algorithm is due to the storage of already calculated bounds and partial solutions. Therefore, the procedure identifies whether some assignment has already been considered and prone repeated nodes. The BB&R was the first algorithm that could solve and prove the optimal answers of all instances of Scholl (1993)'s dataset (described in Section 2.8).

According to Battaïa and Dolgui (2013), standard general solvers are also used for balancing problems. Although they are not specially designed for ALBP and not as efficient as the specialized algorithms, general solvers are an option when no other solution method is available.

2.5.4 Method Comparison

Some publications focused on surveys of solution methods for the Simple Assembly Line Balancing Problem (Baybars, 1986; Scholl and Voß, 1996; Scholl, 1999; Scholl and Klein, 1999). In these surveys, however, the computational comparisons are restricted to either the exact methods or heuristics.

Pape (2015), on the other hand, presents a comparative study of a broad aspect of different solution methods. Pape compared the performance of genetic algorithm (Falkenauer and Delchambre, 1992; Sabuncuoglu et al., 2000), differential evolution algorithm (Nearchou, 2007), ant colony optimization (Bautista and Pereira, 2002), tabu search (Scholl and Voß, 1996; Lapierre et al., 2006), MultiHoffman heuristic (Fleszar and Hindi, 2003), bounded dynamic programming (Bautista and Pereira, 2009), beam search (Blum, 2008), and branch-and-bound (Scholl and Klein, 1997; Scholl and Klein, 1999) for the solution of SALBP-1. The most efficient algorithms are dedicated branch-and-bound or dynamic-programming procedures that use the problem structure to determinate the search procedure.

Furthermore, the results obtained from Sewell and Jacobson (2012)'s BB&R proved to be superior to any previously published method. Therefore, the branch-and-bound procedures seem to be by now the most adequate method to solve balancing problems. A reason for that might be the assignment priorities in the search procedure. The knowl-

edge gathered in years of research on the most promising regions is implemented on the algorithms so that they first explore these areas. Along with cuts, priority rules, and bounds, the enumerative methods seem to be superior than the random characteristics in the search used by metaheuristics.

2.6 Problem Variations and Combinations

The simplification hypotheses used for the Simple Assembly Line Balancing Problem are usually not representative of real assembly lines (Falkenauer, 2005). Several variations of the problem appear by relaxing one or more of these hypotheses. Furthermore, solving the balancing problem along with other integrated systems presents very promising opportunities of more global optimizations.

This section is not supposed to review all the variations nor all the contributions within any one of them. On the other hand, it serves as an introduction to the different applications that are related to balancing problems. Firstly, all the simplification hypotheses presented in Section 2.3 are discussed and examples of problems without the hypotheses are given.

- (SH-1) All stations are equally equipped and manned.

The first simplification hypothesis (SH) assures that the balancing problem does not have to deal with workstations and workers availability or resources. That is, there are no other restrictions such as spacial, ergonomic, and effort limitations.

Bautista and Pereira (2009) presented the Time and Spaced Constrained Assembly Line Balancing Problem (TSALBP) as a balancing problem in which stations have a finite physical space. Each task requires a defined amount of space in the workstation, representing the necessity of assembly pieces and equipment of each task. Therefore, the TSALBP treats each station differently in respect to their equipment capabilities. Furthermore, the stations can have different amounts of available space for the task distribution.

While TSALBP only considers the space constraint, the Assignment Restricted Assembly Line Balancing Problem (ARALBP) proposed by Scholl et al. (2010) generalizes the resource restrictions. For the ARALBP definition, the processing time, space, or any

other restraint that can be summed can be treated as a resource constraint. In this aspect, the ARALBP generalizes the balancing problem in several dimensions: not only the processing time must be balanced, but all the important resources of a line.

A third example on limitations on the assignment of tasks to workstations is due to Otto and Scholl (2011). Otto and Scholl presented the ErgoSALBP, the assembly line balancing problem that considers the ergonomic risks of the task's assignments. The authors argue that using resource constraints to sum the ergonomic characteristics of each task does not treat the problem as it appears in the industry. There is a trade-off between production and the ergonomic risks. Therefore, Otto and Scholl (2011) proposed a multi-objective model to optimize the production considering the cost and downfalls of the tasks' assignment on the workers.

Treating the assignment of workers to stations generally disagrees with SH-1. Fattahi et al. (2011) and Yazgan et al. (2011), for instance, worked with multi-manned stations. That is, more than one worker can be assigned simultaneously to a single workstation. The multiple assignment is more common on the assemblage of large products (buses, trucks, etc) and interference between workers may also have to be considered (Boysen et al., 2008).

Another problem related to the difference between station resources is due to the Assembly Line Worker Assignment Balancing Problem (ALWABP) presented by Miralles et al. (2008). In ALWABP, not only tasks must be assigned to workstations, but also heterogeneous workers must be assigned to workstations. Modeling the decision of the worker allocation can be used to include workers with different abilities and disabled workers together in an assembly line. Similar works are due to Costa and Miralles (2009), who considers the job rotation on assembly line; Moreira et al. (2015), by modeling the inclusion of disabled workers on industrial assembly lines; and Sungur and Yavuz (2015), who modeled the problem based on levels of qualification for the workers in a hierarchical approach.

Sikora, Lopes and Magatão (2017) presented another variation of the balancing problem along with the assignment of workers: the Traveling Worker Assembly Line Balancing Problem (TWALBP). Their model can be used when there is more stations than workers, so that one worker can be responsible for multiple stations. Sikora, Lopes

and Magatão (2017) integrated the balancing and working assignment problems with a traveling salesman problem. In this way, the movements of the workers are measured and their routes are integrated in the line's cycle time. A preliminary study with rougher estimative for the movement times was presented in Sikora et al. (2015).

The cited works can exemplify cases in which stations are not equally equipped or manned. They require, therefore, different formulations and solution methods other than the specific for the simple version of the problem.

- (SH-2) Task processing time is independent of the workstation.

One important application of balancing problems for which the processing time is dependent of the workstation is related to the equipment choice. Rubinovitz et al. (1993) presented the Robotic Assembly Line Balancing Problem (RALBP), followed by further development on solution methods (Kim and Park, 1995; Levitin et al., 2006; Daoud et al., 2014). In robotic assembly lines, different models of robots can be used to perform operations. Tasks, therefore, can present different processing time depending on the type of robot it has been assigned.

Lopes, Sikora, Molina, Schibelbain, Rodrigues and Magatão (2017) proposed a formulation for the optimization of the balancing of welding lines. Welding procedures can require a variety of types of robots or welding guns. Not only the processing time of tasks depends on the robot and gun used, but there may be more than one way to perform each task.

Other example can be given by worker's learning curves. Although the processing time is not dependent on the workstation, it varies with the time. Chakravarty (1988) considered that the workers learn dynamically how to better perform their tasks. Therefore, the processing time is dependent on how longc the line was implemented.

- (SH-3) Any station is able to perform any task.

The third simplification hypothesis is frequently overruled in practical cases. Is not uncommon that tasks require an especial equipment or position to be performed. Therefore, some assignments may be unfeasible. An example given is a task that must be

performed under a car. It is wise to restrict its assignment to only stations whose access to the base of the car is granted (elevated stations, for instance).

These incompatibility or zoning restrictions have been treated in several different contexts (Scholl et al., 2010; Sikora, Lopes, Lopes and Magatão, 2016; Sikora et al., 2015; Sikora, Lopes and Magatão, 2017). Scholl et al.'s ARALBP, for instance, also presents other possible restriction to the assignment of tasks. According to the ARALBP definition, tasks could be linked (assigned to the same station) or incompatible (not assigned to the same station), or have a minimal or maximal distance between their assignment.

- (SH-4) The assembly line is serial, no feeder or subassembly lines are considered.

The most common form of line apart of a serial line is the U-line arrangement. In U-lines, the beginning of the line is close to the end. Workers can cross between both sides of the U, giving more liberty to the assignment of tasks. The balancing on U-lines was firstly presented by Miltenburg and Wijngaard (1994), who showed that the balancing on U-lines is better or equal to the serial lines. This result, however, considers there is no movement time between both sides of the U-line. A formulation for the U-line and also any generic layout with deterministic movement times was given by Sikora, Lopes and Magatão (2017).

Another variation considers the parallelization of workstations. Bard (1989) showed that the parallel stations can reduce the effect of the movement time of pieces between stations (dead time). If the movement time represents a large portion of the available cycle time, the workers or robots are expected to be idle while waiting the passage of pieces. Bard argued that a parallel station producing two pieces every two cycle times dilutes the movement time, increasing the overall production of the line. In a following work, Lopes, Michels, Molina, Sikora and Magatão (2016) presented a model for the design of robotic welding line considering the effects of parallel stations.

- (SH-5) The assembly line is designed for the production of a single product.

As opposed to the single product, an assembly line can produce either multi or mixed models. The multi-product lines produce batches of similar products, usually with

a set-up time between each batch. The mixed-model production, on the other hand, is adapted to work with a mix of products (Boysen et al., 2008).

The first attempt on balancing assembly lines with multiple products is due to Thomopoulos (1967) and Thomopoulos (1970). Multiple products can present different operations and different processing times. This way, the assignment of tasks and the cycle time of the line has to account that pieces of different products may proceed at different rates in the line.

The balancing of mixed-models is usually integrated with the scheduling of work-pieces throughout the line. Several works solve these problems simultaneously (Sawik, 2012; Öztürk et al., 2015; Lopes, Sikora and Magatão, 2016), even with the possibility of considering and/or allocating buffers. Lopes, Sikora, Michels and Magatão (2017a) presented a cyclical formulation for the scheduling of products integrated with the balancing and allocation of buffers among the assembly line.

- (SH-6) All problem's parameters are deterministic (processing time and precedence relations).

The most common form of stochastic parameters in balancing problems is at the processing time. The non-deterministic processing time imply on a balancing that may assure with a certain probability the cycle time of a line (Kottas and Lau, 1973). The stochastic characteristics have been extensively mixed with other types of balancing problems, such as U-lines (Baykasolu and Özbakr, 2006) and two-sided lines (Özcan, 2010), for instance.

- (SH-7) The cycle time of every station is the sum of the assigned tasks' processing time.

One example of model that does not respect the simplification hypothesis 7 is assembly lines with set-up between tasks. The set-up between pairs of tasks increases greatly the difficulty of the problem. Normally, balancing tasks does not require the sequencing or scheduling of tasks assigned to a workstation. This simplification is not true in the presence of set-up times: depending on the order tasks are assigned, different working loads are obtained. Scholl et al. (2008), Andrés et al. (2008), and Scholl et al. (2013)

present models and solution procedures for the assembly line balancing problem with set-up times.

The movement time between welding points presented in the balancing of a welding line in Lopes, Sikora, Molina, Schibelbain, Rodrigues and Magatão (2017) can also exemplify a case in which the cycle time of a station is not the same as the sum of the processing time. In this special case, the movement time can actually account for more than half of the cycle time.

A final cited variant is not directly a relaxation of the simplifications hypotheses. The variant, however, is a particularity that can be found in some real assembly lines. Sikora, Lopes, Schibelbain and Magatão (2017) showed that the modeling of lines containing multiple identical or repetitive tasks can be simplified. Such identical tasks are common in welding procedures, for which tasks are similar in terms of processing time and precedence relations. Sikora, Lopes, Schibelbain and Magatão (2017) presented a formulation that used integer variables to treat groups of tasks, instead of the standard formulation that treats each tasks individually using binary variables.

2.7 Applicability of ALBP Research in Practice

According to Falkenauer (2005), there is a large distance between the research results on Assembly Line Balancing and what is performed in the industry. Recent surveys present reasons for the gap between the application of optimization methods addressed to assembly lines in the industry. Becker and Scholl (2006) focus on General Assembly Line Balancing Problems (GALBP). According to the survey, several real-world conditions (such as equipment selection, station paralleling, U-shaped lines and mixed-model lines) have been identified and modeled. However, sophisticated solution methods for GALBP are not in the same pace to the ones of SALBP.

The lack of easy-to-use software containing state-of-art optimization methods is pointed in a survey by Boysen et al. (2008). According to the authors, only 15 out of more than 300 considered articles present results of a real-world assembly line. Even though the work of Arcus (1965) and Bartholdi (1993) presented a software interface for their heuristic methods, real case scenarios (Agnetis et al., 2007; Battini et al., 2007; Bautista

and Pereira, 2007; Lapierre et al., 2006) are mostly used for testing the solution method.

The more recent survey of Battaïa and Dolgui (2013) shows that the modeling and solution methods for GALBP are evolving: the balancing problem is often treated along with related problems. The survey showed a recent focus on robust optimization due to the uncertainty of the input data.

Further practical considerations for ALBP are due to re-balancing scenarios. According to Falkenauer (2005), the reallocation of tasks of an operating line has several restrictions due to the difficulty of changing previously implemented decisions. Boysen et al. (2007), in their survey, pointed that re-balancing is usually modeled either by assignment restricted models (Bautista and Pereira, 2007; Scholl et al., 2010; Sternatz, 2014) or by cost-oriented models (Gamberini et al., 2009; Makssoud et al., 2015; Zha and Yu, 2014). That is, the re-balancing aspect of assembly lines is treated as a restriction on the final solution. Either the final solution is strictly prohibited to change some predetermined assignments (such as equipment locations or complex tasks) or the cost of the changes are included in a cost calculation. If the benefit is greater than the implementation cost, the final solution considers the new line design, otherwise, the answer maintains the actual assignment.

From what have been presented in this master thesis, great part of papers treat the modeling and solving of balancing problems. In fact, studies on the applicability of solution methods are only a few. It is the objective of this work to show that the implementation conditions may also be considered in the optimization process. One example of effort developed to the application of the research in real assembly lines is due to Otto and Otto (2014), who present a method of constructing the precedence relations of the tasks of an assembly line gathering data from multiple sources. Their algorithm uses historic data of assembly lines to build an approximative precedence diagram. The simplification reduces the warm-up period of study and data acquisition of projects.

The literature on ALBP is extensive: in January 2017, a search for the term Assembly Line Balancing Problem at Scopus, the largest database of peer-reviewed literature (www.scopus.com), returned 1186 documents. Although much has been published in the last 60 years, little is focused in the implementation of the resulting optimal balancing in real lines. In the most recent surveys on ALBP (Baybars, 1986; Becker and

Scholl, 2006; Scholl and Becker, 2006; Boysen et al., 2007; Boysen et al., 2008; Battaïa and Dolgui, 2013), it is noted a strong focus on solution methods, **while nothing is implied about how to implement the outcome of the research effort**.

Assembly lines usually work uninterruptedly during weekdays, reducing the time available for interventions. The redesign of lines is often limited to scheduled maintenance stops at collective vacations. During production periods, however, a project that requires significant modifications in the line might be put on hold. The interventions on assembly lines take time and incur on moving, installing, and training costs. While models focus in the final configuration of the balancing, little is published about implementing the necessary changes, a gap addressed in the presented work.

2.8 Data Sets

Datasets are collections of problem instances used to test solution methods. They are important as a benchmark to not only test but also compare different algorithms. Datasets need to be structured and representative of real instances. Therefore, some researchers invested part of their work into understanding the structure of problems and deriving meaningful data for others to come.

The initial instances presented in the first publications on assembly line balancing problem were merely illustrative. Jackson (1956) and Bowman (1960), for instance, presented a problem with 11 and 8 tasks respectively, that could be easily solved by hand. At the date, however, the computational equipment and the technical development were only at its birth and only increased during the following years.

Further publications on ALBP usually were followed with new instances created or adapted by the authors. Some were inspired by real assembly lines, while others were created to provide instances for their methods. Until 1986, there were no efforts on gathering instances and creating a unified dataset for the balancing problems. Talbot et al. (1986) developed a dataset based on 12 precedence diagrams from 8 to 111 tasks. The dataset is made for SALBP-1 and, along with multiple values for the cycle time of the line, resulted in 64 generated instances. Hoffmann (1992) also proposed a dataset based on the same 12 precedence diagrams, but chose special values for the cycle time in

order to create harder instances resulting in 50 problems.

In a further development, Scholl (1993) gathered the datasets of Talbot et al. and Hoffmann along with larger precedence diagram into a dataset of 269 instances. The instances ranged from 8 to 297 tasks and were also adapted to SALBP-2 and SALBP-E (Scholl, 1999). Scholl's dataset had all its instances solved to optimality only in 2012, when Sewell and Jacobson (2012) published their Branch-Bound-and-Remember algorithm for SALBP.

The first effort of creating a systematic and organized dataset is due to Otto et al. (2013). Otto et al. observed the structure of several real assembly lines to identify similarities and the general structure of precedence diagrams and task's duration time distribution.

According to Otto et al. (2013), real assembly lines frequently present two special structures on their precedence graphs. There are tasks that have several predecessors and/or successors. That is, some tasks are bottlenecks in the assembly process. Another usual characteristic is chains of tasks. Chains are formed when at least three tasks have only one predecessor and one successor, resulting in an aligned chain in the precedence graph.

Otto et al.'s analysis on the tasks' distribution time pointed that most of the tasks have a low processing time compared to the cycle time of the line. This distribution was named **Peak at the Bottom** and was used to propose the dataset's instances. Other distribution variation used for the dataset is a **Bimodal** distribution. For this distribution, tasks with lower processing time are considered along eventual tasks with large processing times. Both the Peak at the Bottom and the Bimodal distributions can be found in real lines. Notwithstanding, the authors proposed a third time distribution: the **Peak in the Middle**. For this distribution, the tasks' processing time are centered on half of the cycle time. Although this distribution is not found in practice, the resulting instances showed to be the most challenging problems of the dataset.

In total, Otto et al. (2013) proposed 2100 instances based on the parameters described in the last paragraphs. The dataset is structured in small, medium, large, and very large instances. Small instances contains 20 tasks, while medium contains 50 tasks, large 100 tasks, and very large 1000 tasks. Seven structures of the precedence diagrams

were used. They are based on a probability of the presence of chains and bottlenecks, along with different values for the order strength (OS) of the graph. The used diagrams can be named BN-0.2, BN-0.6 (with task bottlenecks and OS of 0.2 and 0.6), CH-0.2, CH-0.6 (with chains of tasks and OS of 0.2 and 0.6), MX-0.2, MX-0.6, and MX-0.9 (mixed diagrams with both chains and bottlenecks and OS of 0.2, 0.6, and 0.9). Three different time distribution for the task duration are used: Peak at the Bottom (PB), Peak in the Middle (PM), and Bimodal (BI). The dataset is based on 25 randomly generated instances for each combination of the seven precedence diagrams and three time distribution, resulting in 525 instances for each value of the number of tasks.

For the assembly line balancing problem variations, no structured and widely used dataset is found. Several authors usually adapt Scholl's or Otto et al.'s dataset with the particularities of their problems. Some of the published datasets can be found in the website www.assembly-line-balancing.de.

Once the Assembly Line Balancing Problems and its variations have been reviewed, the next chapter proposes a novel class of problems and describes its relation to the assembly line balancing.

Chapter 3

Problem Definition

This chapter presents the definition of Implementation Problems. The difficulties and boundary conditions for changing the configurations of assembly lines are discussed. These conditions are used to define the Assembly Line Implementation Problem (ALIP), along with its variations and ramifications.

3.1 The Implementation Scheduling Problem

The implementation scheduling concerns the process of transforming a system from an initial to a final configuration. Changes or implementations must be performed to change the system into a goal form. The scheduling is then defined as the way the implementation occurs: which changes are performed and when. An implementation problem is only defined when there are resources or implementation constraints. That is, the implementation is not trivial and its quality depends on the way it is performed.

A classical optimization problem that deals with the implementation of systems is the Project Scheduling (Malcolm et al., 1959; Kelley, 1961; Fulkerson, 1961). According to Baker and Trietsch (2009), project scheduling tools are used to plan and schedule large and non repetitive projects. A project consists of well defined tasks or activities that must be performed. These activities are subject to logical constraints, such as precedence between two activities. In the resource-constrained version of the problem, activities are both subject to precedence and resource restrictions (Wiest, 1964; Pritsker et al., 1969;

Schrage, 1970).

The Project Scheduling consists in scheduling the execution of tasks so that the project duration is minimized (Baker and Trietsch, 2009). A usual form of representing a scheduling answer is based on a Gantt Chart. Figure 3.1 contains a sample solution for a Project Scheduling Problem with 8 activities (or tasks). Activities require an implementation time and precedence relations between tasks can be observed, for instance, activity 4 only starts after the completion of activity 2. Note that such problem formulation does not consider any operation during the implementing process. The system can operate only after the project is completed. That is, the level of occupation of the systems goes from 0% during the project phase to 100% in the production or use phase.

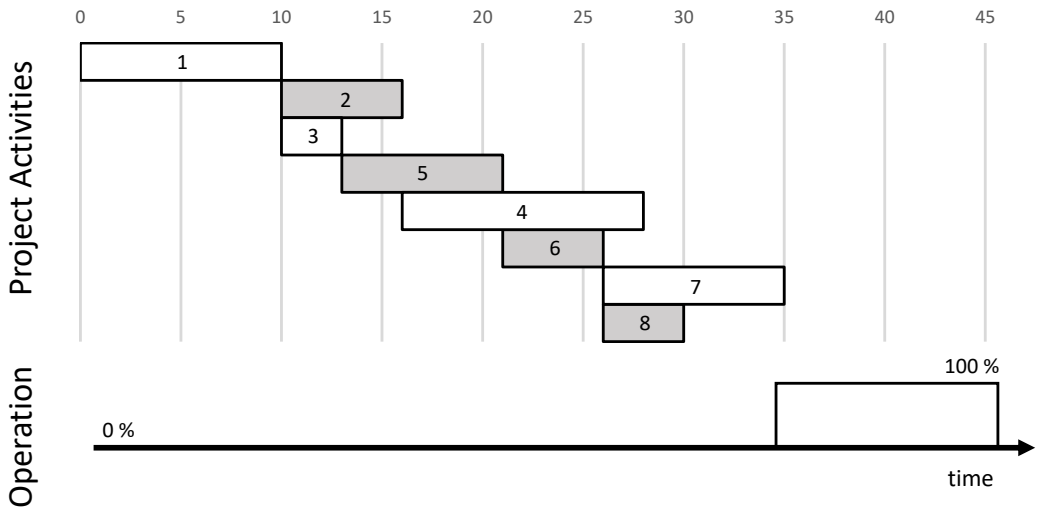

Figure 3.1: Example of a Gantt Chart for the Project Scheduling along with the operation level during time. The operation occupation of the system goes from 0% at the implementation to 100% at the end of the project.

Hartmann and Briskorn (2010) presented a recent survey on variants and extensions of the Resource Constrained Project Scheduling problem. According to the authors, the project scheduling has been generalized in terms of the activity concepts, the temporal constraints, the resource constraints, and objective functions. The activity concepts are related to the scheduled tasks: they can be preemptive; request resources varying with time; have setup times; have multiple forms of being performed; present trade-off problems; etc. The temporal constraints are related to the temporal precedence between

tasks: there could be a minimal or maximal interval between the scheduling of two tasks; release or due dates could exist; intervals in which tasks can be scheduled; or even logical relations (and, or, xor, etc) between the assignment of tasks. The resource constraints relate whether resources are renewable or non-renewable; cumulative; dedicated; continuous or discrete; constant or varying with the time. Finally, the objective functions could be time-oriented; robustness-oriented; rescheduling-oriented; resource-oriented; cost or profit-oriented; or oriented to multiple factors.

None of the variations described by Hartmann and Briskorn (2010) considers explicitly an already operating system. The schedule respect precedence between tasks with different characteristics that must respect resource constraints based on several possible objectives. However, scheduling the operations while the system is producing an output is not mentioned as a variation. One could, however, consider the production and the resulting restrictions of the system during the implementation process. Throughout an implementation, the system passes through intermediary states. In a re-projecting or adaptation effort, one system may require operational restrictions for these intermediary configurations resulting in a new class of problem. Thus, in this master thesis, the Implementation Scheduling Problem is proposed.

An operational job shop system can be used to exemplify the scheduling of running systems. A modification, maintenance or reallocation of machines can be observed as activities that must be performed in an implementation project. These changes on the system have influences on its output levels, once some machines may be stopped in the process. Figure 3.2 contains an exemplified Gantt Chart for such application. The job shop runs in an initial configuration until the first implementation phase takes over (Activities 1-3). During the changes, the production levels are affected due to the interventions on the system. After the first implementation phase, the changes lead to the second system configuration. The changes could either be performed in a single phase implementation just as the Project Scheduling (Figure 3.1) or the implementation can be divided in several stages. In a multi-stage implementation, intermediary production phases can be present between multiple implementation phases (see the operation levels at Figure 3.2). Notice that the operational level in Figure 3.2 accounts only for the occupation of the system (or the availability of the machines, for instance), and not the

production level itself.

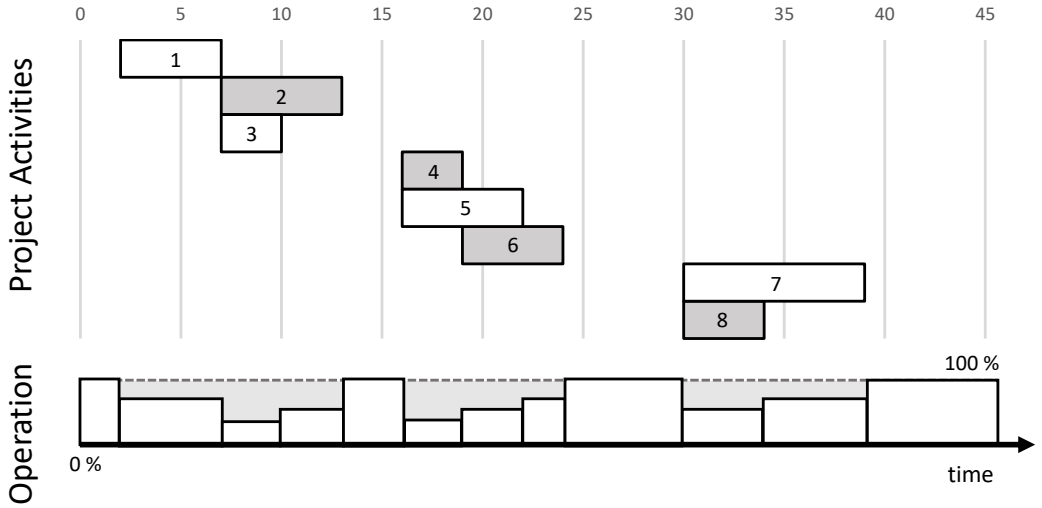

Figure 3.2: Example of a Gantt Chart for a generic Implementation Scheduling Problem along with the operation level during time. The output of the system is not null during the implementation, but is affected by the interventions (gray areas).

The implementation scheduling, just as the project scheduling, has the objective of finding the best way to perform the activities of a project. In the project scheduling, the system's output during the implementation is not important. The project is supposed to start at the end of the implementation. The implementation scheduling, however, considers the operation of the system during the implementation. A minimal production rate or time windows for the delivery of products could be operational conditions of our job shop example.

Figures 3.3 and 3.4 represent special cases of the Implementation Scheduling Problem. There are systems in which changes may not require a significant amount of time to be implemented. The reallocation of people to activities serves as example. The time required for a person to change an activity can be neglected. There could be other associated costs such as the quality of the work, preference or the process learning, but the assignment itself does not require a significant amount of time. Such problems are represented by Figure 3.3: changes occur punctually and separate different system's configurations.

A second special case, Figure 3.4, shows a variation in which interventions require the system stoppage. Flow lines are examples of such systems, once the interruption of

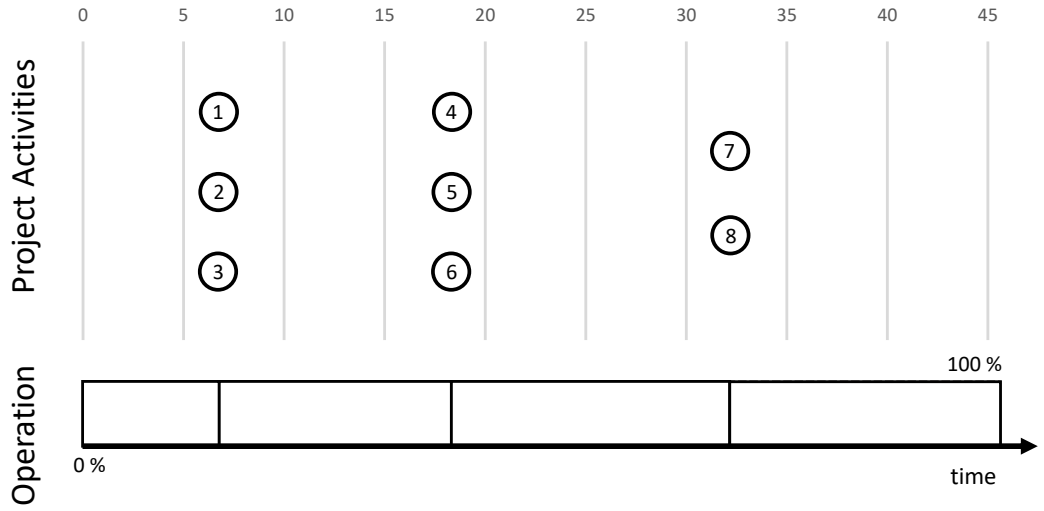

Figure 3.3: Example of a Gantt Chart for an Implementation Scheduling Problem with non-time requiring changes. The operation level during the project duration is always considered 100%, once the changes does not take a considerable amount of time.

any station results in the interruption of the system's entire flow.

Figure 3.5 contains a representation of different scheduling problems in a bar diagram. Grey areas represent unproductive time, while operational phases are shown in white. Diagram A represents the Project Scheduling. The implementations are performed with a non operational system. The system is only active at its final configuration. Diagram B presents a generic Implementation Scheduling Problem. Differently to the Project Scheduling, the system can be used during or between the applied changes. Interventions on the system, however, affect its occupation. A job shop with some of its machines being reallocated can be used as an example. The job shop starts with an initial configuration and passes through other states every time the layout changes. Reallocating a machine requires its stoppage, interfering in the system's output during the operations.

Diagrams C and D from Figure 3.5 represent special cases of the Implementation Scheduling Problem. In Diagram C, the changes are said timeless, the implementation steps do not require significant amount of time. Finally, Diagram D is the variation in which a system must be stopped to perform the implementations. If the implementation occurs in one step, the problem is similar to the Project Scheduling (Diagram A). In a multiphase implementation, this variation alternates between production and implemen-

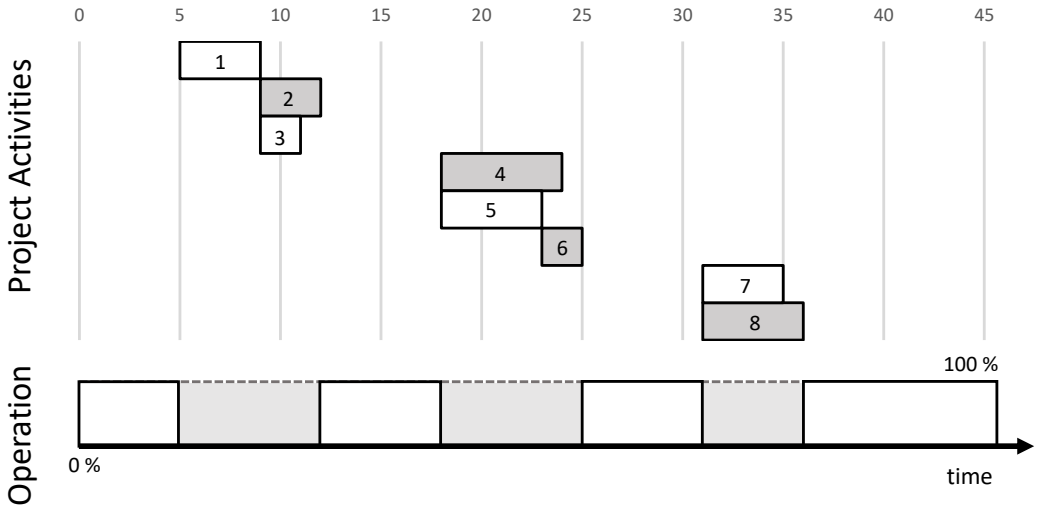

Figure 3.4: Example of a Gantt Chart for a Implementation Scheduling Problem with interruptive interventions. The operational level during the implementation phases is null due to the necessity of stopping the system.

tation phases.

The Implementation Scheduling consists of the problem of scheduling project activities along with the operational restrictions. Using this concept, the Project Scheduling can be seen as a particular case of the Implementation Scheduling when no functional restrictions are present. Both the implementation and the production restrictions depends on the nature of the problem. In Section 3.2 the particularities and simplifications of implementations on assembly lines are discussed.

3.2 Implementing Changes on Assembly Lines

Companies that assembly products usually work with a series of flow shop systems. Buffers are usually present between different assembly lines to absorb possible machine breakdowns or maintenance stops. These buffers are, however, fit to compensate for a few hours of production. A long lasting stoppage in one assembly line would affect the product flow of the entire production system.

Interventions and optimizations on assembly lines cannot be performed indiscreetly. Once a long stoppage of one line stops all the company's related lines, strong restrictions

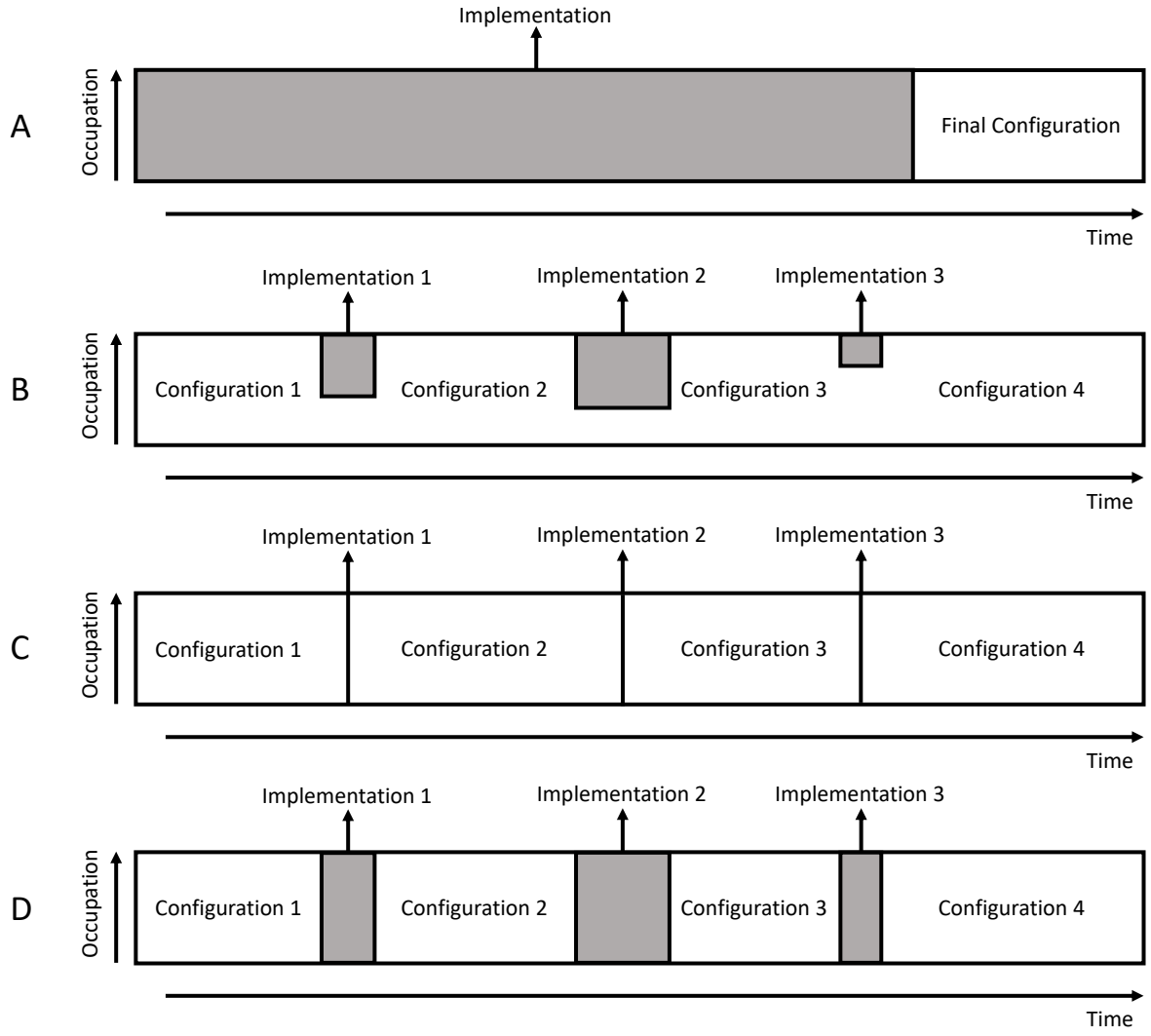

Figure 3.5: Representation of Implementation Scheduling Problems with different characteristics. Diagram A presents a Project Scheduling Problem. Diagram B presents a generic Implementation Scheduling Problem. Diagrams C and D present special cases of Diagram B. In Diagram C, the changes do not require a significant amount of time to be implemented. For the problem represented by D, the operational level during the implementation phases is null due to the necessity of stopping the system to perform the changes.

on the available time for interventions exist. Furthermore, it is common that companies operate 24 hours a day during workdays. So that the opportunities to perform changes are restricted to weekends, holidays or programmed maintenance stoppages. Long implementations are possible to be performed once a year on maintenance stoppages. For more regular intervals, the interventions must be restricted to weekends.

Because of the strong time window restrictions, the scheduling of phases is obvious. In order to not reduce the company's overall production, interventions must be scheduled either once a year or at weekends. A long intervention can be scheduled by a Project Scheduling procedure. On the other hand, small weekend interventions results in a stepwise implementation consisting of an Implementation Problem. The start and end points for the implementation periods are fixed: the line produces in workdays and the transition occurs at the weekend or holidays, as exemplified in Figure 3.6.

M	T	W	T	F	S	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	24	27	28
29	30	31				

Figure 3.6: Calendar of operations. Implementations can be performed on weekends or holidays.

The Implementation Problem for Assembly lines is then simplified as two defined phases. The **production phase** occurs during workdays, and must respect cycle time and process restrictions. The **implementation phase** is responsible for the changes of task assignments. For the interventions, the feasibility of the changes must be valid. That is, the proposed changes must be implementable in the defined time window.

The restrictions that guarantee the feasibility of the implementation depends on the nature of the changes and the parallelization of work. If there are any order relations between the changes, such as precedence relations and multiple workers are used to implement them, it is necessary to **schedule** the changes. If no ordering between tasks is observed, but they are performed by multiple workers, the resulting problem can be

treated as a **bin packing problem**. The bin packing concept is used to assure that the changes can be divided among the workers considering that tasks must be performed individually. A third variation occurs when only one worker perform the changes or the level of coordination and division of tasks do not require the scheduling of the process. In this case a simple **resource constraint** is enough to assure the feasibility of the implementation.

The reassignment of a task on an assembly line may be costly depending on what is necessary to assure its execution in other station. The most common effort is to **actualize the process sheets** after any modification. However, the update of the process sheets alone does not present any difficulties for a new balancing. On the other hand, when a task requires some special equipment such as a drill, the necessary machinery must be moved to the new station. The **reallocation of equipment** is usually the reason a direct implementation is timely costly and justifies the division in a stepwise implementation. In robotic lines, there is an associated effort of **reprogramming robots**. Although some part of the work can be done off-line, moving and adjusting tools and testing new parameters and trajectories requires the line for their application. Another implementation cost could be observed in the **training of line operators**. Changing an employee's assignments may require some learning time, during which the production performance is reduced. The training of workers has effects on the production phase, so that for this particular problem the scheduling effects of the phases may also be consider.

Apart from the training problem, the mentioned necessary changes can usually be modeled using a resource constraint. The models and formulations presented here are based on the assumption that workers can coordinate their operation and no scheduling for the changes are needed. This assumption is further discussed in Section 4.3.

3.3 The Assembly Line Implementation Problem (ALIP)

The Assembly Line Implementation Problem (ALIP), herein proposed in this master thesis, is a special case of the Implementation Problem discussed in Section 3.1. The problem consists in finding the best way to change the task assignments from a given initial to a final configuration. The initial configuration is given and usually represents

the actual state of the system. The final configuration is a goal condition, that can be obtained heuristically or via an optimization method. The final configuration can be previously solved and given as a parameter or a simultaneous approach can consider the final configuration as a model variable. Section 3.5 further discusses about the final state.

The implementation is divided in feasible interventions or implementation phases. One simplification from the general Implementation Scheduling Problem is that the implementation occurs on periods in which the line is stopped. That is, the implementations are already restricted to specific time windows. There is no need to schedule the start and end times of the intervention periods. The available workable time must be given as a problem parameter, usually representing weekends, holidays or an unoccupied work shift.

The problem representation can be seen as in Figure 3.7. The implementation process is divided in two different concepts. The intermediary production phases are represented with squares. These phases are also called stages. The second concept is related to the change phase. The transitions are represented by the arrows in the system. They refer to the period in which task assignments are changed.

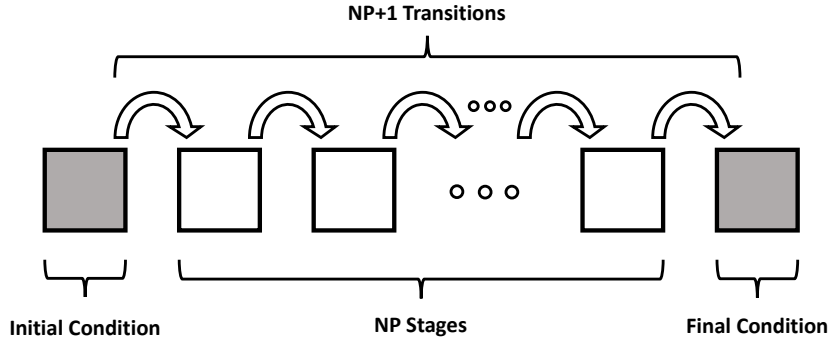

Figure 3.7: Diagram for the division of the problem in production (stages) and implementation (transition) periods.

Each phase on the representation has its restrictions. Stage or production phases must obey the line's production conditions. Throughout the running hours, the assignments should respect the occurrence, precedence and cycle time restrictions. That is, the necessary amount of correctly produced pieces must supply the company needs. For the transitions, an implementation answer must assure the performed changes can be

finished within the transition time windows. Therefore, a resource constraint in terms of implementation effort should be observed.

The problem's objective is to define which changes are performed in which phase and all intermediary production conditions. The optimization process can be performed with different objective functions. Section 3.4 discuss the optimization goals and the resulting problem type of each goal.

3.4 Problem Types for the ALIP

Just as we have seen with the Assembly Line Balancing Problem in Section 2.4, multiple objective functions may apply to implementation problems. Different classes of problems are defined depending on which characteristics are set as parameters and which are optimized. In balancing problems, the two characteristics of interest are the number of stations, related to the line's running cost, and the cycle time, linked to the line's throughput.

For the Implementation Problem, several classes of problems can be defined. Globally, the problem can be optimized in terms of the transition periods, production periods, or both. We shall name the classes as ALIP-T for the transition, ALIP-S for the stage or production period, and ALIP-C, referring to a cost optimization. The acronym ALIP without an extension is used throughout the text to refer to ALIP-T. Once the production levels during the implementation are transitory, more importance may be given for the implementation effort.

The classes of problems (T, S, and C) also contain subclasses. Starting with the transitions, two parameters of interest can be observed: the amount of necessary resources and the implementation duration. Table 3.1 summarizes the problem classes. When both the number of periods and the maximal implementation effort are given, the problem consists in finding a feasible answer. Using the same nomenclature as the SALBP problems (see Table 2.1), we name the feasibility problem as ALIP-T-F. A second variation is defined when a given amount of resource is limited per implementation phase and the number of intervention periods is minimized. This objective is equivalent as minimizing the duration of the implementation, resulting in the nomenclature ALIP-T-1. On the

other hand, when the project has a given duration or number of periods and the intent is to minimize the implementation effort, ALIP-T-2 is defined. In this problem, the maximal implementation effort per intervention is minimized. This class of problem can be used for the implementation crew sizing. Both the number of necessary employees or the amount of required work-hours can be minimized using this objective. Finally, when both the characteristics are set as variables, ALIP-T-E is defined. This variation is equivalent to minimizing the implementation total cost, expressed as the product $NP \cdot Effort$.

Table 3.1: Classification of ALIP-T problems based on the optimization objective.

	Maximal Implementation Effort	
	Given	Minimize
Number of Periods (NP)		
Given	ALIP-T-F	ALIP-T-2
Minimize	ALIP-T-1	ALIP-T-E

For the stage optimization (ALIP-S), usually the cycle time is minimized. That is equivalent to maximizing the production output during a restricted implementation phase (ALIP-S-2). Minimizing workstations as ALBP-1 would only make sense when the final answer has less stations than the initial configuration. In this case, the objective is equivalent to removing the extra stations as soon as possible (ALIP-S-1). ALIP-S-F and ALIP-S-E can also be defined, but the applicability might not be direct or of less importance.

Finally, cost objectives (ALIP-C) can measure both effects. ALIP-C can be useful for an analysis of the cost-benefit of the resources in the implementation phase in regard of the production levels. When it is impossible for the interventions not to reduce the overall production conditions, a cost model can be useful to find the best commitment between the implementation stoppages and the production. The ALIP-C class can present itself in several forms, depending on the combinations of which parameters are set as costs. Therefore, the ALIP-C is here not further classified in subclasses.

3.5 Final State

The Implementation Scheduling Problem presented in Section 3.1 is described as finding the best path of implementation from an initial to a final system configuration.

The initial condition is supposed to be the system's actual configuration, while the final condition is obtained previously, possibly with an optimization method. The implementation difficulty is dependent on the configuration's start and goal. The problem defined in terms of fixed initial and final configurations can be called **blinded** to other final configurations.

The choice of the final state can greatly influence the implementation scheduling difficulty. There can be two equally good solutions in terms of system's output for the final configuration with very different implementing difficulties. It can be advantageous to choose the final configuration considering the implementation effort. For instance, one could optimize the goal configuration also taking into account the proximity to the initial system. If there are equally good answers, a solution closer to the initial system may be easier to implement. This way, the optimization of the final state can be **implementation aware**.

Another way to choose the final configuration is **solving the final state and the implementation problems simultaneously**. The final condition can be set as a model variable, transforming the Implementation Scheduling in a multi-objective problem. The combined problems may be harder to solve, but an **integrated approach** would be able to reach easier implementations. The objective is then finding the easiest way to reach a final configuration that respects the conditions. The **simultaneous problem**, if solved, produce the best results in terms of implementation effort compared to the blinded or implementation aware variations. The comparison of the blinded and the simultaneous approach are further discussed in the result section (Section 6.4.3).

The next two chapters relates to the modeling of the Implementation Problems here proposed. Chapter 4 proposes a modeling guide for the states and transitions for generic Implementation Problems. The formulations for the Assembly Line Balancing Problem (ALIP) are presented and described in Chapter 5.

Chapter 4

Modeling Guide

This chapter proposes a modeling guide for Implementation Problems. Expressions are given for modeling changes between two stages for every possible combination of variables. Furthermore, some directions are given in order to model stages and transition periods.

4.1 Modeling Stages

A multi-phase implementation consists of several operational stages. Between two different configurations, one or more changes are made. The modeling presented here has as hypothesis that the modeling of stages can be made discretely. That is, a set of equations can be used to define the system's configuration for each of a set of periods.

Throughout the implementation, a system can assume several configurations. There are, however, some restrictions that must be obeyed to assure the system's functionality. Therefore, the modeling of stages must describe the functionality of the system and assure that the configuration is feasible for any given period. For this chapter, a variable used to model stages is generically named as x, where x can be binary ($x \in \{0, 1\}$), integer ($x \in \mathbb{Z}$) or continuous ($x \in \mathbb{R}$), depending on the application. Let's also consider an index t for the period ($t \in \mathbb{Z}+$), resulting in x_t. The variable x represent a state within an intermediary stage.

The modeling of intermediary stages is very close to the modeling of the final stage

applied in optimization methods. The regular modeling of the Assembly Line Balancing Problem (ALBP), for instance, presents the following restrictions: (i) all tasks must be assigned; (ii) precedence relations must be obeyed; and (iii) the cycle time must be within a given bound. These are exactly the same restrictions an assembly line has to obey during the intermediary production phases between implementation phases. Therefore, much of the modeling equations for stages can be based on the regular formulations adding the index t for each of the considered periods.

An important remark can be given to the tightness of the feasibility conditions. As an example, Figure 4.1 represents a minimization problem. For the four different conditions (a-d), the left rectangle represents the initial stage and the right rectangle the final stage. The dotted line represents the tightness of the stage restrictions for the intermediary stages. The tightness can be set as the same as the initial condition (a), looser than the initial condition (b) or can vary according to the implementation phases (c). The intermediary phases, however, cannot be restricted as tightly as the final answer (d), or no intermediary stages would be feasible. The model would respond that the only feasible implementation is to reach the final stage in one phase. This is relevant when the modeling presents constraints that work as domain cuts. Although cuts are helpful for finding optimal solutions, the reduction on the search space might prone intermediary configurations that would be feasible and potentially optimal, otherwise. Therefore, cuts and tight restrictions should be used with attention in order to avoid unfeasible intermediary configurations.

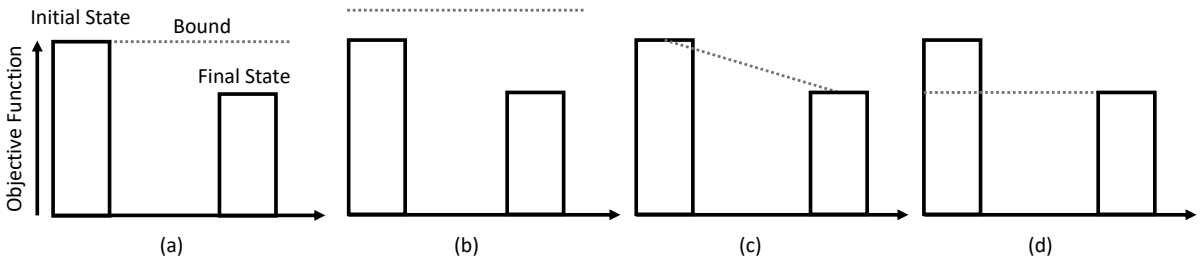

Figure 4.1: Example of restriction tightness for the intermediary states in a minimization problem for 4 different conditions (a-d). For each subfigure, the left bar represents the initial state, the right bar represents the final state, and the doted line represents a bound for the intermediary conditions.

4.2 Modeling Changes

4.2.1 Definition of Variables

Once the stages are modeled, here with generic x variables, we need to identify the changes between stages. As a modeling hypothesis, it is assumed a discrete formulation in terms of the stages. A change of configuration from a stage $t-1$ to a stage t must result in $x_{t-1} \neq x_t$. The identification of changes could be made by the value difference between x_{t-1} and x_t, as represented in the Equation 4.1. For the change variables, the generic symbol y is utilized.

$$y_t = x_t - x_{(t-1)} \qquad \forall\ t\ \in \text{Periods} \tag{4.1}$$

In Assembly Lines Balancing Problems, for the sake of the example, the x variables are used to define in which station each task is assigned. A variable x with the value 1 indicates that the task is performed in the given station, while $x = 0$ states that the task is not assigned. Therefore, using the y variables as the difference between x_{t-1} and x_t results in: $y = 1$ as equivalent to adding a task; $y = -1$ as equivalent to removing a task; and $y = 0$ as maintaining the same state.

Although Equation 4.1 is able to correctly model the existence of a change between states, the measurement of the number of changes or implementation effort is more difficult to obtain. Equation 4.2 illustrates the results of summing the y variables for a given period. Some of the variables assume positive values while others are negative. The summation of such values do not necessarily represent the number of changes in the line. Therefore, the definition of the y variable as is stated in Equation 4.1 produces difficulties for the determination of the total change cost.

$$\sum_t y_t = 1 + 0 - 1 + 0 + 1 + ... \neq \sum changes \tag{4.2}$$

Another modeling possibility exists with $y_t \geq 0$. The y variables present a slightly different nature and some variations can be defined: y_t^+ is used for the positive difference between x_{t-1} and x_t, y_t^- for the negative values, and y_t^m or simply y_t for the module of the difference. In the ALBP example, we then have: $y_t^+ = 1$ as equivalent to adding a task; $y_t^+ = 0$ as equivalent to not adding a task (the task can be either removed or maintained

in the same station); $y_t^- = 1$ represents the removal of a task; $y_t^- = 0$ the non removal of a task; $y_t = 1$ represents a change, that can be either a removed or added task; $y_t = 0$ states that the task's assignment is not changed.

In the example given in the last paragraph, the employed variables are binary. However, x and y can be either binary, integer or continuous. Table 4.1 contains all the combinations of x and y variables. The difference (a change) cannot have richer information than the state itself. Therefore, it does not make sense to measure a change between to binary variables with an integer variable. The transition variable can only measure whether the state variable x changed its value or not. The same occurs with integer and continuous variables: a change on integer variables can only be measured with integer (how much is changed) or binary (if is changed) variables.

Table 4.1: Modeling conditions for transition variables.

State Variable	Transition Variable	Sample Problem
Binary	Binary	SALBP
Integer$^+$	Integer$^+$	Integer SALBP
Integer$^+$	Binary	Integer SALBP
Continuous$^+$	Continuous$^+$	Farming
Continuous$^+$	Integer$^+$	Farming
Continuous$^+$	Binary	Farming

Table 4.1 also contains examples of problems that could use any of the variable combinations. The binary-binary relations can be exemplified by Simple Assembly Line Balancing Problem (SALBP): binary state variables define where each task is assigned; the binary transition y variables measures whether any assignment is, for instance, added to a new station. Integer-Integer relations can be exemplified by the Integer SALBP (Sikora, Lopes, Schibelbain and Magatão, 2017). Instead of having one binary variable to each task-station assignment, the Integer SALBP uses integer variables to measure how many tasks are assigned to each station. This way, a transition variable y could measure how many tasks are added or removed from a station. The Binary-Integer can also be related to Integer SALBP. The x variable shows the number of tasks assigned to a station and the binary y variable could be used to measured whether a change occurred (independent of the amount of reassigned tasks).

For the continuous variables a farming example is given. The continuous x variable could represent the amount of land designated to each culture. Transition continuous vari-

ables could represent an amount of crop that should be harvested or planted. An integer transition variable can be used to define the amount of necessary temporary workers to deal with the harvest of a given area. Finally, binary transition variables could represent the necessity of renting equipment such as tractors or harvesters only for the periods in which they must be used (the crop area changes).

Now that all possible variable types are introduced, we still have to choose how strict the constrains must be. There are different expressions for y_t^+ (y_t^- can be obtained inverting x_t and x_{t-1}) and y_t. Furthermore, we can use an equation on its implication form: if a changed occurred then y must assume a desired value (1 for a binary variable, for instance), otherwise the variable is free; or an equivalence can be used: y assumes 1 if and only if (iff) a change occurred. The difference stays on when no change occurred. In the implication form, y is free, while in the equivalence form, y must be zero. In some formulations, the implication form may be sufficient, mainly when we try to minimize changes. Models in which the transitions are only part of the problem may require the equivalence form.

4.2.2 Formulating Logical Relations

The Mixed-Integer Linear Programming (MILP) models are based on inequalities. Inequality 4.3 serves as example of one Linear Programming expression. Other relations ($<$, $\geq$, $>$, $=$, $\neq$) can also be adapted into regular forms and be written as inequalities.

$$\sum_{j \in J} a_j x_j - b \leq 0 \tag{4.3}$$

If we suppose the variable x_j can assume values between a lower bound l_j and an upper bound u_j, the value of the left side of Expression 4.3 also must be within limited

bounds (L and U), as showed in the following expressions.

$$L \leq \sum_{j \in J} a_j x_j - b \leq U$$
$$L = \sum_{j \in P} a_j l_j + \sum_{j \in N} a_j u_j - b$$
$$U = \sum_{j \in P} a_j u_j + \sum_{j \in N} a_j l_j - b$$
$$P = \{j : a_j > 0\}, N = \{j : a_j < 0\}$$

Magatão (2005) presented a methodology to translate logical statements into linear expressions used in MILP formulations. The development of MILP models use both inequalities and auxiliary binary variables. A binary indicator variable δ can be used to represent the truth or falsehood of a logical preposition p. An extract of the expressions presented by Magatão (2005) is replicated at Table 4.2. For further logic procedures (distributive rule, De Morgan's law, etc.) used along this chapter and in Appendix A, the reader is referred to the book from Gersting (2006).

Table 4.2: Translation of logical statements to MILP expressions. Source: Magatão (2005).

Statement	MILP Expression
p	$\delta = 1$
$\neg p$	$\delta = 0$
$p_1 \vee p_2$	$\delta_1 + \delta_2 \geq 1$
$p_1 \wedge p_2$	$\delta_1 \geq 1; \delta_2 \geq 1$
$p_1 \otimes p_2$	$\delta_1 + \delta_2 = 1$
$p_1 \rightarrow p_2$	$\delta_1 \leq \delta_2$
$p_1 \leftrightarrow p_2$	$\delta_1 = \delta_2$
$p_1 \rightarrow p_2 \wedge p_3$	$\delta_1 \leq \delta_2, \delta_1 \leq \delta_3$
$p_1 \rightarrow p_2 \vee p_3$	$\delta_1 \leq \delta_2 + \delta_3$
$p_1 \wedge p_2 \rightarrow p_3$	$\delta_1 + \delta_2 - \delta_3 \leq 1$
$p_1 \vee p_2 \rightarrow p_3$	$\delta_1 \leq \delta_3, \delta_2 \leq \delta_3$

One interesting feature of such use of logical statements is the possibility of integrating logical and linear expressions. A binary variable δ can be used to activate a MILP restriction, for instance. Magatão (2005) described all the possible combinations of equations to express *either-or*, *if-then*, and *if-the-else* relations from a binary value to linear expressions. The relation from a logical proposition (represented by a binary variable δ) to a linear expression can be given either by an implication or an equivalence. An example

of both implication and equivalence relations extracted from Magatão (2005) are given in Propositions 4.2.1 and 4.2.2. For both examples, the variable δ controls an expression in the form *less or equal than*. All the other combinations ($<$, $\geq$, $>$, $=$, $\neq$) are described in Magatão (2005). Some expressions need a small positive value ε to adapt equations into inequalities.

Proposition 4.2.1.

$$\underbrace{\left(\delta = 1 \rightarrow \sum_j a_j x_j - b \leq 0\right)}_{\textit{if antecedent then consequent}} \quad \equiv \quad \underbrace{\sum_j a_j x_j - b \leq U(1-\delta)}_{\textit{MILP expression}}$$

Proof. In proposition 4.2.1, δ is a binary variable, that is, it can assume either values 0 or 1. Testing both cases in the MILP expression results in:

$$\delta = 0 \rightarrow \sum_j a_j x_j - b \leq U \therefore \text{tautology}$$

$$\delta = 1 \rightarrow \sum_j a_j x_j - b \leq 0 \therefore \text{proposition holds}$$

Therefore, setting $\delta = 0$ imposes the relation $\sum_j a_j x_j - b \leq U$, which is valid for any valid value the variable x_j can assume if U is large enough. Setting $\delta = 1$ makes the constraint $\sum_j a_j x_j - b \leq 0$ valid. Thus, proposition 4.2.1 holds. □

Proposition 4.2.2.

$$\underbrace{\left(\delta = 1 \leftrightarrow \sum_j a_j x_j - b \leq 0\right)}_{\textit{if and only if antecedent then consequent}} \quad \equiv \quad \underbrace{\begin{cases} \sum_j a_j x_j - b \leq U(1-\delta) \\ \sum_j a_j x_j - b \geq (L-\varepsilon)\delta + \varepsilon \end{cases}}_{\textit{MILP expression}}$$

Proof. An equivalence can be generically expressed as:

$$p \leftrightarrow q \equiv \underbrace{(p \rightarrow q)}_{(i_1)} \wedge \underbrace{(\neg p \rightarrow \neg q)}_{(i_2)}$$

Hence, if implications (i_1) and (i_2) hold, then the equivalence holds. In the particular equivalence presented in Proposition 4.2.2, implications (i_1) and (i_2) can be written as:

$$\text{Implication}(i_1) : \delta = 1 \rightarrow \sum_j a_j x_j - b \leq 0$$

$$\text{Implication}(i_2) : \delta = 0 \rightarrow \sum_j a_j x_j - b \nleq 0 \therefore \sum_j a_j x_j - b > 0$$

In proposition 4.2.2, δ can assume either values 0 or 1. Testing both cases in the set of MILP expression yields:

$$\delta = 0 \rightarrow \left\{ \begin{array}{l} \sum_j a_j x_j - b \leq U \therefore \text{tautology} \\ \sum_j a_j x_j - b \geq \varepsilon \therefore \sum_j a_j x_j - b > 0 \end{array} \right\} \therefore \text{implication } (i_2) \text{ holds}$$

$$\delta = 1 \rightarrow \left\{ \begin{array}{l} \sum_j a_j x_j - b \leq 0 \\ \sum_j a_j x_j - b \geq L \therefore \text{tautology} \end{array} \right\} \therefore \text{implication } (i_1) \text{ holds}$$

Implications (i_1) and (i_2) hold, therefore proposition 4.2.2 holds. □

Remark 4.2.1. $(L \leq 0) \wedge (U \geq \varepsilon)$

Magatão (2005) also described how to model logical relations between two linear expressions (such as $Expression_1 \rightarrow Expression_2$). The methodology consists in linking each expression with an equivalence relation to a binary variable ($\delta_1 \leftrightarrow Expression_1$ and $\delta_2 \leftrightarrow Expression_2$). Afterwards, the relation between the expressions is modeled based on the δ variables: $\delta_1 \rightarrow \delta_2$. The result consists in a set of equations that may or not be able to be simplified. Following the presented procedure, Expression 4.4 is translated to MILP expressions as an example.

$$\sum_j a_j x_j - b \leq 0 \rightarrow \sum_j c_j y_j - d \leq 0 \tag{4.4}$$

Firstly, the variables δ_1 and δ_2 are defined as auxiliary variables. The implication relation can now be expressed as:

$$\left\{ \begin{array}{l} \delta_1 \leftrightarrow \left(\sum_j a_j x_j - b \leq 0 \right) \\ \delta_2 \leftrightarrow \left(\sum_j c_j x_j - d \leq 0 \right) \\ \delta_1 \rightarrow \delta_2 \end{array} \right. \tag{4.5}$$

By applying the rules of Table 4.2 to the implication between δ_1 and δ_2 and Proposition 4.2.2 to the equivalent relations, Equation Set 4.6 can be obtained.

$$\left\{ \begin{array}{l} \sum_j a_j x_j - b \leq U_1(1 - \delta_1) \\ \sum_j a_j x_j - b \geq (L_1 - \varepsilon)\delta_1 + \varepsilon \\ \sum_j c_j y_j - d \leq U_2(1 - \delta_2) \\ \sum_j c_j y_j - d \geq (L_2 - \varepsilon)\delta_2 + \varepsilon \\ \delta_1 \leq \delta_2 \end{array} \right. \tag{4.6}$$

4.2.3 Modeling Structures

In this section, Magatão (2005)'s methodology and propositions are used to define the expressions for the relations between state and transition variables for each of the combinations given by Table 4.1.

As described in Section 4.2.1, there are different expressions for the variables y_t^+ (representing a positive change) and y_t (representing the module of the change amplitude). Furthermore, the relations between stages and transitions can be either linked by implication or equivalence statements. For the explanation of the rest of the section, the term "assignment" is used as the stage variable meaning. Assignments, however, are just an example of how the variable can be used.

Binary to Binary Relations

Starting by the binary-binary relations, y_t^+ is defined as 1 when a new assignment is made. In other words, $y_t^+ = 1$ implies that the state variable from a previously period is false ($x_{(t-1)} = 0$) and becomes true at the concerning transition ($x_t = 1$). The modeling of y_t^- follows the same principle and is not described. It is sufficient to invert the positions of x_t and $x_{(t-1)}$. The transition variable for the module of the changes, y_t, is defined as 1 either in the case of a new assignment ($x_{(t-1)} = 0, x_t = 1$) or when an assignment is removed ($x_{(t-1)} = 1, x_t = 0$). The logical statements in the implication form for the given variable's definition is given herein:

$$\underbrace{x_t = 1 \wedge x_{(t-1)} = 0}_{\text{new assignment}} \rightarrow y_t^+ \qquad \equiv x_t \wedge \neg x_{(t-1)} \rightarrow y_t^+ \tag{4.7}$$

$$\underbrace{x_t = 0 \wedge x_{(t-1)} = 1}_{\text{assignment removal}} \rightarrow y_t^- \qquad \equiv \neg x_t \wedge x_{(t-1)} \rightarrow y_t^- \tag{4.8}$$

$$\underbrace{(x_t = 1 \wedge x_{(t-1)} = 0) \vee (x_t = 0 \wedge x_{(t-1)} = 1)}_{\text{assignment change}} \rightarrow y_t \qquad \equiv x_t \otimes x_{(t-1)} \rightarrow y_t \tag{4.9}$$

The Expressions 4.7 to 4.9 can be translated into MILP expressions by using Table 4.2. Once a set of prepositions $p_1 \wedge p_2 \rightarrow p_3$ can be written as $\delta_1 + \delta_2 - \delta_3 \leq 1$ and $p_1 \vee p_2 \rightarrow p_3$ is equivalent to $(\delta_1 \leq \delta_3, \delta_2 \leq \delta_3)$, we have the propositions:

Proposition 4.2.3.

$$\underbrace{\left(x_t \wedge \neg x_{(t-1)} \rightarrow y_t^+\right)}_{\textit{if antecedent then consequent}} \quad \equiv \quad x_t + (1 - x_{(t-1)}) - y_t^+ \leq 1 \quad \equiv \quad \underbrace{y_t^+ \geq x_t - x_{(t-1)}}_{\textit{MILP expression}}$$

Proof. In proposition 4.2.3, x_t and $x_{(t-1)}$ are binary variables, that is, they can assume either values 0 or 1. The truth table of the statement is expressed bellow, along with the resulting MILP expressions. The column representing the value of y_t^+ represents the value y_t^+ has to assume so that $x_t \wedge \neg x_{(t-1)} \rightarrow y_t^+$ is true.

x_t	$x_{(t-1)}$	$x_t \wedge \neg x_{(t-1)}$	y_t^+	MILP Expression	Conclusion
0	0	0	free	$y_t^+ \geq 0$	$\therefore$ tautology
0	1	0	free	$y_t^+ \geq -1$	$\therefore$ tautology
1	0	1	1	$y_t^+ \geq 1$	$\therefore y_t^+ = 1$, proposition holds
1	1	0	free	$y_t^+ \geq 0$	$\therefore$ tautology

Therefore, setting x_t and $x_{(t-1)}$ the values {(0;0), (0;1), and (1;1)} imposes the relation $y_t^+ \geq 0$ or $y_t^+ \geq -1$, which is valid for any value the variable y_t^+ can assume. Setting $x_t = 1$ and $x_{(t-1)} = 0$ makes $y_t^+ \geq 1$, resulting in $y_t^+ = 1$. Thus, proposition 4.2.3 holds. □

Proposition 4.2.4.

$$\underbrace{(x_t \wedge \neg x_{(t-1)}) \vee (\neg x_t \wedge x_{(t-1)}) \rightarrow y_t}_{\textit{if antecedent then consequent}} \equiv \underbrace{(x_t \wedge \neg x_{(t-1)}) \rightarrow y_t \wedge (\neg x_t \wedge x_{(t-1)}) \rightarrow y_t}_{\textit{distributive rule for the implication}} \equiv$$

$$\equiv \begin{cases} x_t + (1 - x_{(t-1)}) - y_t \leq 1 \\ (1 - x_t) + x_{(t-1)} - y_t \leq 1 \end{cases} \quad \equiv \quad \underbrace{\begin{cases} y_t \geq x_t - x_{(t-1)} \\ y_t \geq x_{(t-1)} - x_t \end{cases}}_{\textit{MILP expression}}$$

Proof. In proposition 4.2.4, x_t and $x_{(t-1)}$ are binary variables, that is, they can assume either values 0 or 1. The truth table of the statements is expressed bellow, along with the resulting MILP expressions. The column representing the value of y_t represents the value y_t has to assume so that $x_t \otimes x_{(t-1)} \rightarrow y_t$ is true.

x_t	$x_{(t-1)}$	$x_t \otimes x_{(t-1)}$	y_t	MILP Expr.	Conclusion
0	0	0	free	$\begin{cases} y_t \geq 0 \\ y_t \geq 0 \end{cases}$	$\therefore$ tautology
0	1	1	1	$\begin{cases} y_t \geq -1 \\ y_t \geq 1 \end{cases}$	$\therefore y_t = 1$, proposition holds
1	0	1	1	$\begin{cases} y_t \geq 1 \\ y_t \geq -1 \end{cases}$	$\therefore y_t = 1$, proposition holds
1	1	0	free	$\begin{cases} y_t \geq 0 \\ y_t \geq 0 \end{cases}$	$\therefore$ tautology

Therefore, setting x_t and $x_{(t-1)}$ the values $\{(0;0)$ and $(1;1)\}$ imposes the relation $y_t \geq 0$, which is valid for any valid value the variable y_t can assume. Setting x_t and $x_{(t-1)}$ the values $\{(0;1)$ and $(1;0)\}$ makes $y_t \geq 1$ and $y_t \geq -1$, resulting in $y_t = 1$. Thus, proposition 4.2.4 holds. □

Table 4.3 contains the four possible expressions for the binary-binary relations. The expressions using equivalence relations (the third and fourth) are developed and proved in the Appendix A.

Table 4.3: Logical Constraints for binary state variables (x_t) and binary transition variables (y_t).

Logical Constraint ($\forall t \in T$)	Equivalent Set of MILP Expressions ($\forall t \in T$)
$x_t \wedge \neg x_{(t-1)} \rightarrow y_t^+$	$y_t^+ \geq x_t - x_{(t-1)}$
$x_t \otimes x_{(t-1)} \rightarrow y_t$	$\begin{cases} y_t \geq x_t - x_{(t-1)} \\ y_t \geq x_{(t-1)} - x_t \end{cases}$
$x_t \wedge \neg x_{(t-1)} \leftrightarrow y_t^+$	$\begin{cases} y_t^+ \geq x_t - x_{(t-1)} \\ y_t^+ \leq x_t \\ y_t^+ \leq 1 - x_{(t-1)} \end{cases}$
$x_t \otimes x_{(t-1)} \leftrightarrow y_t$	$\begin{cases} y_t \geq x_t - x_{(t-1)} \\ y_t \geq x_{(t-1)} - x_t \\ y_t \leq 2 - x_t - x_{(t-1)} \\ y_t \leq x_t + x_{(t-1)} \end{cases}$

Note that the expressions for y_t^+ are simpler than the ones from y_t. Furthermore,

the implication relations also present less equations than the equivalence relations. This is due to the fact that an equivalence can be written as two implications ($p \leftrightarrow q \equiv (p \rightarrow q) \wedge (\neg p \rightarrow \neg q)$).

Integer or Continuous to Binary Relations

The next expressions consist in the link between integer or continuous state variables and binary transition variables. The transition variables measure whether the state variables changed their value. Both integer and continuous state variables result in the same expressions.

In this chapter, only the first expression of Table 4.4 is developed and proved, the other proofs can be found in Appendix A. The logic statement for the implication form for the variable y_t^+ is given by Expression 4.10.

$$\underbrace{\big(x_t - x_{(t-1)} > 0\big)}_{\text{positive change in state variables}} \rightarrow y_t^+ \tag{4.10}$$

We now have a linear expression as part of a logical statement. According to the procedure described in Section 4.2.2, it is necessary to define an auxiliary binary variable δ so that $\delta \leftrightarrow \big(x_t - x_{(t-1)} > 0\big)$. The translation of the auxiliary statement to MILP expressions is given in Formulation 4.11.

$$\begin{aligned} \delta \leftrightarrow \big(x_t - x_{(t-1)} > 0\big) &\equiv \underbrace{\big(\delta \rightarrow (x_t - x_{(t-1)} > 0)\big) \wedge \big(\neg\delta \rightarrow (x_t - x_{(t-1)} \leq 0)\big)}_{\text{equivalence}} \equiv \\ &\equiv \underbrace{\begin{cases} x_t - x_{(t-1)} \geq (L-\varepsilon)(1-\delta) + \varepsilon \\ x_t - x_{(t-1)} \leq U(1-(1-\delta)) = U\delta \end{cases}}_{\text{implication form expressions (Magatão, 2005)}} \equiv \underbrace{\begin{cases} \delta \leq \frac{x_t - x_{(t-1)} - L}{-L+\varepsilon} \\ \delta \geq \frac{x_t - x_{(t-1)}}{U} \end{cases}}_{\text{isolating } \delta} \end{aligned} \tag{4.11}$$

With the variable δ defined, the implication relation $\delta \rightarrow y_t^+$ (Equation 4.10) can be determined. According to Table 4.2, $\delta \rightarrow y_t^+ \equiv y_t^+ \geq \delta$. From the developed expressions 4.11, $\delta \geq \frac{x_t - x_{(t-1)}}{U}$, so that $y_t^+ \geq \frac{x_t - x_{(t-1)}}{U}$. Once the link from δ and y_t^+ is only in the implication form, the expression $\delta \leq \frac{x_t - x_{(t-1)} - L}{-L+\varepsilon}$ is not relevant for the variable y_t^+. Therefore, the result MILP expression can be rewritten as $Uy_t^+ \geq x_t - x_{(t-1)}$.

Proposition 4.2.5.

$$\underbrace{\overbrace{\big(x_t - x_{(t-1)} > 0\big)}^{\textit{positive change in state variables}} \;\rightarrow\; y_t^+}_{\textit{if antecedent then consequent}} \quad \equiv \quad \underbrace{Uy_t^+ \geq x_t - x_{(t-1)}}_{\textit{MILP expression}}$$

Proof. In proposition 4.2.5, $(x_t - x_{(t-1)} > 0)$ can either be true of false. Testing both cases in the MILP expression results in:

$$x_t - x_{(t-1)} > 0 \;\rightarrow\; Uy_t^+ \geq x_t - x_{(t-1)} > 0 \;\therefore\; y_t^+ = 1 \qquad \therefore \text{ proposition holds}$$
$$x_t - x_{(t-1)} \leq 0 \;\rightarrow\; Uy_t^+ \geq x_t - x_{(t-1)} \leq 0 \qquad \therefore \text{ tautology}$$

Therefore, assuming $x_t \leq x_{(t-1)}$ imposes the relation $Uy_t^+ \geq x_t - x_{(t-1)}$, whose right side is less or equal than zero. Therefore the expression is valid for any value the variable y_t^+ can assume. Assuming $x_t > x_{(t-1)}$ imposes $y_t^+ = 1$ in the constraint $Uy_t^+ \geq x_t - x_{(t-1)}$. Thus, proposition 4.2.5 holds. □

Table 4.4 contains the expressions for both y_t^+ and y_t for the implication and equivalence relations for the link between integer or continuous state variables to binary transient variables.

Table 4.4: Logical Constraints for integer or continuous state variables (x_t) and binary transition variables (y_t).

Logical Constraint ($\forall t \in T$)	Equivalent Set of MILP Expressions ($\forall t \in T$)	Remark
$\big(x_t - x_{(t-1)} > 0\big) \rightarrow y_t^+$	$Uy_t^+ \geq x_t - x_{(t-1)}$	$L \leq 0$
$\big(\lvert x_t - x_{(t-1)}\rvert > 0\big) \rightarrow y_t$	$\begin{cases} Uy_t \geq x_t - x_{(t-1)} \\ Uy_t \geq x_{(t-1)} - x_t \end{cases}$	$L \leq 0$
$\big(x_t - x_{(t-1)} > 0\big) \leftrightarrow y_t^+$	$\begin{cases} Uy_t^+ \geq x_t - x_{(t-1)} \\ (L-\varepsilon)(1-y_t^+) + \varepsilon \leq x_t - x_{(t-1)} \end{cases}$	$\begin{cases} L \leq -\varepsilon \\ U \geq 0 \end{cases}$
$\big(\lvert x_t - x_{(t-1)}\rvert > 0\big) \leftrightarrow y_t$	$\begin{cases} U\delta_t^+ \geq x_t - x_{(t-1)} \\ (L-\varepsilon)(1-\delta_t^+) + \varepsilon \leq x_t - x_{(t-1)} \\ U\delta_t^- \geq x_{(t-1)} - x_t \\ (L-\varepsilon)(1-\delta_t^-) + \varepsilon \leq x_{(t-1)} - x_t \\ y_t = \delta_t^+ + \delta_t^- \end{cases}$	$\begin{cases} L \leq -\varepsilon \\ U \geq 0 \end{cases}$

Integer to Integer or Continuous to Continuous Relations

We now introduce the third type of combination of variables. The expressions derived from integer-integer and continuous-continuous relations are identical. The modeling of integer or continuous transient variables has more variants than its binary counterpart. Besides the different uses of the y_t^+ and y_t the definition of the consequent also assumes multiple forms. For the binary transition variable, the logical constraint that models the implication form for y_t^+ is $x_t - x_{(t-1)} > 0 \rightarrow y_t^+$. When y_t^+ becomes an integer variable, there are two forms to express this relationship: with an inequality ($x_t - x_{(t-1)} > 0 \rightarrow y_t^+ \geq x_t - x_{(t-1)}$) and with an equality ($x_t - x_{(t-1)} > 0 \rightarrow y_t^+ = x_t - x_{(t-1)}$).

The difference of the consequent formed with an inequality and with an equality also resides in the tightness of the model. If assuring that y_t^+ is at least as great as the difference between the x variables is enough, the inequality is sufficient. On the other hand, if one has to assure y_t^+ cannot assume values greater than $x_t - x_{(t-1)}$, the equality and, therefore, more complex expressions, must be used.

For the integer-integer relations, the consequent is no longer a binary variable, it consists now of linear expressions. The predicate and the consequent are linked either with an implication or with an if-then-else statement instead of the equivalence.

The development of the expressions given in Table 4.5 do not follow strictly the procedure of the binary variations. Although the same methodology could be applied, it would result in several more inequalities than the necessary. For the formulation of Table 4.5's expressions, several simplifications are applied.

The first expression, for instance, $x_t - x_{(t-1)} > 0 \rightarrow y_t^+ \geq x_t - x_{(t-1)}$ can be translated to the single linear expression $y_t^+ \geq x_t - x_{(t-1)}$. Once the predicate and the consequent have similar components ($x_t - x_{(t-1)}$), no auxiliary binary variable is needed. If $x_t - x_{(t-1)} < 0$, the linear expression $y_t^+ \geq x_t - x_{(t-1)}$ is equivalent to a tautology.

In this section, the logical constraints for each variation of Table 4.5 is explained, while the proof of the equivalent MILP expressions are presented in Appendix A.

The first four expressions relate to the non-negative y_t^+ variable. The first expression $x_t - x_{(t-1)} > 0 \rightarrow y_t^+ \geq x_t - x_{(t-1)}$ states that if there is an increase in the value of two state variables, the transition variable y_t^+ must be at least as great as the difference. The

Table 4.5: Logical Constraints for integer state variables (x_t) and integer transition variables (y_t) or continuous stage variables (x_t) and continuous transition variables (y_t).

Logical Constraint ($\forall t \in T$)	Equivalent Set of MILP Expressions ($\forall t \in T$)	Remark
$x_t - x_{(t-1)} > 0 \rightarrow y_t^+ \geq x_t - x_{(t-1)}$	$y_t^+ \geq x_t - x_{(t-1)}$	
$x_t - x_{(t-1)} > 0 \rightarrow y_t^+ = x_t - x_{(t-1)}$	$\begin{cases} U\delta \geq x_t - x_{(t-1)} \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq x_t - x_{(t-1)} + U(1-\delta) \end{cases}$	$\begin{cases} L \leq 0 \\ U \geq 0 \end{cases}$
If $\left(x_t - x_{(t-1)} > 0\right)$ then $\left(y_t^+ \geq x_t - x_{(t-1)}\right)$ else $\left(y_t^+ = 0\right)$	$\begin{cases} U\delta \geq x_t - x_{(t-1)} \\ (L-\varepsilon)(1-\delta) + \varepsilon \leq x_t - x_{(t-1)} \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq U\delta \end{cases}$	$\begin{cases} L \leq 0 \\ U \geq +\varepsilon \end{cases}$
If $\left(x_t - x_{(t-1)} > 0\right)$ then $\left(y_t^+ = x_t - x_{(t-1)}\right)$ else $\left(y_t^+ = 0\right) \equiv$ $\equiv$ If $\left(x_t - x_{(t-1)} \geq 0\right)$ then $\left(y_t^+ = x_t - x_{(t-1)}\right)$ else $\left(y_t^+ = 0\right)$	$\begin{cases} U\delta \geq x_t - x_{(t-1)} \\ L(1-\delta) + \varepsilon \leq x_t - x_{(t-1)} \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq x_t - x_{(t-1)} + U(1-\delta) \\ y_t^+ \leq U\delta \end{cases}$	$\begin{cases} L \leq 0 \\ U \geq +\varepsilon \end{cases}$
$\lvert x_t - x_{(t-1)} \rvert > 0 \rightarrow y_t \geq \lvert x_t - x_{(t-1)} \rvert$	$\begin{cases} y_t \geq x_t - x_{(t-1)} \\ y_t \geq x_{(t-1)} - x_t \end{cases}$	
$\lvert x_t - x_{(t-1)} \rvert > 0 \rightarrow y_t = \lvert x_t - x_{(t-1)} \rvert$	$\begin{cases} U\delta^+ \geq x_t - x_{(t-1)} \\ U\delta^- \geq x_{(t-1)} - x_t \\ y_t \geq x_t - x_{(t-1)} + L(1-\delta^+) \\ y_t \leq x_t - x_{(t-1)} + U(1-\delta^+) \\ y_t \geq x_{(t-1)} - x_t + L(1-\delta^-) \\ y_t \leq x_{(t-1)} - x_t + U(1-\delta^-) \end{cases}$	$\begin{cases} L \leq 0 \\ U \geq 0 \end{cases}$
If $\left(\lvert x_t - x_{(t-1)} \rvert > 0\right)$ then $\left(y_t \geq \lvert x_t - x_{(t-1)} \rvert\right)$ else $\left(y_t = 0\right)$	$\begin{cases} (L-\varepsilon)(1-\delta^+) + \varepsilon \leq x_t - x_{(t-1)} \\ (L-\varepsilon)(1-\delta^-) + \varepsilon \leq x_{(t-1)} - x_t \\ y_t \leq U\delta^+ + U\delta^- \\ y_t \geq x_t - x_{(t-1)} \\ y_t \geq x_{(t-1)} - x_t \end{cases}$	$\begin{cases} L \leq -\varepsilon \\ U \geq +\varepsilon \end{cases}$
If $\left(\lvert x_t - x_{(t-1)} \rvert > 0\right)$ then $\left(y_t = \lvert x_t - x_{(t-1)} \rvert\right)$ else $\left(y_t = 0\right) \equiv$ $\equiv \left(y_t = \lvert x_t - x_{(t-1)} \rvert\right)$	$\begin{cases} U\delta \geq x_t - x_{(t-1)} \\ L(1-\delta) \leq x_t - x_{(t-1)} \\ y_t \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t \leq x_t - x_{(t-1)} + U(1-\delta) \\ y_t \geq x_{(t-1)} - x_t + L\delta \\ y_t \leq x_{(t-1)} - x_t + U\delta \end{cases}$	$\begin{cases} L \leq 0 \\ U \geq 0 \end{cases}$

second expression ($x_t - x_{(t-1)} > 0 \rightarrow y_t^+ = x_t - x_{(t-1)}$) only differs by demanding that y_t^+ assumes exactly the value of the difference. For the two first expressions, however, when $x_t \leq x_{(t-1)}$, the variable y_t^+ is free to assume any value. The third and fourth expressions also present an upper bound for y_t^+. The statement *If* $(x_t - x_{(t-1)} > 0)$ *then* $(y_t^+ \geq x_t - x_{(t-1)})$ *else* $(y_t^+ = 0)$ assures with the else clause that y_t^+ is null if $x_t \leq x_{(t-1)}$.

The fifth to eighth expressions from Table 4.5 relates to the non negative variable y_t, responsible to measure the module of the difference between x_t and $x_{(t-1)}$. The fifth and sixth expressions contains the implication logical clauses, while the seventh and eighth set of statements are if-then-else clauses.

Continuous to Integer Relations

Finally, Table 4.6 contains the expressions to link continuous state variables to integer transition variables. These expressions are very similar to the integer-integer relations presented in Table 4.5. The differences are that the logical constraint contains a ceiling function. For instance, if the difference between x_t and $x_{(t-1)}$ is 1.5 units, the integer variable y_t^+ should account for the change of 2 units. The resulting MILP expressions are the same as in Table 4.5 for the logical statements that contain $y_t^+ \geq \lceil x_t - x_{(t-1)} \rceil$. For the expressions with $y_t^+ = \lceil x_t - x_{(t-1)} \rceil$, however, an adjustment on the lower bound restrictions for y_t^+ must be made. This adjustment consists on assuring y_t^+ is limited between the bounds $x_t - x_{(t-1)} \leq y_t^+ \leq x_t - x_{(t-1)} + (1 - \varepsilon)$, with ε as a small positive value.

Other variations can derive from Tables 4.5 and 4.6 here presented. For continuous or integer variables, for instance, the transition variables do not need to respect the 1:1 proportion on the difference of state variables. One could use a constant different than 1 to model $y_t^+ = k(x_t - x_{(t-1)})$ if necessary. In another possibility, the y variables could be limited to a set of constants based on intervals for $(x_t - x_{(t-1)})$. Moreover, a piecewise linear relation can also be defined between the variables.

Table 4.6: Logical Constraints for continuous state variables (x_t) and integer transition variables (y_t) using the ceiling function. Remarks are the same as Table 4.5 and are therefore omitted.

Logical Constraint ($\forall t \in T$)	Equivalent Set of MILP Expressions ($\forall t \in T$)
$x_t - x_{(t-1)} > 0 \rightarrow y_t^+ \geq \lceil x_t - x_{(t-1)} \rceil$	$y_t^+ \geq x_t - x_{(t-1)}$
$x_t - x_{(t-1)} > 0 \rightarrow y_t^+ = \lceil x_t - x_{(t-1)} \rceil$	$\begin{cases} U\delta \geq x_t - x_{(t-1)} \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq x_t - x_{(t-1)} + (1-\varepsilon) + U(1-\delta) \end{cases}$
If $\left(x_t - x_{(t-1)} > 0\right)$ then $\left(y_t^+ \geq \lceil x_t - x_{(t-1)} \rceil\right)$ else $\left(y_t^+ = 0\right)$	$\begin{cases} U\delta \geq x_t - x_{(t-1)} \\ (L-\varepsilon)(1-\delta) + \varepsilon \leq x_t - x_{(t-1)} \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq U\delta \end{cases}$
If $\left(x_t - x_{(t-1)} > 0\right)$ then $\left(y_t^+ = \lceil x_t - x_{(t-1)} \rceil\right)$ else $\left(y_t^+ = 0\right)$	$\begin{cases} U\delta \geq x_t - x_{(t-1)} \\ L(1-\delta) + \varepsilon \leq x_t - x_{(t-1)} \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq x_t - x_{(t-1)} + (1-\varepsilon) + U(1-\delta) \\ y_t^+ \leq U\delta \end{cases}$
$\lvert x_t - x_{(t-1)} \rvert > 0 \rightarrow y_t \geq \lceil \lvert x_t - x_{(t-1)} \rvert \rceil$	$\begin{cases} y_t \geq x_t - x_{(t-1)} \\ y_t \geq x_{(t-1)} - x_t \end{cases}$
$\lvert x_t - x_{(t-1)} \rvert > 0 \rightarrow y_t = \lceil \lvert x_t - x_{(t-1)} \rvert \rceil$	$\begin{cases} U\delta^+ \geq x_t - x_{(t-1)} \\ U\delta^- \geq x_{(t-1)} - x_t \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta^+) \\ y_t^+ \leq x_t - x_{(t-1)} + (1-\varepsilon) + U(1-\delta^+) \\ y_t^+ \geq x_{(t-1)} - x_t + L(1-\delta^-) \\ y_t^+ \leq x_{(t-1)} - x_t + (1-\varepsilon) + U(1-\delta^-) \end{cases}$
If $\left(\lvert x_t - x_{(t-1)} \rvert > 0\right)$ then $\left(y_t \geq \lceil \lvert x_t - x_{(t-1)} \rvert \rceil\right)$ else $\left(y_t = 0\right)$	$\begin{cases} (L-\varepsilon)(1-\delta^+) + \varepsilon \leq x_t - x_{(t-1)} \\ (L-\varepsilon)(1-\delta^-) + \varepsilon \leq x_{(t-1)} - x_t \\ y_t \leq U\delta^+ + U\delta^- \\ y_t \geq x_t - x_{(t-1)} \\ y_t \geq x_{(t-1)} - x_t \end{cases}$
If $\left(\lvert x_t - x_{(t-1)} \rvert > 0\right)$ then $\left(y_t = \lceil \lvert x_t - x_{(t-1)} \rvert \rceil\right)$ else $\left(y_t = 0\right) \equiv$ $\equiv \left(y_t = \lceil \lvert x_t - x_{(t-1)} \rvert \rceil\right)$	$\begin{cases} U\delta \geq x_t - x_{(t-1)} \\ L(1-\delta) \leq x_t - x_{(t-1)} \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq x_t - x_{(t-1)} + (1-\varepsilon) + U(1-\delta) \\ y_t^+ \geq x_{(t-1)} - x_t + L\delta \\ y_t^+ \leq x_{(t-1)} - x_t + (1-\varepsilon) + U\delta \end{cases}$

4.3 Modeling Transitions

The main purpose of modeling transition is the measurement of the implementation effort. Assuring the feasibility or optimizing a multi phase implementation requires the correct determination of the implementation costs.

The first step in the modeling is determining what requires effort in the implementation. In other words, which of the variables x in the model has a cost if its value changes. A cost here can be seen not just as monetary, but also as time, man-force, or resources in general. An implementation cost is, for instance, the change of assignment of a machine in a flow-shop. Changing this assignment may require money to pay specialized work, work hours of an employee, or stoppage time of the production system. Maintaining the machine in the same station, however, has a null implementation cost. The amount of energy used for the machines, for instance, cannot be modeled as an implementation cost. The electric bill has to be paid based on the value of its use (x variable) and not by the change between the usage of two periods (y variable).

Once the implementation cost have been identified, the changes of the variables that result in costs can be modeled by the Tables 4.3 - 4.6. While the x variables defines the system's configuration for each of the periods, the y variables identify changes between two periods.

The measurement of the implementation effort is based on the changes identified by the y variables. The most simple form to calculate the effort is by summing the cost of every individual change. This can be made by summing the product of every y variable to the cost c of each change, as shown in Equation 4.12.

$$Effort_t = \sum_i y_{it} \cdot c_i \quad \forall\ t \in \text{Transitions} \tag{4.12}$$

However, more complex forms of calculating the necessary effort may arise. Synergy between similar changes can be a reasonable assumption in the modeling of transitions. Suppose, for instance, a problem with $y_1, y_2, ..., y_n$ as transition variables. Every y_i (with $i = 1...n$) variable is associated to a cost c_i if the change is performed individually. If pairs of related changes are performed in the same transition, however, their costs are reduced due to similarities.

A set of binary variables z_{ij} can be used to model such synergy effects. Suppose that if task i and j are performed in the same transition, a cost reduction of r_{ij} can be applied. The cost function can be then rewritten as Equation 4.13. Inequalities 4.14 - 4.16 model the link between y and z variables. The expressions represent a *logic and* between y_i, y_j, and z_{ij} for every t.

$$Effort_t = \sum_i y_{it} \cdot c_i - \sum_{i,j} z_{ijt} \cdot r_{ij} \quad \forall\ t \in \text{Transitions} \tag{4.13}$$

$$z_{ijt} \leq y_{it} \qquad \forall\ t \in \text{Transitions}, i, j \in \text{Changes} \tag{4.14}$$

$$z_{ijt} \leq y_{jt} \qquad \forall\ t \in \text{Transitions}, i, j \in \text{Changes} \tag{4.15}$$

$$z_{ijt} \geq y_{it} + y_{jt} - 1 \qquad \forall\ t \in \text{Transitions}, i, j \in \text{Changes} \tag{4.16}$$

If an implementation is performed by more than one person or agent, some coordination issues may arise. Suppose that changes cannot be divided among different agents: the re-installation of a machine might require one worker, while a second worker could not accelerate the process. In other words, some changes can be considered indivisible.

This consideration may link the implementation problem with the bin packing problem. As an example, suppose there are three changes that must be performed requiring five hours of work each. If the changes were divisible, two workers with each 8 hours available could be able to perform the implementation ($5+5+5 = 15 \leq 16 = 8 \cdot 2$). If we consider the changes to be indivisible, only two workers would not be able to perform the implementation. Each worker would perform a task in five hours. The third task could only be performed by a single worker, but both of them have only 3 hours left.

One manner to model the bin packing transition phase is with the use of auxiliary binary variables z. Suppose y variables are defined with a generic index i along with the index t for transitions (y_{it}). The y variables that are equal to 1 in each transition must be allocated between one of the workers, agents, or, more generically, bins ($1...B$). The variable z can then be defined as z_{ibt}. The effort cost per bin can then be calculated based on the z variables, as it is shown in Equation 4.17. The link between y and z variables is performed by Equation 4.18. This equation works as an occurrence constraint. It assures the change i is assigned to a bin (left side of the equation) if the change i is performed at

the given period (y_{it} is true).

$$Effort_{tb} = \sum_{i} z_{ibt} \cdot c_i \quad \forall\ t \in \text{Transitions}, b \in \text{Bins} \tag{4.17}$$

$$\sum_{b} z_{ibt} = y_{it} \quad \forall\ t \in \text{Transitions}, i \in \text{Changes} \tag{4.18}$$

Another complication may arise when there are precedence relations between the changes. Suppose that an implementation require the repositioning and calibration of machines. The calibration, however, must be performed in the new location, or it would be useless after the relocation. There is, therefore, a precedence relation between the repositioning and calibration procedures. If multiple workers are to perform changes linked with precedence relations, a scheduling model should be defined.

A formulation contemplating the scheduling of the changes is much more complex than the other variants. Not only changes must be assigned to agents or workers, just as the bin packing variation, but also they need to be sequenced and scheduled.

For this formulation example, the variable y_{it} represents the change i at transition t; z_{ibt} represents if change i is performed in bin or by worker b at transition t; Se_{ibpt} represents whether change i is the p^{th} performed change at bin b at transition t; $Tstart_{ibt}$ and $Tend_{ibt}$ represent the start and end time of change i at bin b at transition t.

The modeling can be divided in three parts: change allocation, sequencing, and scheduling. The allocation part is the same as the Bin Packing restriction. According to Equation 4.19, the performed changes (y) must be assigned to one of the bins, implemented by variable z.

$$\sum_{b} z_{ibt} = y_{it} \quad \forall\ t \in \text{Transitions}, i \in \text{Changes} \tag{4.19}$$

The sequencing of changes is performed by linking the variables z and Se. Equation 4.20 assures that every change assigned to a bin is sequenced to one of the p positions in the ordering of changes. Inequality 4.21 assures that every position is occupied by a maximal of one change. Finally, Inequality 4.22 makes the changes occupy the earliest position available. For instance, if 3 changes are to be sequenced, without Inequality 4.22, the positions 1, 2, and 5 would be a feasible answer. With Expression 4.22, however, the

three changes would occupy the first 3 positions.

$$\sum_{p} Se_{ibpt} = z_{ibt} \qquad \forall\ i \in \text{Changes}, b \in \text{Bins}, t \in \text{Transitions} \tag{4.20}$$

$$\sum_{i} Se_{ibpt} \leq 1 \qquad \forall\ b \in \text{Bins}, p \in \text{Positions}, t \in \text{Transitions} \tag{4.21}$$

$$\sum_{i} Se_{ibpt} \leq \sum_{i} Se_{ib(p-1)t} \qquad \forall\ b \in \text{Bins}, p \in \text{Positions}, t \in \text{Transitions} : p > 1 \tag{4.22}$$

The scheduling part of the model is mainly based on the variables $Tstart$ and $Tend$. Inequality 4.23 assures that the end time of a change is at least the change start time plus the amount of time needed for the task. The time is controlled for each of the bins b in each of the transitions t. The time cost c_i is only accounted if the variable z_{ibt} is equal to one. Inequality 4.24 is a non-overlapping restriction. It assures that the start point of a change ($Tstart$) is equal or superior to the end point of another change ($Tend$). Note that this relation should only be valid for changes that are performed in the same bin at the same transition. Therefore, a sufficiently large value ($BigM$) is used to transform the inequality into a tautology otherwise. The expression $2 - Se_{i_1b(p-1)t} - Se_{i_2bpt}$ is only equal to zero (and therefore activate the inequality) if both $Se_{i_1b(p-1)t} = 1$ and $Se_{i_2bpt} = 1$. That is, only if change i_1 is performed at buffer b and transition t in the $p-1$ position and change i_2 is a subsequent change at the same bin and transition. Finally, Inequality 4.25 controls the precedence restrictions between changes. The same logical expression is used as in Inequality 4.24, but it controls whether both changes i_1 and i_2 are performed in the same transition. If they are, despite in which bin they are, the precedence restriction ($Prec$) must be respected.

$$Tend_{ibt} \geq Tstart_{ibt} + z_{ibt} \cdot c_i \quad \forall\ i \in \text{Changes}, b \in \text{Bins}, t \in \text{Transitions} \tag{4.23}$$

$$Tstart_{i_2bt} \geq Tend_{i_1bt} - BigM \cdot (2 - Se_{i_1b(p-1)t} - Se_{i_2bpt}) \qquad \forall\ i_1, i_2 \in \text{Changes}, b \in \text{Bins}, t \in \text{Transitions}, p \in \text{Positions} : p > 1 \tag{4.24}$$

$$\sum_{b} Tstart_{i_2bt} \geq \sum_{b} Tend_{i_1bt} - BigM \cdot (2 - \sum_{b} z_{i_1bt} - \sum_{b} z_{i_2bt}) \qquad \forall\ (i_1, i_2) \in \text{Prec}, t \in \text{Transitions} \tag{4.25}$$

The implementation effort for the scheduling can be then defined in terms of $Tend$, as it is shown in Inequality 4.26.

$$Effort \geq Tend_{ibt} \quad \forall\ i \in \text{Change}, b \in \text{Bin}, t \in \text{Transition} \tag{4.26}$$

Expressions 4.27 to 4.29 can be used as domain cuts. Inequality 4.27 assures that no change that has not been assigned is scheduled. The variable $Tend$ is set to zero if the correspondent z variable is false. Furthermore, Expressions 4.28 and 4.29 represent lower bounds for the implementation effort. Inequality 4.28 is the bound calculated by the sum of the changes, while Inequality 4.29 is the bin packing bound. Although none of these expressions are necessary, they can be used to reduce the computational burden. In preliminary tests, the scheduling formulation proved to be substantially more difficult to solve than the other presented variants.

$$Tend_{ibt} \leq BigM \cdot z_{ibt} \qquad \forall\ i \in \text{Change}, b \in \text{Bin}, t \in \text{Transition} \tag{4.27}$$

$$Effort \geq \frac{\sum_i y_{i,t} \cdot c_i}{\text{Number of Bins}} \qquad \forall t \in \text{Transitions} \tag{4.28}$$

$$Effort \geq \sum_i z_{ibt} \cdot c_i \qquad \forall b \in \text{Bins}, t \in \text{Transitions} \tag{4.29}$$

In this section only four ideas of transition modeling are presented. Depending on the nature of the problem, more variants can be defined.

In Chapter 5, the modeling concepts proposed in this model guide are used for the formulation of the Assembly Line Implementation Problem and variants.

Chapter 5

Problems' Formulations

This chapter presents the modeling of the Assembly Line Implementation Problem. Some variations and the variable reduction techniques are also present in the chapter.

5.1 Tuple Notation

In mathematical modeling, it is not uncommon to have restrictions applied to a sparse set of variables. The sparse sets can be treated with set operations or represented by tuples. Tuples are ordered sets of elements. They may contain indexes, values or descriptions and can be used as lists of useful values.

The application of tuples in mathematical modeling is exemplified with a simple assignment model. Suppose we have to assign tasks from set L to people from set A. The assignment is formulated as maximizing people's preference or minimizing costs of the assignment of one task for each person.

$$A = \{1, 2, 3, 4\}, B = \{1, 2, 4\}, C = \{1, 3\}, L = \{1, 2, 3, 4\}, M = \{1, 2, 3\}, N = \{2, 4\}$$

Imagine, however, that not every person can perform every task. Suppose that in set A, there are people with different abilities. Set B, for instance, could contain the people with managerial abilities, while set C contained people with engineering capabilities. Similarly with the task sets, we divide set L into sets M and N. Set M contains the tasks that can be assigned to people with managerial abilities and set N can be performed by

people of set C.

The decision variable of the problem is x_{ij}, a binary variable indicating whether person i is responsible for task j. Table 5.1 contains the restrictions and the equivalent model when the incompatibilities are not considered. Note that the underline variables are incompatible. For instance, person 3 is on set C, while task 1 is only on set M. Therefore, x_{31} would not be a feasible assignment.

Table 5.1: Assignment problem formulation without the incompatible variables removal.

Implicit Form	Explicit Form
$\sum_{i \in A} x_{ij} = 1 \quad \forall j \in L$	$\begin{cases} x_{11} + x_{21} + \underline{x_{31}} + x_{41} = 1 & (j = 1) \\ x_{12} + x_{22} + x_{32} + x_{42} = 1 & (j = 2) \\ x_{13} + x_{23} + \underline{x_{33}} + x_{43} = 1 & (j = 3) \\ x_{14} + \underline{x_{24}} + x_{34} + \underline{x_{44}} = 1 & (j = 4) \end{cases}$
$\sum_{j \in L} x_{ij} = 1 \quad \forall i \in A$	$\begin{cases} x_{11} + x_{12} + x_{13} + x_{14} = 1 & (i = 1) \\ x_{21} + x_{22} + x_{23} + \underline{x_{24}} = 1 & (i = 2) \\ \underline{x_{31}} + x_{32} + \underline{x_{33}} + x_{34} = 1 & (i = 3) \\ x_{41} + x_{42} + x_{43} + \underline{x_{44}} = 1 & (i = 4) \end{cases}$

There are two forms of dealing with these incompatibility issues. The simplest form is using the objective function to force the incompatible values to assume 0. In a maximization problem, those underlined variables could be given a negative value of preference in the objective function. If this value is negative enough, the optimal answer would not consider the incompatible variable. A problem rises when the incompatible variables are not the main variables of the problem, but only part of a more complete formulation. In those cases, the determination of the objective values is harder and can produce modeling errors.

The second form is modeling the problem without the inexistent assignment options. This form requires set or tuples operations and are more difficult to implement. However, a sparse modeling contains less variables and can better represent the real problem in the modeling.

Table 5.2 contains a formulation for the same problem without the incompatible tasks. The restrictions, however, are written using set operations. The two original restrictions from Table 5.1 (every task should be assigned to one person and every person

should perform one task) are now divided in six expressions.

Table 5.2: Assignment problem formulation removing the incompatible variables with logical operations.

Number	Implicit Form	Explicit Form
1	$\sum_{i \in B} x_{ij} = 1 \quad \forall j \in M - N$	$x_{11} + x_{21} + x_{41} = 1 \quad (j = 1)$ $x_{13} + x_{23} + x_{43} = 1 \quad (j = 3)$
2	$\sum_{i \in B \cup C} x_{ij} = 1 \quad \forall j \in M \cap N$	$x_{12} + x_{22} + x_{32} + x_{42} = 1 \quad (j = 2)$
3	$\sum_{i \in C} x_{ij} = 1 \quad \forall j \in N - M$	$x_{14} + x_{34} = 1 \quad (j = 4)$
4	$\sum_{j \in M} x_{ij} = 1 \quad \forall i \in B - C$	$x_{21} + x_{22} + x_{23} = 1 \quad (i = 2)$ $x_{41} + x_{42} + x_{43} = 1 \quad (i = 4)$
5	$\sum_{i \in M \cup N} x_{ij} = 1 \quad \forall i \in B \cap C$	$x_{11} + x_{12} + x_{13} + x_{14} = 1 \quad (i = 1)$
6	$\sum_{i \in N} x_{ij} = 1 \quad \forall i \in C - B$	$x_{32} + x_{34} = 1 \quad (i = 3)$

The column *Number* in Table 5.2 is used to classify the expressions. For both people and tasks, we have to treat each capability separately and the intersection of sets. In Expression Number 1, we treat only the exclusive managerial tasks (set $M - N$). The restriction implies that each task exclusive to managerial people (set $M - N$) has to be assigned to a person with managerial abilities (set B). The Expression Number 2 treats the tasks that can be performed by both managerial and engineering personal. Therefore, each task that takes part of both sets (set $M \cap N$) must be performed by one person of any capability (set $B \cup C$). The third expression is related to the exclusive engineering activities (set $N - M$). The constraint assures each exclusive engineering task is performed by a person with engineering capabilities (set C). These three expressions are needed to model the first restraint of the problem: every task should be assigned to a person. A similar reasoning is applied to the Expressions 4-6 for the restriction that every person should be responsible for a task.

Note that with the division of sets in only two categories the model has become more demanding to formulate and understand. With more sets and other set operations, several expressions must be written to represent only a simple restriction.

The tuple notation is especially useful for the compact formulation of sparse sets.

Herein, the same formulation would be modeled in terms of tuple notation. A tuple FA can be defined containing all the Feasible Assignments derived from the set operations. The feasible assignment options are every combination of a person with managerial skills (set B) and tasks of set M plus the combinations of the engineering skilled people (set C) with the tasks of set N.

$$FA = \{(i,j) \mid i \in B, j \in M\} \cup \{(i,j) \mid i \in C, j \in N\}$$

The tuple FA represents a list of assignment possibilities:

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

With the use of the tuple FA, the sparse formulation can be easily expressed using only the two original restrictions but applied only to the tuple FA. Note that in the formulation from Table 5.3, every sum function in the restrictions is written in terms of FA. This means that only the listed elements of FA are considered in the expression. Using the first expression as an example with $j = 1$ we have: $\sum_{(i,1)\in FA} x_{i1} = 1$. The elements of FA that have the form $(i,1)$ are: $(1,1)$, $(2,1)$ and $(4,1)$. Therefore, these are the elements that should appear in the explicit form of the formulation (x_{11}, x_{21} and x_{41}).

Table 5.3: Assignment problem formulation removing the incompatible variables with the use of tuples.

Implicit Form	Explicit Form
$\sum_{(i,j)\in FA} x_{ij} = 1 \quad \forall j \in L$	$\begin{cases} x_{11} + x_{21} + x_{41} = 1 & (j=1) \\ x_{12} + x_{22} + x_{32} + x_{42} = 1 & (j=2) \\ x_{13} + x_{23} + x_{43} = 1 & (j=3) \\ x_{14} + x_{34} = 1 & (j=4) \end{cases}$
$\sum_{(i,j)\in FA} x_{ij} = 1 \quad \forall i \in A$	$\begin{cases} x_{11} + x_{12} + x_{13} + x_{14} = 1 & (i=1) \\ x_{21} + x_{22} + x_{23} = 1 & (i=2) \\ x_{32} + x_{34} = 1 & (i=3) \\ x_{41} + x_{42} + x_{43} = 1 & (i=4) \end{cases}$

5.2 Implementation Sequencing Examples for ALBP

The Assembly Line Balancing Problem (ALBP) inspired the modeling and the application of the implementation optimization here presented. An ALBP consists in dividing the workload of an assembly line among multiple workers. The problem's variables correspond to the assignment of each individual task into workstations. The assignment variables, here represented by x, represent medium or long-term decisions of the line's configuration. Changing assignments, represented by y variables, implicate in reconfiguration costs. In practice, a new assignment might require moving and reinstalling equipment, training employees, reprogramming robots, redefining logistics, and/or updating the process documentation.

Performing changes on assembly lines is strongly restricted. The industrial production usually occurs in multiple work-shifts, so that there is not many opportunities for long lasting interventions. Small adjustments can be made during night-shifts, holidays or weekends while major changes may be restricted to maintenance stoppages. Dividing the reconfiguration changes into small but serial implementation packages can improve the feasibility of improvement projects. Nevertheless, the line must be able to operate in all intermediary production periods.

Due to the above considerations, the Assembly Line Balancing problem presents the conditions stated on the Modeling Guide Chapter (Chapter 4): it makes sense to model and optimize the implementation steps for an ALBP. The state variables (x) represent permanent decisions, for which alterations (variables y) result in costs. Furthermore, the necessary changes can be divided in implementation periods. The problem can be discretized in stages and transition periods. One normal operation pattern could consist in production periods during workdays and implementation efforts during weekends, for example.

In this section, the modeling of the implementation optimization is exemplified using four ALBP models. Section 5.2.1 contains the formulation for the Simple Assembly Line Balancing Problem (SALBP). The model contains state variables representing the assignment of tasks into stations and transition variables to map assignment changes. Section 5.2.2 also contains a formulation for SALBP. However, instead of having state

and transition variables, the second formulation expresses stage and transition restrictions based only on transition variables. The first two models are both based on binary variables. An integer version of the SALBP is modeled in Section 5.2.3. In this model, both states and transitions are modeled with integer variables. The last variation consists in the integer-SALBP containing a mixed objective function. Section 5.2.4 presents a formulation in which reassignment tasks have part of the costs proportional to the number of task changes but also a fixed cost of reallocating a type of task.

5.2.1 SALBP

For this formulation, we suppose an assembly line operating in a suboptimal state. The best configuration for the line in terms of cycle time is known (e.g. by means of an MILP model solution), but the necessary changes would require an unfeasible line stoppage. The implementation effort, however, could be performed in smaller phases during weekends or small feasible production stoppages. The problem structure is illustrated in Figure 5.1. The final condition is reached after multiple transition and production periods.

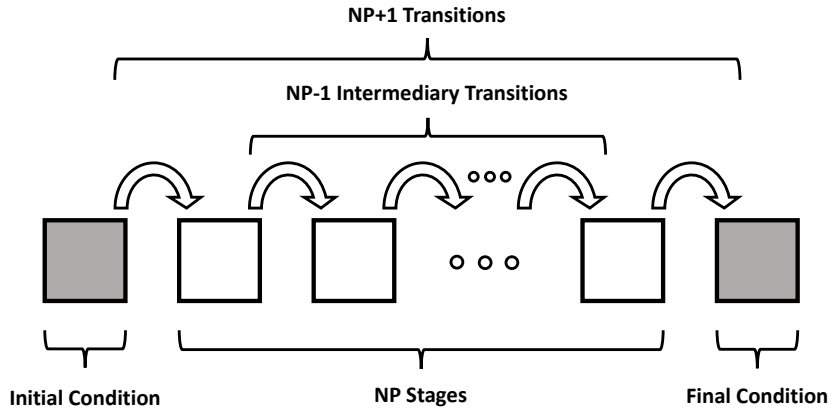

Figure 5.1: Diagram for the division of the problem in production (NP stages) and implementation ($NP + 1$ transition) periods.

The implementation optimization problem is defined as minimizing the implementation effort by dividing necessary changes along transition periods. Every production period, or intermediary state, must be able to produce at accepted output levels. For the presented model, we define that each stage or production period must be feasible. That

is, all tasks must be performed, precedence relations must be obeyed, and the production must respect a cycle time. On the other hand, the transition periods are the focus of the optimization. In this formulation we present the second version of the problem (ALIP-2), in which we minimize the necessary effort for a given number of transition periods, as described by Sikora, Lopes and Magatão (2016). An adaptation for minimizing the number of periods given a maximal implementation effort (version 1, ALIP-1) is detailed in Section 5.3.

The parameters, sets and variables used for the formulation are described in Table 5.4. The nomenclature is maintained for the other models. The ALBP is defined based on the set $Tasks$ of tasks t ranging from 1 to NT and the set $Stations$ of workstations s from 1 to NS. For the transient part of the problem, sets for the stages and transitions are also defined. Once the indexes t and s are already used for tasks and stations, the sets $Periods$ for the stages and $Transitions$ for the assignment changes are represented by indexes p and c. The problem is defined according to Figure 5.1: for NP stages, $NP+1$ transition periods are defined. The set $MidTransitions$ is used to relate to the intermediary transitions. Once the first and last transitions relate stages to the initial and final condition, they need specific restrictions in the formulation. The parameter TD_t represents the duration time of each task t, while the cost of reassigning a task t to a new workstation is defined by c_t. For this model, every stage is subjected to a unique value of cycle time (CT_{max}). This parameter, however, can also be defined by a different value for each period.

As explained in Section 5.1, variables and restrictions can be expressed by tuple notation. For this formulation, a tuple is defined by the set of Feasible Task Assignment (FTA). That is, the model's elements are defined within the sparse tuple containing only valid assignments. For SALBP, the set FTA can be defined in terms of the Earliest Station (E_t) and Latest Station (L_t) a task can be assigned (Patterson and Albracht, 1975). The definition of E_t, L_t and FTA is discussed in Section 5.4. For the understanding of the model, however, it is sufficient to acknowledge that FTA consists of all feasible pairs of tasks t and stations s. Another important data is the precedence graph (A). From the precedence diagram the pairs of tasks with precedence relations ($Prec$) are determined. Finally, the parameters IA_{ts} and FA_{ts} contain the initial and final assignment of tasks

Table 5.4: Parameters, sets and variables used in the formulation for ALIP based on SALBP.

Parameter	Dimension	Description
NT	-	Number of tasks in the line
NS	-	Number of stations in the line
NP	-	Number of periods for the implementation planning
TD_t	t	Duration time of each task t
c_t	t	Cost of moving task t to a new station
CT_{max}	-	Limiting cycle time considered in each period
E_t	t	Earliest station in which task t can be assigned
L_t	t	Latest station in which task t can be assigned
A	-	Precedence relation graph for the tasks
IA_{ts}	t, s	Binary value that represents the initial assignment of tasks
FA_{ts}	t, s	Binary value that represents the final assignment of tasks
Set	**Element**	**Description**
$Tasks$	t	Set of tasks to be performed in the line ($1...NT$)
$Stations$	s	Set of line's stations ($1...NS$)
$Periods$ (P)	p	Set of considered planning periods ($1...NP$)
$Transitions$ (C)	c	Set of considered implementation steps ($1...NP+1$)
$MidTransitions$	m	Set of intermediary transitions ($2...NP$)
FTA	t, s	Set of Feasible Task Assignments: contains the stations E_t to L_t
$Prec$	t_1, t_2	Set of all precedence relations for task pairs t_1, t_2 in A
Variable [Type]	**Set**	**Description**
x_{tsp} [binary]	FTA, P	$\begin{cases} 1, \text{if task } t \text{ is assigned to station } s \text{ at period } p \\ 0, \text{otherwise} \end{cases}$
y^{+}_{tsc} [binary]	FTA, C	$\begin{cases} 1, \text{if task } t \text{ moved to station } s \text{ at transition } c \\ 0, \text{otherwise} \end{cases}$
$Effort$ [cont.]	-	Effort or cost of implementing a transition

to station s. For this first model, IA_{ts} and FA_{ts} are binary values that indicate whether task t is assigned to station s in the initial and final configuration, respectively.

Three groups of variables are used to define the model. In each production phase, the stage can be defined by the assignment of tasks to stations. The task allocation is enough to observe the precedence relations and determinate the cycle time. Variable x_{tsp} represents whether task t is assigned to station s at period p, defining the stages.

The second group of variables is used for modeling the transitions. The formulation objective is to measure the implementation effort in order to minimize it. As seen in Chapter 4, just a lower bound for the reconfiguration cost for each implementation is needed. Therefore, it is enough to satisfy the expression $\{\text{New Assignment} \rightarrow y = 1\}$ in the implication form. Note that if no new assignment occurs, y is free to assume either 0 or 1. Along with the minimization objective, however, it is not necessary to use more restricted expressions.

In an ALBP, every task has to be performed exactly once. Therefore, each task that is removed from a station is assigned to another. It is sufficient, then, to map only the insertion (or only removal) of tasks. In these conditions, the variable y^{+}_{tsc} is used to determinate whether task t is **added** to station s at the transition c.

Finally, the third variable ($Effort$) measure the reconfiguration costs of the most loaded implementation period. This variable is used to determinate the maximal amount of effort per period the implementation would require. The objective function of the problem is described in Expression 5.1. The variable $Effort$ represents the maximal cost of any given implementation phase. Therefore, by minimizing $Effort$, the model balances the changes among the implementation phases.

$$\text{Minimize: } Effort \tag{5.1}$$

The restrictions used to describe the problem can be divided in two modules. The first assures the feasibility of each production phase. The second models and measures the implementation changes in every transition.

The used ALBP model is based on the formulation from Patterson and Albracht (1975). The model's first equation is the Occurrence Restriction. According to Equation 5.2, every task should be assigned to exactly one station for every considered period. The Cycle Time Restriction is represented by Inequality 5.3. The sum of the processing time in each station must be lower than the target cycle time for each stage.

$$\sum_{(t,s)\in FTA} x_{tsp} = 1 \qquad \forall\ t \in Tasks, p \in Periods \tag{5.2}$$

$$\sum_{(t,s)\in FTA} x_{tsp} \cdot TD_t \leq CT_{max} \qquad \forall\ s \in Stations, p \in Periods \tag{5.3}$$

There are multiple forms of modeling the precedence relations. Inequality 5.4 was firstly proposed by Bowman (1960) and improved by White (1961). Using this inequality, a dependent task t_2 can only be assigned to station s if task t_1 is assigned to any station k so that $k \leq s$. This condition is achieved by forcing the variable x_{t_2sp} to be smaller or

equal to the sum of the variables x_{t_1kp} for all stations k earlier than s.

$$x_{t_2sp} \leq \sum_{(t_1,k)\in FTA:\ k\leq s} x_{t_1kp} \qquad \forall\ (t_1,t_2) \in Prec, p \in Periods, (t_2,s) \in FTA \tag{5.4}$$

A second variant for the precedence relation formulation is presented by Thangavelu and Shetty (1971) and Patterson and Albracht (1975). Although the authors expressed the inequality in different forms, both can be rewritten as Expression 5.5. In this expression, for each pair of tasks with precedence relations, the product $x_{tsp} \cdot s$ for the dependent task must be greater or equal than the product of the precedent task. This means that for the assignment in which x_{tsp} is true, the value of the station (s) must be greater for the dependent task.

$$\sum_{(t_2,s)\in FTA} x_{t_2sp} \cdot s \geq \sum_{(t_1,s)\in FTA} x_{t_1sp} \cdot s \qquad \forall\ (t_1,t_2) \in Prec, p \in Periods \tag{5.5}$$

The last formulation is due to Ritt and Costa (2015). The expression is similar to Inequality 5.4, but sums variables x_{tsp} in both sides of the inequality. According to Ritt and Costa (2015), Inequality 5.6 dominates Inequalities 5.4 and 5.5. However, Ritt and Costa's formulation needs more constraints than the expression 5.5. Although Inequality 5.6 produces a tighter linear relaxation, it does not necessary mean it is the most computationally effective formulation for the generated MILP model.

$$\sum_{(t_2,k)\in FTA:\ k\leq s} x_{t_2kp} \leq \sum_{(t_1,k)\in FTA:\ k\leq s} x_{t_1kp}$$
$$\forall\ (t_1,t_2) \in Prec, p \in Periods, (t_1,s)\ \&\ (t_2,s) \in FTA \tag{5.6}$$

The transition module has to link variables from adjacent production periods. As discussed previously, the formulation has to ensure y^+ is equal to one when a task is added to a station. Once only the implication of this expression is needed, the suitable expression to link binary-state to binary-transition variables is the first found in Table 4.3. Equation 5.7 models the implementation changes for the intermediary periods. From this inequality, y^+_{tsm} has to assume 1 if a task t is not assigned to station s at period $m-1$ ($x_{ts(m-1)} = 0$) but is assigned to that station at period m ($x_{tsm} = 1$). Inequalities 5.8 and 5.9 represent the same expression, but are used for the first and last period along with

the initial and final assignment parameters.

$$y^{+}_{tsm} \geq x_{tsm} - x_{ts(m-1)} \qquad \forall\ (t,s) \in FTA, m \in MidTransitions \tag{5.7}$$

$$y^{+}_{ts1} \geq x_{ts1} - IA_{ts} \qquad \forall\ (t,s) \in FTA \tag{5.8}$$

$$y^{+}_{ts(NP+1)} \geq FA_{ts} - x_{ts(NP)} \qquad \forall\ (t,s) \in FTA \tag{5.9}$$

Here, we suppose that the total effort is the sum of the cost of reinstalling every individual task. Other formulations could consider synergy effects or economies of scale. Expression 5.10 assures the variable $Effort$ assumes the value of the most costly transition. The objective function is then balancing the implementation cost along the multiple interventions.

$$\sum_{(t,s)\in FTA} y^{+}_{tsc} \cdot c_t \leq Effort \qquad \forall\ c \in Transitions \tag{5.10}$$

5.2.2 SALBP with only transition variables

In a SALBP instance, the information of which task is assigned to which station is enough to model all feasibility conditions. The SALBP is part of a set of problems which are entirely defined by the assignment variables. Such problems can be formulated without explicit state variables, because it is possible to express assignment variables as functions of transition variables.

A state variable can be determined by its initial value and the effect of every change occurred until the period of interest. As expressed bellow, both the augmenting (y^{+}) and diminishing (y^{-}) of values should be considered to express state variables as a sum of transition variables as defined in Equation 5.11. An example of the use of the variable substitution is shown in Figure 5.2.

$$x_{tsp} = IA_{ts} + \sum_{k\in Periods:\ k\leq p} (y^{+}_{tsk} - y^{-}_{tsk}) \quad \forall t \in \text{Tasks}, s \in \text{Stations}, p \in \text{Periods} \tag{5.11}$$

This section presents a SALBP model with only transition variables. The model is equivalent to the one presented in Section 5.2.1, but the substitution expressed in Equation 5.11 is applied to all state variables. The parameters and sets remain unaltered and refer to Table 5.4. On the other side, the variables should be substituted by the ones in Table 5.5. Note that both the transition variables for adding (y^{+}) and removing (y^{-}) tasks are needed.

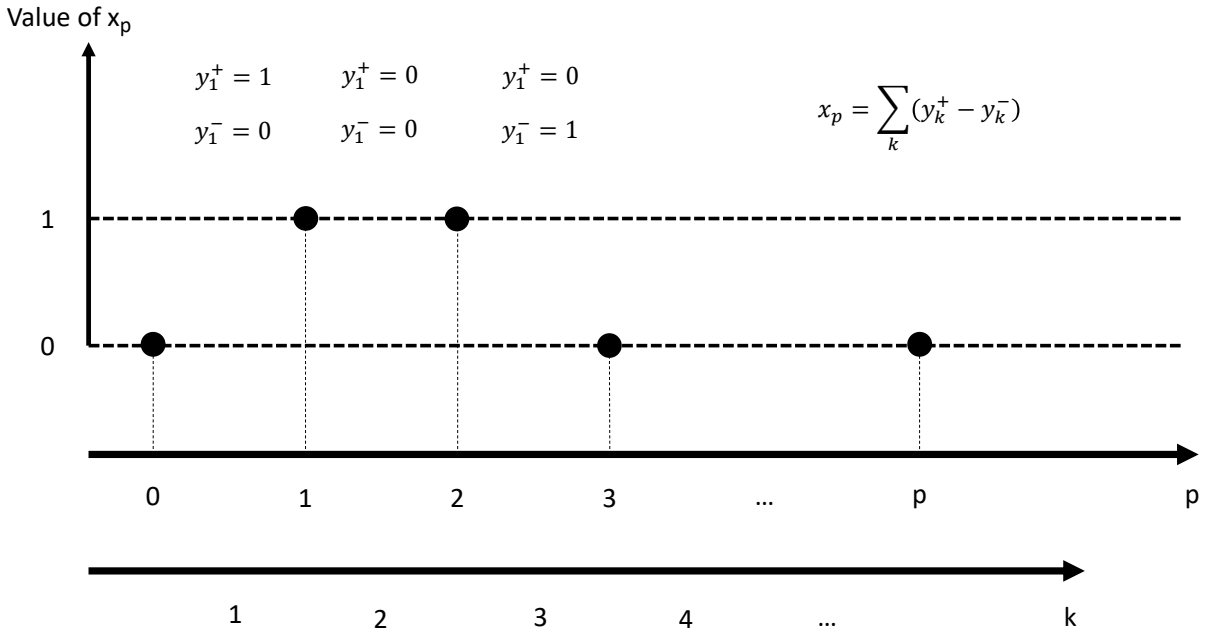

Figure 5.2: Example of the use of the variable substitution.

Table 5.5: Variables used for the formulation of ALIP based SALBP with only transition variables.

Variable [Type]	Set	Description
y^+_{tsc} [binary]	FTA, C	$\begin{cases} 1, \text{if task } t \text{ moved to station } s \text{ at transition } c \\ 0, \text{otherwise} \end{cases}$
y^-_{tsc} [binary]	FTA, C	$\begin{cases} 1, \text{if task } t \text{ moved away from station } s \text{ at transition } c \\ 0, \text{otherwise} \end{cases}$
$Effort$ [cont.]	-	Effort or cost of implementing a transition

The Occurrence Restriction presents itself in a different form. Once all tasks should be performed in the initial configuration, assuring that a task removed from a station is assigned to another station is enough to guarantee all tasks are performed in every period (Equation 5.12). Furthermore, Constraint 5.13 is necessary to assure no negative assignment is possible in the formulation.

$$\sum_{(t,s)\in FTA} y^+_{tsc} = \sum_{(t,s)\in FTA} y^-_{tsc} \qquad \forall\ t \in Tasks, c \in Transitions \tag{5.12}$$

$$IA_{ts} + \sum_{k\in Transitions: k\leq c} (y^+_{tsk} - y^-_{tsk}) \geq 0 \qquad \forall\ (t,s) \in FTA, c \in Transitions \tag{5.13}$$

The Cycle Time Restriction (Inequality 5.14) as well as the presented Precedence Constraints (Expressions 5.15, 5.16 and 5.17) are equivalent to the formulation of Section

5.2.1 with the variable substitution from Expression 5.11.

$$\sum_{(t,s)\in FTA}\left(IA_{ts}+\sum_{k\in Periods:\ k\leq p}(y^{+}_{tsk}-y^{-}_{tsk})\right)\cdot TD_t \leq CT_{max} \qquad (5.14)$$
$$\forall\ s\in Stations, p\in Periods$$

$$IA_{t_2s}+\sum_{k\in Periods:\ k\leq p}(y^{+}_{t_2sk}-y^{-}_{t_2sk}) \leq \sum_{(t_1,j)\in FTA:\ j\leq s}\left(IA_{t_1j}+\sum_{k\in Periods:\ k\leq p}(y^{+}_{tjk}-y^{-}_{tjk})\right) \qquad (5.15)$$
$$\forall\ (t_1,t_2)\in Prec, p\in Periods, (t_2,s)\in FTA$$

$$\sum_{(t_2,s)\in FTA}\left(IA_{t_2s}+\sum_{k\in Periods:\ k\leq p}(y^{+}_{t_2sk}-y^{-}_{t_2sk})\right)\cdot s \geq \sum_{(t_1,s)\in FTA}\left(IA_{t_1s}+\sum_{k\in Periods:\ k\leq p}(y^{+}_{t_1sk}-y^{-}_{t_1sk})\right)\cdot s \qquad (5.16)$$
$$\forall\ (t_1,t_2)\in Prec, p\in Periods$$

$$\sum_{(t_2,j)\in FTA:\ j\leq s}\left(IA_{t_2j}+\sum_{k\in Periods:\ k\leq p}(y^{+}_{t_2jk}-y^{-}_{t_2jk})\right) \leq \sum_{(t_1,k)\in FTA:\ j\leq s}\left(IA_{t_1j}+\sum_{k\in Periods:\ k\leq p}(y^{+}_{t_1jk}-y^{-}_{t_1jk})\right) \qquad (5.17)$$
$$\forall\ (t_1,t_2)\in Prec, p\in Periods, (t_1,s)\ \&\ (t_2,s)\in FTA$$

As the Occurrence Restriction, the formulation for transitions also changes. The transition variables cannot be expressed in terms of a difference of state variables. Equation 5.18 assures that the applied changes during the implementation result in the final condition. In other words, the sum of the changes in each assignment applied to the initial configuration must result in the aimed configuration.

$$IA_{ts}+\sum_{c\in Transitions}(y^{+}_{tsc}-y^{-}_{tsc})=FA_{ts} \qquad \forall\ (t,s)\in FTA \qquad (5.18)$$

Finally, the cost function remains the same. Once the insertion and removal of tasks are symmetrical, either y^{+} or y^{-} could be used in Expression 5.19

$$\sum_{(t,s)\in FTA} y^{+}_{tsc}\cdot c_t \leq Effort \qquad \forall\ c\in Transitions \qquad (5.19)$$

5.2.3 Integer SALBP

In assembly lines with several tasks with the same properties (precedence relations and processing times) it may be advantageous to gather them in task groups. Sikora, Lopes, Schibelbain and Magatão (2017) presented an integer variable based model for the Simple Assembly Line Problem. In this formulation, the decision variables are not binary values that represent whether a task is assigned to a station but integer variables defining how many copies of each task are assigned to each station. Spot welding or screw operations are examples of identical tasks that justify the use of the model. The binary and the integer representations are exemplified in precedence diagrams in Figures 5.3 and 5.4 respectively. Note that the binary representation contains more tasks, while the integer representation has an extra information on the precedence diagram: the number of copies of each task (on the top-left side of each node). The integer formulation, however, do not require that entire groups of tasks must be assigned to a single station. The multiple copies of tasks can be assigned to different stations, just as the binary formulation. The advantage of the model with integer variables lays on a compact formulation.

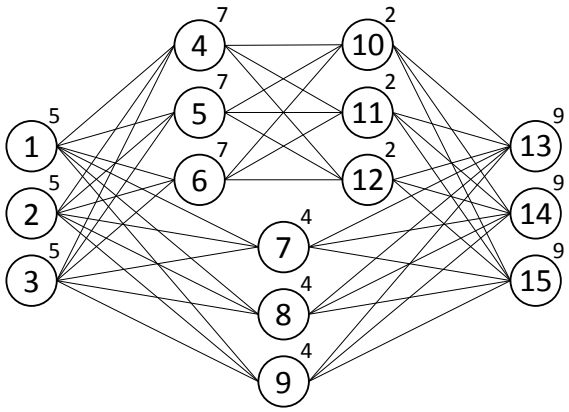

Figure 5.3: Example of binary representation of a balancing problem with repeated tasks.

The parameters, sets and variables for the formulation of the optimization of the implementation for an integer-based assembly line balancing problem are described in Table 5.6. Most of the model's elements are identical to those in Table 5.4. The specific parameters for the integer version are that each task has a number of copies (N_t) and an upper bound for the assignments of copies of tasks in each station (M_{ts}). For the variables, instead of the binary x_{tsp} the main decision variable is defined as an integer

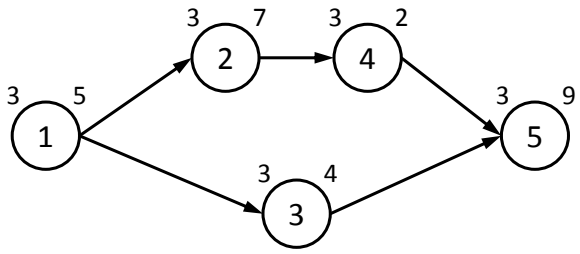

Figure 5.4: Example of binary representation of a balancing problem with repeated tasks.

z_{tsp}. For the precedence relations, an auxiliary binary variable is necessary: x^c_{tsp} is used to define in which stations all copies of a given task are already completed.

As part of the variable reduction described in Section 5.4, an upper bound for the number of copies can be calculated for each task-station combination. The value M_{ts} can be used as a bound to the variables z_{tsp} (Inequality 5.20). The integer version of the Occurrence Restriction (Equation 5.21) has to assure every N_t copies of each task are performed in each period. The Cycle Time Restriction is identical to the binary version, as expressed in Inequality 5.22.

$$z_{tsp} \leq M_{ts} \qquad \forall\ (t,s) \in FTA, p \in Periods \tag{5.20}$$

$$\sum_{(t,s)\in FTA} z_{tsp} = N_t \qquad \forall\ t \in Tasks, p \in Periods \tag{5.21}$$

$$\sum_{(t,s)\in FTA} z_{tsp} \cdot TD_t \leq CT_{max} \qquad \forall\ s \in Stations, p \in Periods \tag{5.22}$$

For the correct modeling of precedence relations using integer assignment variables, an auxiliary binary variable (x^c_{tsp}) is used. The integer precedence relation is enunciated differently than the binary one. Any copy of a dependent task can only be assigned if **all copies** of the precedence task are already assigned. The auxiliary variable x^c_{tsp} is used to determinate if all copies of a task t are already assigned up to station s. Expression 5.23 is used to force x^c_{tsp} to assume 1 when all copies of a given task t is performed in stations up to s. Inequality 5.24, on the other side, assures that x^c_{tsp} assumes 0 when not all N_t copies of task t are assigned. The precedence relation is then expressed by Inequality 5.25. The auxiliary variable x^c_{tsp} is used to allow only the assignment of tasks when the

Table 5.6: Parameters, sets and variables used in the formulation for ALIP based on the Integer SALBP.

Parameter	Dimension	Description
NT	-	Number of task types in the line
NS	-	Number of stations in the line
NP	-	Number of periods for the planning
N_t	t	Number of copies of each task t
TD	t	Duration time of each copy of task t
c_t	t	Cost of moving a copy of task t to a new station
CT_{max}	-	Limiting cycle time considered in each period
E_t	t	Earliest station in which task t can be assigned
L_t	t	Latest station in which task t can be assigned
A	-	Precedence relation graph for the tasks
IA_{ts}	t, s	Integer value representing the initial assignment of copies of tasks
FA_{ts}	t, s	Integer value representing the final assignment of copies of tasks
M_{ts}	t, s	Upper bound for the assignment of copies of task t to station s
Set	**Element**	**Description**
$Tasks$	t	Set of tasks to be performed in the line ($1...NT$)
$Stations$	s	Set of line's stations ($1...NS$)
$Periods$ (P)	p	Set of considered planning periods ($1...NP$)
$Transitions$ (C)	c	Set of considered implementation steps ($1...NP+1$)
$MidTransitions$	m	Set of intermediary transitions ($2...NP$)
FTA	t, s	Set of Feasible Task Assignment: contains the stations E_t to L_t
$Prec$	t_1, t_2	Set of all precedence relations for task pairs t_1, t_2 in A
Variable [Type]	**Set**	**Description**
z_{tsp} [integer]	FTA, P	n, if n copies of task t are assigned to station s at period p 0, otherwise
y^+_{tsc} [integer]	FTA, C	n, if n copies of task t are moved to station s at transition c 0, otherwise
x^c_{tsp} [binary]	FTA, P	1, if all copies of task t are concluded up to station s at period p 0, otherwise
$Effort$ [cont.]	-	Effort or cost of implementing a transition

precedent tasks are completed.

$$x^c_{tsp} + N_t - 1 \geq \sum_{\substack{k \in Stations \\ (t,k) \in FTA:\ k \leq s}} z_{tkp} \qquad \forall\ (t,s) \in FTA, p \in Periods \tag{5.23}$$

$$x^c_{tsp} \leq \sum_{\substack{k \in Stations \\ (t,k) \in FTA:\ k \leq s}} \frac{z_{tkp}}{N_t} \qquad \forall\ (t,s) \in FTA, p \in Periods \tag{5.24}$$

$$z_{t_2sp} \leq x^c_{t_1kp} \cdot N_{t_2} \qquad \forall\ (t_1,t_2) \in Prec, p \in Periods, (t_1,s)\ \&\ (t_2,s) \in FTA \tag{5.25}$$

The transition module is also similar to the binary formulation. For this model, the first expression form found in Table 4.5 is used. Expression 5.26 assures that if tasks copies of task t are added to station s at transition c, then y^+_{tsm} must assume a value greater or equal to the difference of assigned tasks. Once the objective is minimize the

sum of y^+, more complex modeling options from Table 4.5 are not necessary. Inequalities 5.27 and 5.28 are used to model the first and last transition, respectively.

$$y^+_{tsm} \geq z_{tsm} - z_{ts(m-1)} \qquad \forall\ (t,s) \in FTA, m \in MidTransitions \tag{5.26}$$

$$y^+_{ts1} \geq z_{ts1} - IA_{ts} \qquad \forall\ (t,s) \in FTA \tag{5.27}$$

$$y^+_{ts(NP+1)} \geq FA_{ts} - z_{ts(NP)} \qquad \forall\ (t,s) \in FTA \tag{5.28}$$

For this formulation, the cost is assumed to be proportional to each reassigned copy of a task. Expression 5.29 assures that the variable $Effort$ represents the sum of the cost of the most difficult implementation phase. An objective function such as minimizing the value of $Effort$ would result in the balancing of implementation workload between the implementation periods.

$$\sum_{(t,s)\in FTA} y^+_{tsc} \cdot c_t \leq Effort \qquad \forall\ c \in Transitions \tag{5.29}$$

5.2.4 Integer SALBP with Multi Objective Function

The formulation of Section 5.2.3 presented a model for the implementation optimization applied to the integer version of SALBP. The cost of an implementation step is supposed to be the sum of the cost of reassigning copies of tasks. However, the cost may not be totally proportional to the number of tasks.

The programming of robots for spot welding can be used as example to a non-linearly proportional reassigning costs. Every added copy of a task requires only a small amount of work if there are already other copies assigned to the same robot. Identical tasks contain similar properties and the assignment of another copy can be just an extension of the already assigned copies. The assignment of the first copy to a new station, however, may be more complex than changing the number of copies in a station. In the spot welding example, an estimative of time needed to add a point within an already assigned region (a new copy) could be 30 minutes. Creating a new route to a different region (for a first copy of a task) could require about 90 minutes of programming.

This section presents a variation of the integer model of Section 5.2.3 with a multi objective function. The cost of an implementation phase is supposed to be based both on the cost of assigning copies and the cost of assigning a task type to a station. The Ex-

pressions 5.20 to 5.28 are also considered for this formulation. Only the extra constraints are here described.

Table 5.7 presents the parameters and variables that differ from the ones of Table 5.6. In this fourth formulation, the reassigning costs are separated in two groups: c_t^i as the cost of reassigning a copy of task t and c_t^b as the cost of adapting a station for the first copy of a task t. Therefore, the costs have an integer (c_t^i) and a binary (c_t^b) component.

Table 5.7: Parameters and variables used in the formulation for integer SALBP with mixed objective function.

Parameter	Dimension	Description
c_t^i	t	Cost of moving a copy of task t to a new station
c_t^b	t	Cost of adapting a station to a task t
IA_{ts}^b	t, s	Binary value that represents the initial assignment tasks
FA_{ts}^b	t, s	Binary value that represents the final assignment of tasks
Variable [Type]	Set	Description
x_{tsp} [binary]	FTA, P	$\begin{cases} 1, \text{if any copy of task } t \text{ is assigned to station } s \text{ at period } p \\ 0, \text{otherwise} \end{cases}$
y_{tsc}^b [binary]	FTA, C	$\begin{cases} 1, \text{if station } s \text{ needs to be adapted for task } t \text{ at transition } c \\ 0, \text{otherwise} \end{cases}$

In order to control the assignments of the first copies in a station, binary variables x_{tsp} and y_{tsc}^b are used. Variable x_{tsp} measures whether any copy of task t is assigned to station s at period p. The binary transition variable y_{tsc}^b is used to model the assignments of the first copy of tasks to stations. The parameters IA_{ts}^b and FA_{ts}^b are also needed for the correct modeling of the transitions. They are the binary equivalents of IA_{ts} and FA_{ts} of Table 5.6 and measure whether some copy of task t is assigned to station s at the initial and final configurations.

In order to obtain the formulation for the multi objective, the following expressions must be added to the model of Section 5.2.3. Inequalities 5.30 and 5.31 are used to link the binary assignment variables x_{tsp} to the integer assignment variable z_{tsp}. Expression 5.30 provides the upper bound, x_{tsp} has to assume the value 0 if no copy of task t is assigned ($z_{tsp} = 0$). As a lower bound, Inequality 5.31 assures x_{tsp} the value of 1 when at least one copy of task is assigned.

$$x_{tsp} \leq z_{tsp} \qquad \forall\ (t, s) \in FTA, p \in Periods \tag{5.30}$$

$$x_{tsp} \cdot M_{ts} \geq z_{tsp} \qquad \forall\ (t, s) \in FTA, p \in Periods \tag{5.31}$$

The expressions for the transitions for the binary variables are the same as those of Section 5.2.1. Inequality 5.32 is used for the intermediary transitions, while Inequalities 5.33 and 5.34 are used for the initial and final transition, respectively.

$$y^b_{tsm} \geq x_{tsm} - x_{ts(m-1)} \qquad \forall\ (t,s) \in FTA, m \in MidTransition \tag{5.32}$$

$$y^b_{ts1} \geq x_{ts1} - IA^b_{ts} \qquad \forall\ (t,s) \in FTA \tag{5.33}$$

$$y^b_{ts(NP+1)} \geq FA^b_{ts} - x_{ts(NP)} \qquad \forall\ (t,s) \in FTA \tag{5.34}$$

The main difference of this model lies on the calculation of $Effort$. For this variation, the cost is assumed to be the sum of the effort of reassigning the copies of tasks plus the cost of assigning new types of tasks to stations.

$$\sum_{(t,s)\in FTA} \left(y^+_{tsc} \cdot c^i_t + y^b_{tsc} \cdot c^b_t\right) \leq Effort \qquad \forall\ c \in Transitions \tag{5.35}$$

Minimizing $Effort$ would balance the implementation effort based on a compound expression for the cost in the transition periods.

5.3 Adaptation for Other Problem Types

The models in Section 5.2 presented the objective function of minimizing the implementation effort for a given number of periods. Furthermore, a maximal fixed cycle time is given for each production period.

These assumptions can be altered based on optimization objectives. Other forms of optimizing the transitions are described in Section 5.3.1. Section 5.3.2 contains the adaptations for the optimization of the stages given the transition conditions and Section 5.3.3 is presented for mixed stage and transition objective functions.

5.3.1 Optimizing Transitions

As seen in Section 3.4, the scheduling of implementation changes can be divided in four problem types. Based on the ALBP classification, they can be written as type-F, type-1, type-2 and type-E (Scholl, 1999). Type-F is the feasibility problem. For a given number of periods and a maximal amount of effort, the problem consists in finding a

feasible solution if it exists. The models presented in Section 5.2 can be adapted to the type-F problem by using $Effort$ as a parameter, instead of a variable and omitting the objective function.

The type-2 implementation problem has as objective the minimization of the effort for a given number of transitions. This version of the problem is modeled in the examples of Section 5.2.

The type-1 version of the problem consists in the minimization of the number of periods for a given amount of maximal effort for a period. For this problem, a known amount of workforce or allowable cost is imposed to every transition, while the objective is to minimize the project duration. The models of Section 5.2 can be adapted to the type-1 problem by adding a binary variable T_c that measures whether any change in performed in a transition. An upper bound for the number of periods is necessary to define NP and the variables. The cost equation can be adapted to a type-1 problem by using Expression 5.36. Inequality 5.37 can be used as a cut to simplify the problem. The expression forces the changes to occur in the first transitions. Finally, Expression 5.38 can be used as objective function: the number of transition periods used is minimized.

$$\sum_{(t,s)\in FTA} y^{+}_{tsc} \cdot c_t \leq Effort \cdot T_c \qquad \forall\ c \in Transitions \tag{5.36}$$

$$T_c \leq T_{c-1} \qquad \forall c \in Transitions, c \geq 2 \tag{5.37}$$

$$Minimize : \sum_{c\in C} T_c \tag{5.38}$$

The last variation is the type-E problem. From the balancing classification definition, E stands for efficiency. In an E-Instance, both the $Effort$ and the number of transition periods (NC) are variables. The objective is to minimize the total effort of the implementation ($NC \cdot Effort$). The problem only makes sense when the variables $Effort$ and NC are limited by bounds: an answer with only one transition would be an optimal answer if the amount of $Effort$ required is feasible. The type-E represents a non-linear problem, whose objective function can be expressed as Expression 5.39.

$$Minimize : \quad \sum_{c\in C} T_c \cdot Effort \tag{5.39}$$

Instead of solving a non-linear instance, one could separate the problem into several type-1 or type-2 instances, similarly to what can be applied to SALBP-E instances (Scholl,

1999). If the assumed interval for NC in the type-E instance is from 3 to 10 periods, for instance, solving 8 type-2 problems (with NC from 3 to 10) would be sufficient to reach the optimal answer.

5.3.2 Optimizing Stages

The models of Section 5.2 are formulated for the optimization of the transition periods. Another way to model the problem would be the maximization of the line's output during the implementation period. In this version, the transition restrictions only must be satisfied, while the cycle time is the subject of the objective function.

The models from Section 5.2 can be easily adapted by assuming $Effort$ as a parameter and CT_{max} as a problem variable. By assuming Expression 5.40 as objective function, the result is an implementation with the minimal impact (or the best improvement) of the cycle time during the implementation period.

$$Minimize: \qquad CT_{max} \tag{5.40}$$

Another optimization possibility considers the cycle time of each period separately. We name CT_p the line's cycle time at the production period p. A formulation with Expression 5.41 as objective function would result in the best average production levels throughout the implementation phase. A further adjustment can be made by assuming different weights for the production periods, as showed in Expression 5.42.

$$Minimize: \qquad \sum_{p \in NP} CT_p \tag{5.41}$$

$$Minimize: \qquad \sum_{p \in NP} k_p \cdot CT_p \tag{5.42}$$

5.3.3 Optimizing a Cost Function

There can also be a case in which both the production and implementation phases are subject of the optimization. To express both the cycle time, implementation periods and implementation effort in one expression, a cost function may be necessary. Coefficients k' and k'' must be assumed along with the Expressions 5.43, 5.44 or 5.45 depending on the optimization focus.

$$Minimize: \quad \sum_{p \in NP} k'_p \cdot CT_p + \sum_{p} k''_p \cdot T_p \tag{5.43}$$

$$Minimize: \quad \sum_{p \in NP} k'_p \cdot CT_p + k'' \cdot Effort \tag{5.44}$$

$$Minimize: \quad \sum_{p \in NP} k'_p \cdot CT_p + \sum_{p} k''_p \cdot T_p \cdot Effort \tag{5.45}$$

5.4 Variable Reduction

This section shows how the reduction of variables for the balancing and change sequencing problems can be calculated. The variable reduction is used to create sparse sets for the formulation eliminating variables that are unfeasible or could not produce optimal answers. The reduction is problem dependent, once characteristics of the given instance are used to calculate bounds and limits to conclude unfeasible or non-optimum assignments.

5.4.1 Variable Reduction for ALBP

In all presented models, balancing variables and restrictions are all based on the Feasible Task Assignment (FTA) tuple. Variables are only defined when they are part of the set FTA. The definition of the Simple Assembly Line Balancing supposes any task can be assigned to any workstation. However, it can be inferred that not all assignments could produce an optimal or feasible answer in terms of cycle time. A task with several dependent tasks, for instance, could not be assigned to the very end of the assembly line. All its successors would also need to be assigned to the last stations resulting in a very high cycle time for the end of the line.

For a given cycle time, one can use the precedence relations to exclude task assignments that are proven to be unfeasible. FTA is built based on the earliest and latest station a task can be allocated. For the binary version of SALBP, according to Patterson and Albracht (1975), the earliest station (E_i) for a task i can be calculated by adding the time of all tasks that precede i (P_i^*) divided by the cycle time. The ratio measures at least how many stations have to be filled in order to the predecessors of i are all as-

signed (Equation 5.46). Once stations' indexes are integer values, the ceiling function is used. Analogously, the latest station (L_i) is calculated based on the duration times of the successors (F_i^*), as showed in Equation 5.47. FTA is defined as all the task-station pairs in which the stations are within the interval between the earliest and latest stations, according to Equation 5.48. For SALBP-2 instances, an upper bound for the cycle time should be used.

$$E_i = \left\lceil \frac{TD_i + \sum_{j \in P_i^*} TD_j}{CT} \right\rceil \qquad \forall\ i = 1, ..., N \tag{5.46}$$

$$L_i = NS - 1 + \left\lceil \frac{TD_i + \sum_{j \in F_i^*} TD_j}{CT} \right\rceil \qquad \forall\ i = 1, ..., N \tag{5.47}$$

$$FTA = \big\{(i, j) : j = E_i, ..., L_i\big\} \qquad \forall\ i = 1, ..., N \tag{5.48}$$

Equations 5.49 and 5.50 are the integer adaptation of the reduction strategy proposed by Sikora, Lopes, Schibelbain and Magatão (2017). These expressions also consider the number of copies of each task (N_j). Note that the bounds are slightly softer for the integer formulation. E_i^*, in the integer formulation, means that at least one copy of task i can be allocated in such station, but not necessarily all the copies. This bound can be strengthened by also defining an upper bound of the number of copies that can be assigned to a station.

$$E_i^* = \left\lceil \frac{TD_i + \sum_{j \in P_i^*} TD_j \cdot N_j}{CT} \right\rceil \qquad \forall\ i = 1, ..., N \tag{5.49}$$

$$L_i^* = NS - 1 + \left\lceil \frac{TD_i + \sum_{j \in F_i^*} TD_j \cdot N_j}{CT} \right\rceil \qquad \forall\ i = 1, ..., N \tag{5.50}$$

The maximal amount of task's copies can be also determined by counting the duration time of predecessors and successors. The theoretical amount of copies of task i that can be allocated in or before (after) station j is defined as $M_{B_{ij}}$ ($M_{A_{ij}}$). These values are used to calculate the maximal number of copies that can be allocated in each station M_{ij}. In Equation 5.51, $M_{B_{ij}}$ is defined as the amount of copies that can be fit in a station after the processing time of all its predecessors are considered. Similarly, Equation 5.52 defines $M_{A_{ij}}$ based on the follower tasks.

The upper bound for the number of copies of task i in station j (M_{ij}) is defined in the interval $[E_i^*; L_i^*]$. This bound is determined by the minimal value between the

number of available copies of the task (N_i), the amount of tasks an empty station can support $\left\lfloor \frac{CT}{TD_i} \right\rfloor$ and the amount of tasks that fit in the station when one also considers the predecessors ($M_{B_{ij}}$) and successors ($M_{A_{ij}}$), as seen in Equation 5.53.

$$M_{B_{ij}} = \left\lfloor \frac{j \cdot CT - \sum_{P_i^*} N_k \cdot TD_k}{TD_i} \right\rfloor \tag{5.51}$$

$$M_{A_{ij}} = \left\lfloor \frac{(NS + 1 - j) \cdot CT - \sum_{F_i^*} N_k \cdot TD_k}{TD_i} \right\rfloor \tag{5.52}$$

$$M_{ij} = \min \left(n_i, \left\lfloor \frac{CT}{TD_i} \right\rfloor, M_{B_{ij}}, M_{A_{ij}} \right) \tag{5.53}$$

5.4.2 Variable Reduction for Change Sequencing

In the proposed models, no variable reduction in terms of transition variables is used. However, there are forms of inferring that some of the changes are unfeasible in certain periods.

As an example, we use the precedence diagram of Figure 5.5 with the precedence relations presented in the arcs and the cost of moving each task to a new station at the top right side. A reference as initial or last configuration is needed to observe the feasible and unfeasible assignments.

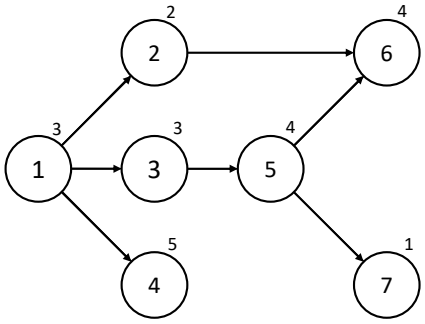

Figure 5.5: Precedence graph with task number inside the circles and moving costs on the right side.

The variable elimination can be made either from the initial solution onwards or from the final configuration backwards. For the given example, Figure 5.6 is used as a reference assignment. The proposed reduction procedure consists in summing the costs of necessary changes for the reassigning of tasks. Task 3, in Figure 5.6 is assigned to station

2. If we were to move Task 3 to Station 3, the implementation cost would be 3 (the cost of moving task 3) plus 4 (the cost of moving Task 5). Once Task 5 succeeds Task 3, both of them must be moved in order to assign Task 3 at Station 3. A similar example occurs with Tasks 2 and 6. In order to move Task 6 to any other station, Task 2 should also be moved.

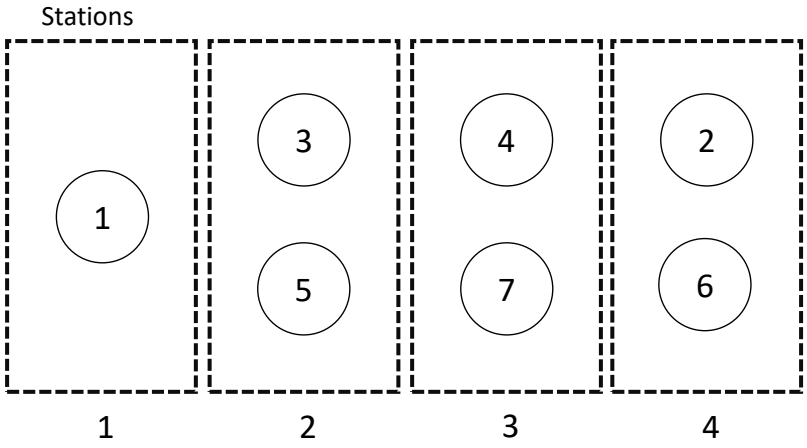

Figure 5.6: Example of an initial assignment.

The left side of Table 5.8 contains the reassigning costs of every task to every workstation. The moving cost consists in the individual task cost plus the cost of every precedent or dependent task that have to be moved along. If a task is already assigned to a station, the reassigning cost to the same station would be zero. For instance, if we take Task 1 on Station 1, its reassignment cost is zero, once Task 1 is already assigned to the first station (see Figure 5.6).

Suppose that the maximal effort performed in each implementation phase is 5 (the maximal amount of effort per period could be given by the problem instance or a maximal bound should be estimated when the effort is unknown). The left side of Table 5.8, containing the reassigning costs, can be used to eliminate unfeasible assignments. The underlined values contain assignments that would require more effort than the available for the first implementation period. Therefore, the assignment variables x_{131}, x_{141}, x_{331}, x_{341}, x_{511}, x_{611}, x_{621}, x_{631} and x_{711} can be eliminated from the formulation (variables are represented as x_{tsp}). Note that the sixth task cannot be assigned anywhere else, once the removal of Task 2 is necessary to move Task 6.

The same procedure can be used to remove variables from further transition periods.

Table 5.8: Example of the feasible changes and the effect of the reduction of variables due to the sequencing.

1 transition: Cost = 5								2 transitions: Cost = 10							
Station - Task	1	2	3	4	5	6	7	Station - Task	1	2	3	4	5	6	7
1	0	2	3	5	7	13	8	1	0	2	3	5	7	13	8
2	3	2	0	5	0	6	1	2	3	2	0	5	0	6	1
3	10	2	7	0	4	6	0	3	**10**	2	7	0	4	6	0
4	16	0	8	5	5	0	1	4	16	0	8	5	5	0	1

The right side of Table 5.8 shows the possible reassigning options after 2 transitions. For the second period, movements up to 10 units of effort can be performed. The variable reduction is weakened in every further period. For 2 transitions, only 2 assignments have costs greater than 10: x_{132} and x_{612} can be eliminated. A further unfeasible variable is x_{132}. Although the total moving cost of 10 units of effort could be a feasible reassignment, the division in two periods is not possible. Moving Task 1 to Station 3 requires reassigning Tasks 1, 3 and 5 with moving cost of 3, 3 and 4. Although the sum of the costs is 10, there is no way of performing the changes in 2 transitions with the maximal cost of 5. Therefore x_{132} can also be eliminated.

This procedure can be used in the forward and backward directions. If any variable is eliminated in a direction, the formulation can be written without the variable. The variable reduction is sensitive to the allowed implementation effort per period and the number of periods. High values of implementation effort eliminates only few variables. Furthermore, once the reduction weakens during further periods, the variable elimination occurs mainly in the very firsts and very lasts periods.

The variable reduction for the transitions here described is weaker than the reduction based on SALBP characteristics. Although variables could be eliminated, preliminary tests showed that the reduction variable based on transitions have not decisively contributed to the computational reduction, at least to the optimization software used. Therefore, for the results of Chapter 6, only the reduction based on SALBP is used.

Chapter 6

Results and Discussions

This chapter presents the results of the Assembly Line Balancing Problem from Chapter 5. A dataset is developed and the effect of every forming parameter is investigated.

6.1 Didactic/Motivational Examples

In this section, we propose two fictional case studies that exemplify the use of the model in sequencing the implementation of a new balancing. The cases represent examples of imaginable structural changes that need several allocation changes to reach the new desired configuration. Both cases are based on the precedence diagram presented by Buxey (1974) (Figure 6.2 without Task 30). With 29 tasks, the problem presents enough elements to allow model insights. At the same time, the number of tasks is still adequate to represent the allocations in simple figures. These cases are also presented in Sikora, Lopes and Magatão (2016).

6.1.1 Augmenting the number of stations

For this case study we suppose a fictional case based on a real industry situation. In this first situation, a company is producing pieces according to the precedence diagram of Buxey (1974) using 10 workstations. The optimal answer of the balancing problem with 10 stations is equal to 34 time units. Due to an increasing demand of the market, the

production sector of the company intends to add a workstation to reduce the cycle time. The engineers solved the new problem and obtained the optimal answer for the cycle time of 32 time units, enough to attend the new demand.

When engineers compared the initial implementation with the new goal, they found that 22 allocations have changed, out of 29 total possible allocations. They estimated that reassigning a task would require 2 hours of work, consisting in moving and installing equipment and updating process files. The line runs intermittently during weekdays, so that the changes have to be performed at weekends.

The manager calculates that a single pass implementation would require 44 hours of overtime. This change requires the assignment of a team of 6 employees working on Saturday or 3 employees working the whole weekend. Since this is not the only project being scheduled for the month, the manager evaluates whether the change can be implemented in periods with less employees being needed in each intervention. After each pass, however, the line must be able to operate at the initial cycle time of 34. That is, the intermediary changes must not negatively effect the output of the line.

The results using the model of Section 5.2.1 are displayed in Table 6.1 for several values of transitions. Each column in the table represents a solved instance. The cost of moving each task (c_t) is equal to 2 hours of work. As a result, the value of the variable $Effort$ represents the number of hours necessary to implement the changes for the most loaded transition. This value can be used to determinate the size of the team needed for the project. The results of Table 6.1 are obtained in less than 1 minute for $NP \leq 10$ and up to 10 minutes for more periods in a personal computer (2.3 GHz, Intel i7-3610 QM, 16.0 GB of RAM).

Table 6.1: Results for the model for the first case study. The value of $Effort$ is diluted between stages of implementation. Stages are the intermediary configurations of the line.

Stages	0	1	2	3	4	5	6	7	8	9	10	11	...	21	..	24
Transitions	1	2	3	4	5	6	7	8	9	10	11	12	...	22	..	25
Effort (hours)	44	22	16	12	10	8	8	6	6	6	6	4	...	4	...	2

According to the results of Table 6.1, the size of the employed team can significantly reduce by allowing intermediary stages. With one intermediary period, the amount of

hours is reduced to half: 22 hours of work in 2 implementations. Although the effort needed in the implementation reduces with a greater number of stages, some solutions are clearly dominated. Dividing the implementation in 6 stages, for instance, has at least one transition that needs a team for 8 hours of work. A similar result can be obtained using only 5 stages. Note that to reduce the effort to only 2 hours (one change) per transition, 24 periods are necessary. This means that when only one reallocation at a time is allowed, it is necessary to perform extra reassignments: some tasks will be moved more than once between stations (25 changes for the 22 allocation differences).

The model does not only calculates the amount of effort needed for the implementation, but also finds every intermediary configuration that must be applied. Figure 6.1 exemplifies one possible planning option for 1 intermediary stage, where each task is represented by a different color. Note that in the initial condition, the station 11 is empty. At Stage 1, some tasks are relocated, while the cycle time of the line remains unchanged (34 Time Units). For the final condition, an assignment with 32 Time Units for the cycle time is reached.

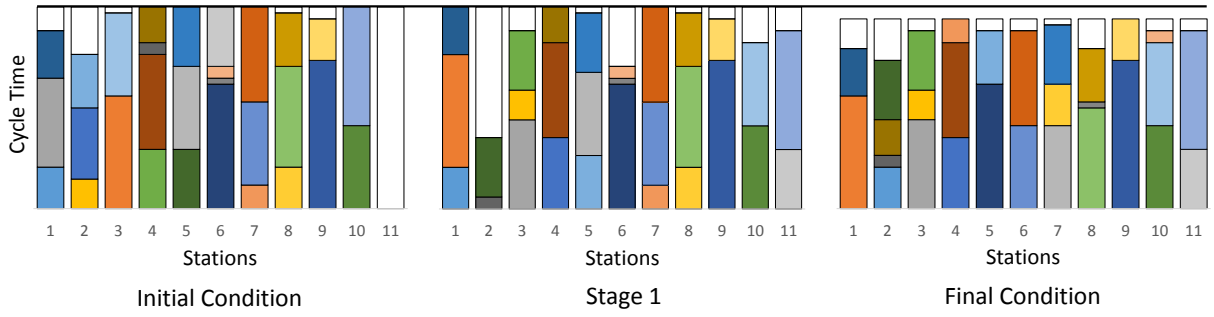

Figure 6.1: Implementation planning with one intermediary period. Each task is represented by a block with a size proportional to its duration time. The assignments are changed so that the line's configuration go from the initial to final condition with two transitions. Note that the cycle time reduces from 34 to 32 Time Units.

The number of variables of the model is proportional to the size of the set FTA and the number of periods (NP) considered. For each period, the binary variables x_{tsp} are defined. At the same time, each transition uses one variable y_{tsc} for each element of FTA. At a given instance, the number of periods is defined as NP, resulting in $NP + 1$ transitions. Finally, a single variable ($Effort$) is used in the objective function. For this case study, the number of possible allocations (FTA) is equal to 185. This way, we

can calculate the number of variables used by: $|FTA| \cdot (NP + NP + 1) + 1 \therefore 185 \cdot (2 \cdot NP + 1) + 1$.

6.1.2 Adding a new task

For the second case study, a second instance based on a real industry is presented. We suppose that after the line was adapted to support 11 workstations, the marketing department recognized a new trend in the market and required a product modification. The project department developed the extra feature, which would require a new task to be performed. The production engineers concluded that the new task (number 30) would require 18 time units to be processed, resulting in the precedence diagram showed in Figure 6.2.

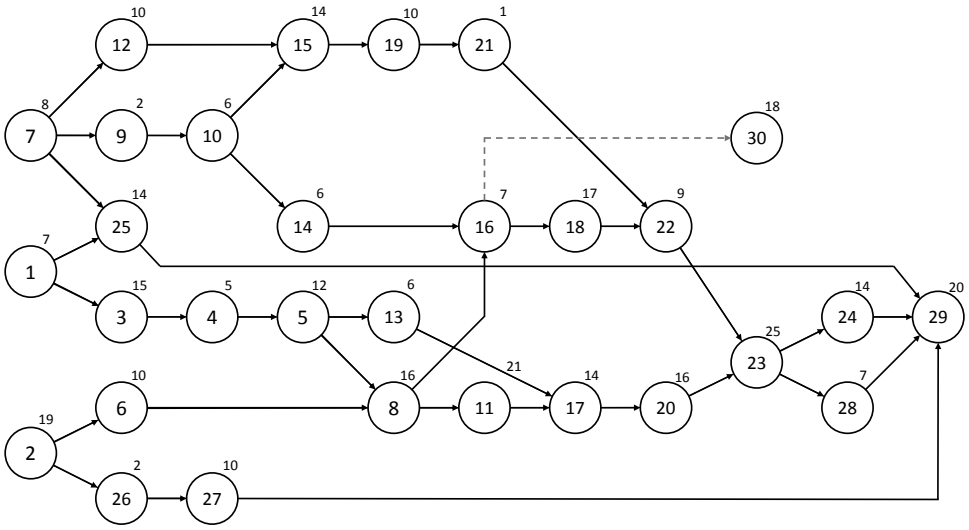

Figure 6.2: Adapted precedence diagram from Buxey (1974), which originally contains 29 tasks. The values inside circles show the number of the task while the duration time of each task is represented at above each task. Task 30 (18 time units) is added to the problem for this case study along with the precedence relation between tasks 16 and 30.

The final condition showed in Figure 6.1 (or initial condition in Figure 6.3) contains several workstations with idle time (namely stations 1, 2, 3, 5, 6, 7, 8, 10 and 11). Individually, no station has an idle time of 18 time units, but the sum of the idle time for the line is 28. By rearranging tasks, it is possible to add Task 30 in the line without impacting the cycle time and the number of stations. The production engineers chose

a new balancing that, however, required 18 task changes or 36 hours of overtime. Once again, the manager wondered if the implementation could be done in smaller steps.

Table 6.2 shows the results for an interval of the number of stages. With one intermediary stage, it is possible to divide the 18 changes in exactly 2 implementations of 9 changes (18 hours of overtime). This second case proved to be more difficult to find solutions with only one change at a time. Even if we allow 36 transitions, two for each difference in initial and final allocations, we still need at least one transition with 2 changes. The instances with less than 10 intermediary periods were solved with less than 1 minute, while up to 10 minutes were required for the larger instance.

Table 6.2: Results for the model for the second case study. The value of $Effort$ is diluted between stages of implementation.

Stages	0	1	2	3	4	5	6	7	8	9	10	...	35
Transitions	1	2	3	4	5	6	7	8	9	10	11	...	36
Effort (hours)	36	18	14	10	8	8	6	6	6	6	4	...	4

Figure 6.3 shows a diagram with the planning solution for ten intermediary stages. In this figure, the initial condition, the last intermediary stage and the final assignment are represented. Note that the new task (colored in black) is only added in the final condition. That is, the occurrence and precedence restrictions for Task 30 are dismissed for the intermediary stages. The first transitions are used to create space for the new task, whose assignment happens in the last transition.

With these results, the manager has more options to schedule and implement projects in assembly lines. The model's result can be interpreted as the size of the team needed to implement the changes, which varies according to the number of stages allowed. These simple examples are used to clarify the model utilization and the obtainable results applied to Assembly Line Balancing Problems.

6.2 An Implementation Sequencing Dataset

In order to verify the models performance and the effect of different problem's parameters, a new dataset for the Assembly Line Implementation Problem was developed.

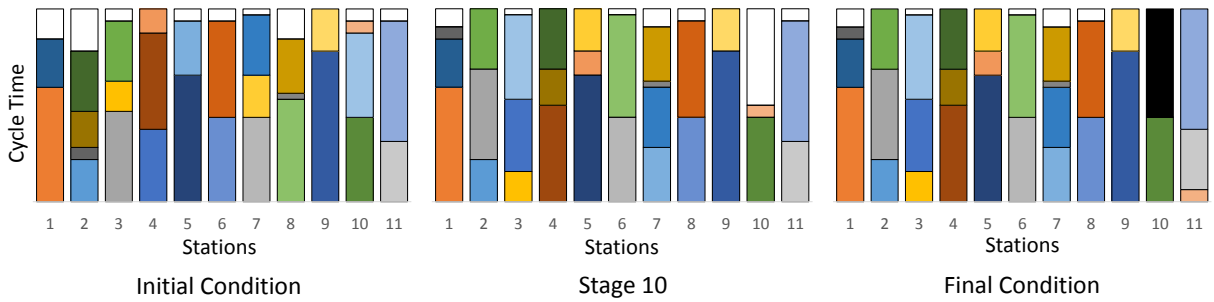

Figure 6.3: Implementation planning with ten intermediary periods. The initial and final conditions as well as the last intermediary stage are showed. Each task is represented by a block with a size proportional to its duration time. Task 30 is represented only in the final condition, in black, at station 10. The cycle time of all configurations is 32 Time Units.

The instances are based on the dataset from Otto et al. (2013), which is described in Section 2.8.

Otto et al. (2013) constructed a dataset containing SALBP instances of four sizes: small (20 tasks), medium (50 tasks), large (100 tasks), and very large (1000 tasks). Seven structures for the precedence graph were used for the instances: the precedence graph could either present chains of tasks (CH), bottleneck tasks (BN), or both, mixed (MX). These structures are observed in real assembly lines and, therefore, are used to inspire the definition of the dataset. The authors also varied the order strength (OS) of the graphs, resulting in 7 variations for the precedence graph: graph with chains of tasks with OS of 0.2 and 0.6 (CH-0.2; CH-0.6); graph with task bottlenecks with OS of 0.2 and 0.6 (BN-0.2; BN-0.6); graphs with both chains and bottlenecks with OS of 0.2, 0.6, and 0.9 (MX-0.2; MX-0.6; MX-0.9). As a third parameter, the tasks' processing time distribution are classified in three categories: peak at the bottom (PB), peak in the middle (PM), and bimodal (BI). The peak at the bottom contains tasks with small processing times comparing to the line's cycle time. The tasks' processing time distribution on peak in the middle instances is centered on half of the cycle time. The bimodal distribution is a mixture of both distributions. The instances are built to represent SALBP-1 instances. A standard value of 1000 is set for the cycle time limitation for all instances. Finally, Otto et al. created 25 variations for every combination of number of tasks, precedence diagram, and time distribution resulting in 2.100 instances.

Based on the same characteristics for the SALBP, but also integrating other important parameters of sequencing, we propose an adaptation and extension of the dataset to the Assembly Line Implementation Problem (ALIP). The proposed dataset contains 378 instances, formulated accordingly to permit the analysis of problem's parameters. The dataset is divided in two subsets. The first subset contains the small and medium problems (with 20 and 50 tasks), while the second set contains the large instances (100 tasks).

Besides the number of tasks, precedence diagram, and time distribution, the number of periods (NP), and the limit for the cycle time restriction (CT) were also integrated in the dataset formulation. The instances are created for an interval of 1 to 3 intermediary stages and two levels of cycle time limitation. For every combination of the factors, one instance is defined. The parameters used to create the dataset are summarized in Table 6.3.

Table 6.3: Parameters used to create the Assembly Line Implementation Problem Dataset.

Parameter	Values
Number of Periods (NP)	1 - 2 - 3
Number of Tasks (NT)	20 - 50 - 100
Cycle Time Limit (CT)	1000 - 1100
Time Distribution (TB)	PB - PM - BI
Precedence Diagram Types (PD)	BN-(0.2, 0.6) - CH-(0.2, 0.6) - MX–(0.2, 0.6, 0.9)

An ALIP instance contains both the ALBP information (tasks' time, precedence diagram) as well as the implementation requiring information, as exemplified in Figure 6.4. Once the implementation transforms a system from an initial configuration to an optimal one, both initial and final assignments are necessary to sequence the implementation. Therefore, the solution procedure SALOME (Klein and Scholl, 1996) was used to obtain the final optimal answers (for the procedure description, see Section 2.5). The optimal solution for the SALBP-2 instances are used as the goal configuration for the sequencing model.

In order to obtain an initial configuration, task assignments were obtained using an heuristic procedure. Based on the heuristic methods presented in Section 2.5.1, solutions can be achieved by assigning tasks to station in a specific order. For this dataset, we used a station-oriented solution scheme with the task time as the assignment priority (Scholl

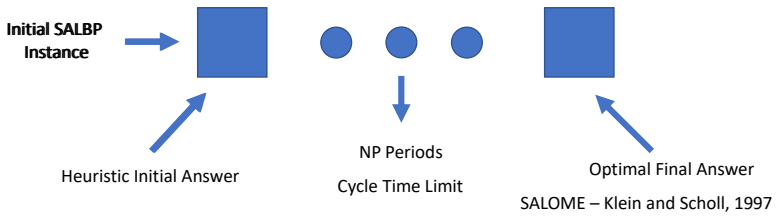

Figure 6.4: Elements of the proposed dataset.

and Klein, 1999). In other words, for one station at a time, all tasks that can be assigned to the station are considered. The task with the highest processing time that does not extrapolate the cycle time limit and obeys the precedence relations is assigned to the station. When no more tasks can be assigned, the station is closed and the procedure continues with the following station until all tasks are assigned.

The heuristic procedure for the initial answer was chosen for its simplicity and reproducibility. If the same rules are applied, the resulting initial configuration would be the same in every run. Furthermore, the processing time priority rule alone can provide reasonable answers but, at the same time, has a low probability of reaching optimal assignments. Once the ALIP consists in finding an implementation path from the initial to the optimal final answer, an optimal initial answer would result in an irrelevant implementation problem.

Using the processing time priority rule heuristic, the initial answers for the instances were calculated both with the cycle time limit of 1000 and 1100 time units. The interval of time limits provides significantly different initial answers. The most restricted cycle time produces less loaded stations, but may require more stations than the optimal assignment. The limit of 1100 time units, on the other hand, may use less stations, but the assignments of tasks within the stations tend to present more differences to the optimal answers. Furthermore, the cycle time limit used for the initial answer is also the limit used for any intermediary instance (Equation 5.3).

Finally, for the 378 instances created in the dataset, the cost of changing the assignment of any task is set to 1. That means that the cost of moving any task is assumed to be the same.

6.3 Dataset Results

In this section the dataset results as well as the effects of every parameter used to define the instances are described. The tests were solved on a 2.3 GHz, Intel i7-3610 QM, computer with 16.0 GB of RAM using the universal solver IBM ILOG CPLEX Optimization Studio 12.6.2. For all tests, the precedence relation from Patterson and Albracht (1975) (Expression 5.5) was used.

Firstly the results for the small and medium instances are described. Once all 252 small and medium instances were solved to optimality, a more complex analysis of the problem can be performed. The results for the 126 large instances are described in Section 6.3.6.

The average results for the solution of the first dataset are shown in Table 6.4. All the 252 instances (#Cases) were solved to optimality ($\#Opt$) in an average solution time of 22.0 seconds (Sol. Time). The table also shows the average number of differences between the initial heuristic solution and the final optimal solution for SALBP-2 (Diff.). The column Perf. Changes contains the average number of changes that must be performed to achieve the final configuration with either 1, 2, or 3 intermediary stages. The number of performed changes is calculated by multiplying the number of periods (NP) with the number of changes per period. The average difference between the performed changes and the differences is shown in the column Extra Changes (26.28 - 24.52 = 1.76). Note that some divisions may inevitably require more changes. Sequencing the changes of 22 tasks in 3 transitions, for instance, would require at least one period with 8 changes. If we multiply the 8 necessary changes by the 3 transitions, we obtain 24 as the number of performed changes. Once the model for ALIP-2 minimizes the crew required to perform the changes in all transitions, some transitions may require less changes than the most loaded one.

Table 6.4: Summary of the average results from the ALIP dataset.

#Cases	#Opt	Sol.Time (s)	Diff.	Perf. Changes	Extra Changes
252	252	22.0	24.52	26.28	1.76

Note that the dataset showed that it is possible to divide the implementation in phases by only increasing the total amount of necessary changes by 1.76 on average.

Although more changes must be performed in a step-wise implementation, in every phase only a subset of the changes must be implemented. The 1.76 extra changes represent only 7.1% of the number of differences, supporting the use of the sequencing to the implementation of projects.

In the following subsections, the effect of every parameter used to create the dataset is further investigated. The nomenclature used in Table 6.4 is the same in all the result tables.

6.3.1 Effect of the Number of Periods

The dataset was formulated with instances containing 1, 2, and 3 intermediary periods for the implementation. In other words, the instances were constructed to allow 2, 3, and 4 implementation phases. Table 6.5 contains the results of the solved dataset separated by the number of intermediary periods (NP).

Table 6.5: Summary of the average results from the ALIP dataset per Number of Periods.

NP	#Cases	#Opt	Sol.Time (s)	Diff.	Perf. Changes	Extra Changes
1	84	84	0.5	24.52	25.50	0.98
2	84	84	7.5	24.52	26.35	1.83
3	84	84	58.0	24.52	27.00	2.48

The most significant effect of the number of periods is on the solution time. From the interval from 1 to 3 periods, the necessary computation effort increases from an average of 0.5 seconds to 58.0 seconds. This increase is justified by the complexity of multi-phase problems. The number of variables and restrictions of the given formulation depends on the number of periods and transitions. A higher number of phases results in bigger models that tend to be harder to solve. Furthermore, the decision of multi-phased models can be significantly more difficult: the model should not only calculate which changes must be performed, but also when, while all period feasibility constraints must be obeyed. Nonetheless, the computational effort may not increase for any number of periods. Although the tested instances only contain from 1 to 3 intermediary periods, some instances with several implementation phases may be easy to solve. Due to the division of changes into phases, highly divided implementations contain very low flexibility for each transition. Therefore, there are two competing effects on the problem size and

its solution difficulty. For the tested interval (1 to 3), however, a clear increase on the solution time was observed.

Although the number of differences from the initial to the final answer is the same for all values of NP, the average number of performed changes differs. The instances with more intermediary periods tend to require more changes to be performed. By dividing the implementation in 2 (NP = 1), only 0.98 extra changes were necessary. The division by 3 and 4 (NP = 2 and 3) required 1.83 and 2.48 extra changes on average, respectively.

6.3.2 Effect of the Number of Tasks

The instances used to create the first part of dataset were based on the small and medium cases from Otto et al. (2013). The small instances contain 20 tasks, while the medium instances were created with 50 tasks. Table 6.6 contains the result data for each value of tasks used.

Table 6.6: Summary of the average results from the ALIP dataset per Number of Tasks.

NT	#Cases	#Opt	Sol.Time (s)	Diff.	Perf. Changes	Extra Changes
20	126	126	0.5	12.07	13.65	1.58
50	126	126	43.5	36.98	38.92	1.94

The number of tasks presented huge effects on the measured parameters. The solution time for the small instances has an average of 0.5 seconds, while the cases with 50 tasks required on average 43.5 seconds. All the initial answers were obtained with the same heuristic described in Section 6.2. The number of different assignments from the initial answer to the optimal answer for the small instances is 12.07 changes on average. That means 60.1% of the tasks must be reassigned during the implementation. The medium instances required on average 36.98 changes or 74.0%. The bigger instances also required more performed changes: 13.65 task reassignments were performed in the small cases and 38.92 in the medium ones. Finally, the number of extra changes was higher for the medium instances, although the difference is less evident: 1.58 to 1.94.

In summary the quantity of tasks greatly influences the difficulty of the problem solution as well as the implementation itself. An assembly line with more tasks may present more changes, resulting in more workload to implement the optimal configuration.

Furthermore, the number of variables and restriction in the model is directly related to the number of tasks, suggesting that problems with more tasks may be more difficult to solve.

6.3.3 Effect of the Cycle Time Limit

The effect on the cycle time limit has influence in two aspects of the problem. Firstly, a limit for the cycle time is used for the heuristic to obtain an initial answer (as seen in 6.2). Furthermore, the cycle time limitation is imposed on every intermediary stage during the implementation scheduling (Equation 5.3, Section 5.2.1). The dataset of Otto et al. (2013) was formulated to have 1000 time units as the cycle time. Therefore, to test the effect of different limits, instances were created with the same limit of 1000 for tight implementations and 1100 time units for more relaxed cases. Table 6.7 summarizes the effect of the cycle time limit on the obtained solutions.

Table 6.7: Summary of the average results from the ALIP dataset per Cycle Time Limit.

CT	#Cases	#Opt	Sol.Time (s)	Diff.	Perf. Changes	Extra Changes
1000	126	126	42.1	23.74	26.01	2.27
1100	126	126	1.9	25.31	26.56	1.25

According to Table 6.7, a tighter bound for the cycle time produces more difficult problems. The average solution time of the instances with CT = 1000 is 42.1 seconds while with a more relaxed bound the average solution time is just 1.9 seconds. The number of extra changes is also significantly higher for the instances with CT = 1000 time units. Once the bound for the cycle time is lower, fewer task assignments are possible. Therefore, more changes are necessary to reach the final configuration (2.27 against 1.25).

The number of differences and the number of performed changes are higher for the instance with CT = 1100. This difference, however, can be credited to the heuristic used for the initial answer. The solutions produced with the limit of 1100 for the cycle time allowed stations to have more tasks than the optimal answer. In general, the initial solutions using this heuristic produced instances that required more changes.

6.3.4 Effect of the Time Distribution

The time distribution of the tasks' processing time also proved to be influent on the difficulty of implementing a new solution. The results for the three time distributions (**P**eak at the **B**ottom, **P**eak in the **M**iddle and **BI**modal) are showed in Table 6.8.

Table 6.8: Summary of the average results from the ALIP dataset per Processing Time Distribution.

TD	#Cases	#Opt	Sol.Time (s)	Diff.	Perf. Changes	Extra Changes
PB	84	84	0.5	19.86	20.91	1.05
PM	84	84	49.9	30.50	32.96	2.46
BI	84	84	15.6	23.21	24.98	1.77

The Peak at the Bottom distribution contains tasks with low processing time. These instances result in problems with a small number of stations once less stations are needed to fit all tasks. With less assignments possibilities, this class of problem resulted in easier instances when compared to the others. The instances for this distribution required only 0.5 seconds on average, and presented the smallest amount of differences from the initial to final answer (19.86), performed changes (20.91), and extra changes (1.05).

The Peak in the Middle distribution produced the opposite effects. With tasks' time distributed centered on 500 time units, more workstations are needed to fit tasks. These instances, therefore, present more variables than the other distributions once the variables must be defined for each station. Furthermore, reassigning a high processing time task ("large task") can be more difficult than a small task. A large task would not normally fit in the idle time of other stations. Hence, reassigning a large task may require the reassignment of multiple tasks. By the results of Table 6.8, the Peak in the Middle instances required more time to be solved (49.9 seconds), and they presented more differences (30.50), performed changes (32.96), and extra changes (2.46).

Finally, the bimodal distribution presents results between the two extremes. These instances have both small and large tasks, proving themselves to be harder than the Peak at the Bottom instances, but easier than the Peak in the Middle ones.

6.3.5 Effect of the Precedence Diagram

For the last parameter, Table 6.9 summarizes the results separated by the precedence diagram. Instances represent diagrams containing Bottlenecks (BN), Chains (CH), or both (MX). The bottlenecks are tasks with several predecessors and successors. They are key operations that divide the precedence diagram in parts. Chains are also common in real assembly lines: they are a sequence of tasks that have only one predecessor and one successor. The chains appear when one set of tasks must be performed in a specific order. Finally, the MX instances contained a mix of both bottlenecks and chains.

Table 6.9: Summary of the results for the ALIP dataset per Precedence Diagram.

TD	OS	#Cases	#Opt	Sol.Time (s)	Diff.	Perf. Changes	Extra Changes
BN	0.2	36	36	20.8	29.08	31.30	2.22
BN	0.6	36	36	2.3	24.17	25.48	1.31
CH	0.2	36	36	26.5	26.67	28.78	2.11
CH	0.6	36	36	59.9	23.83	25.86	2.03
MX	0.2	36	36	40.0	26.25	27.75	1.50
MX	0.6	36	36	4.0	25.75	27.39	1.64
MX	0.9	36	36	0.6	15.92	17.45	1.53

Table 6.9 shows that the precedence diagram can influence the solution time and the number of necessary changes. However, for the tested instances, no clear rule can be inferred. All precedence diagrams had at least one value of the order strength (OS) that produced time consuming instances.

From the data tested, no conclusion can be drawn based on the order strength of the graphs. Although a higher OS decreased the solving time of the bottleneck and mixed instances, the opposite behavior occurred for the chain instances. The total number of differences and performed changes seem to decrease with higher levels of OS, although nothing can be inferred from the extra changes.

In summary, the structure of the precedence graph seems to have a role on the difficulty of the ALIP instance. However, the tested dataset proved to be insufficient to map the effects of every type of graph.

6.3.6 Model's performance on large instances

This section presents the results for the second part of the proposed dataset. The 126 large instances were solved with a time limit of 3,600 seconds. Table 6.10 contains the results of the large instances. The last row contains the average results for all instances. The other rows presents the average results for the instances filtered by parameter. For instance, the parameter NP separates the results of the instances with 1, 2, and 3 intermediary periods. The columns contain the number of cases, the number of instances solved to optimality and the number of instances without an incumbent solution within the given time. For the instances with an incumbent solution, but without a proven solution, the average gap are also reported. The results are shown filtered by each of the dataset parameter, along with the total number of instances and the average solution time. Once not all instances were solved to optimality, no comparisons are made to the number of necessary changes. Furthermore, the solver was unable to find feasible answers for 7 of the proposed instances.

Table 6.10: Performance of the model for the large instances filtered by the dataset parameters. The row Average GAP contains the average gap for the instances that were not solved, but presented a feasible answer. The last row contains the sum of the number of solved cases and the average solution time.

Parameter	Value	#Cases	#Opt	#Fail	Average GAP	Sol.Time (s)
NP	1	42	42	0	-	11.6
	2	42	35	3	6.1%	821.3
	3	42	27	4	10.9%	1403.9
CT	1000	63	45	7	11.1%	1170.7
	1100	63	59	0	5.7%	320.5
TD	PB	42	40	0	9.3%	271.6
	PM	42	27	7	7.6%	1383.7
	BI	42	37	0	12.9%	581.6
PD	BN-0.2	18	13	0	12.4%	1118.4
	BN-0.6	18	16	0	9.4%	431.1
	CH-0.2	18	13	2	5.8%	1230.1
	CH-0.6	18	16	2	-	430.9
	MX-0.2	18	13	2	8.7%	1276.2
	MX-0.6	18	15	1	10.1%	705.0
	MX-0.9	18	18	0	-	27.8
Total	all	126	104	7	9.6%	745.2

According to Table 6.10, the number of periods has impacts on the solution time and the number of solved instances. Just as it can be seen for the small and medium

instances in Section 6.3.1, an increase in the number of intermediary periods increases the difficulty of solving instances. Not only the average solution time increases, but also do the number of unsolved instances and the number of failed instances.

The effect of the cycle time limitation also agrees from the one showed in the small and medium instances (Section 6.3.3). A tighter bound results in more difficult instances. Comparing the bound of 1100 and 1000 time units as limits, the average solution time increased almost four times for the more restricted problem (320.5 to 1170.7 seconds). Accordingly, the number of unsolved and failed instances also increases.

The observed time distribution effect also agreed between the two tested datasets. For the small and medium instances, Section 6.3.4, the Peak at the Bottom instances are the easiest to solve, the Peak in the Middle instances showed themselves the most hard instances, and the Bimodal instances presented an intermediary behavior. This pattern repeats for the large instances: the average solution time of the Peak at the Bottom instances is the lowest (271.6 seconds) comparing to the intermediary Bimodal (581.6 seconds) and the highest Peak in the Middle (1383.7 seconds). The number of unsolved and failed instances also suggests that the Peak in the Middle instances are the most difficult time distribution to solve.

From the different precedence diagrams used in the dataset, from Table 6.10 it seems that instances with smaller order strength (OS) may be harder to solve. For all precedence diagram patterns (Bottleneck, Chain, and Mixed), instances with lower precedence relations required more time to be solved and more of them resulted unsolved or failed.

6.3.7 Importance of the Parameters Summary

The Assembly Line Implementation Problem is based on a combination of problems. Several parameters influence directly the definition of each instance. For the proposed dataset, five control parameters were chosen to create a varied range of problems. Table 6.11 contains a summary of the observed effects for the proposed dataset.

The tests showed that for small values of the number of periods, the solution time and the number of changes increase. This may be due to both the increase of the number

Table 6.11: Summary of the effect on the increase of each parameter on the solution time and number of differences and changes.

Parameter	Sol.Time (s)	Diff.	Perf. Changes	Extra Changes
↑ NP	↑	=	↑	↑
↑ Tasks	↑	↑	↑	-
↑ CT	↓	-	-	↓
↑ Duration of Tasks	↑	-	-	↑
↑ OS	?	?	?	?

of variables and restrictions, but also on the extra number of intermediary stages that must obey the feasibility restraints. The number of tasks also presented an important role on the solution effort. More tasks result in a more difficult solution problem also requiring more changes in the implementation. For the extra changes, however, no strong conclusion can be drawn.

An increase on the cycle time limit has an opposed effect on the solution difficulty. The looser the limit for the cycle time is, the lesser is the solution time and the number of necessary extra changes. Values for the number of differences and performed changes are omitted due to the effect of the heuristic for the initial solution. Using another heuristic may produce other insights on these fields.

From the time distributions, the average duration of tasks could be observed as an important factor for the solution difficult. The higher the average processing time of the tasks, the more difficult it is to plan the implementation. The Peak in the Middle instances, that contains the largest instances, are the most difficult from the dataset. The presented differences on the number of differences and the total number of changes depends on the heuristic used rather than the time distribution itself. Differently, some trends on the effect of the order strength (OS) can be observed, but they are not consistent through both parts of the dataset.

The behavior showed in Table 6.11 should be seen as trends observed in the tested data for a small interval of the parameters' value. The reader is advised that to extend or generalize the obtained patterns more tests should be performed.

6.4 Formulation Comparison

Chapter 5 presented 4 models for the Assembly Line Implementation Problem and variations. In this section, comparisons between the methods using the dataset described in Section 6.2 are presented. For these comparisons, only the first part of the dataset is tested (small and medium instances).

6.4.1 State Variables vs. only Transition Variables

The first model comparison is between the models from Sections 5.2.1 (State-Model) and 5.2.2 (Transition-Model). These sections present the model for the Implementation Sequencing for the Simple Assembly Line Balancing Problem. Section 5.2.1 contains the standard formulation as defined in Chapter 3: the implementation is divided in intermediary stages and transitions, with different variables used to treat each of them. The model of Section 5.2.2, on the other hand, has only transition variables to describe both the stages and the transitions. The states are defined as a sum of the initial condition to all changes occurred until the final state.

Table 6.12: Comparison of the models using state variables and using only transition variables.

Model	#Cases	#Opt Found	#Opt Proven	Avg. Sol.Time (s)
State-Model	252	252	252	22.0
Transition-Model	252	242	240	214.8

Table 6.12 contains the summary of the model comparison. All the 252 cases from the dataset of Section 6.2 were tested using both formulations. Within the time limit of 3,600 seconds, only the State-Model was able to find and prove all the answers. The column #Opt Found stands for the number of optimal solutions found: the Transition-Model failed to find 10 optimal answers in the available time. While the State-Model proved all 252 optimal answers (Column #Opt Proven), the Transition-Model solved to optimality 240 of the instances.

The non-solved instances from the Transition-Model resulted in a higher average solution time. The State-Model required only 22.0 seconds on average to find and prove all the 252 optimal solutions. On the other hand, the Transition-Model spent an average

of 214.8 seconds.

Table 6.13: Comparison of the models using state variables and using only transition variables by specific parameters.

			State-Model		Transition-Model	
Parameter	Value	#Cases	#Opt	Sol.Time	#Opt	Sol.Time
NP	1	84	84	0.5	83	2.9
	2	84	84	7.5	81	203.8
	3	84	84	58.0	74	437.7
Tasks	20	126	126	0.5	124	11.7
	50	126	126	43.5	114	417.9
CT	1000	126	126	42.1	114	342.9
	1100	126	126	1.9	124	86.7
TD	PB	84	84	0.5	84	0.6
	PM	84	84	49.9	75	421.9
	BI	84	84	15.6	79	221.9

Table 6.13 shows the results discretized per the dataset's construction parameters. It can be noted that the behaviors summarized in Section 6.3.7 appear equally on both models. For the given dataset, the State-Model showed itself to be superior than the Transition-Model for every parameter. The State-Model solved more instances in a lesser solution time for the average of any instance group. The Peak at the Bottom problems, however, showed similar results. All 84 instances were solved by each model in an average of less than a second.

6.4.2 Binary vs. Integer Formulation

This section compares the binary and standard model of Section 5.2.1 to the model based on integer decision variables described in Section 5.2.3. Sikora, Lopes, Schibelbain and Magatão (2017) showed that the integer formulation can be of advantage for solving balancing problems with multiple identical tasks.

For this test, the small and medium instances of the dataset described in Section 6.2 were adapted. A new dataset containing 10 copies of each task was built using the same procedure. Therefore, the instances contain either 200 or 500 tasks. The cycle time restriction is also increased to 10,000 and 11,000 for the integer problem.

Table 6.14 contains the summary of the formulations' comparison. Out of the 252 cases, the integer formulation was able to solve more instances (236) compared to the

Table 6.14: Comparison of the binary model to the integer model.

Model	#Cases	#Opt Proven	Sol.Time	Differences
Binary	252	221	510.2	222.2
Integer	252	236	239.6	216.9
Model	GAP = 0%	0% < GAP ≤ 5%	GAP >5%	Fail
Binary	221	16	15	0
Integer	236	11	2	3

binary formulation (221) within the 3,600 seconds time limit. Representing copies of tasks with integer variables also proved to be advantageous for the assembly line balancing implementation sequencing problem: the average solution time reduced from 510.2 to 239.6 seconds. Furthermore, the integer formulation resulted in smaller solution gaps for the instances that both formulations were unable to solve. While the binary formulation had 15 instances with a gap greater than 5%, the integer formulation had only 5 (2 + 3 fails). However, the integer formulation was prone to not finding feasible answers for 3 instances. That is, no solution was found at the given time limit. Nevertheless, the best solutions found for these 3 instances by the binary formulation were also very bad conditioned.

Note that the integer formulation resulted in a lower number of differences. This is due to the different concepts of the formulations. For the binary formulation, if two copies of the same tasks are assigned to different stations, a swap of tasks must be made according to the formulation. The integer formulation, on the other hand, treat each group of tasks as the same task. Therefore swaps of tasks are not accounted for.

Figures 6.5 and 6.6 are examples of two implementation problems. Both cases have the same initial and final solution. For the binary model, either task is unique. Therefore, if a task is not assigned in the same station in the initial and final solution, a necessary change is defined. On the other hand, the integer model treats each type of task as identical tasks. There is no difference between two copies. Hence the integer formulation perceives less changes comparing to the binary formulation (216.9 to 222.2 changes on average).

The two different formulations not only have different computational performances, but also the nature of their answers is different. If the tasks are identical, there is no sense in swapping two identical tasks. In this way, the integer formulation is superior, because

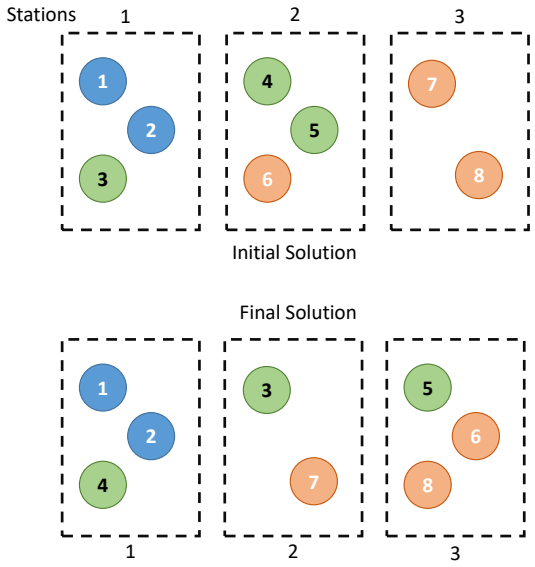

Figure 6.5: Example of a solution for the binary based model. Each color represents a type of task. Note that five tasks changed stations according to the binary formulation.

these differences are not even considered in the formulation. In order to have the same amount of changes, the solution method used for the final solution should consider the total amount of changes or a swap procedure must be applied to the solution.

Finally, Table 6.15 compares the two formulations based on the parameters used to build the dataset. The same conclusions found in Section 6.3.7 can be seen in the integer formulation. The problems became more difficult for greater values of intermediary periods; a greater number of tasks produces more difficult instances; a tighter cycle time limitation is also harder to solve.

Table 6.15: Comparison of the binary and integer models by specific parameters.

			Binary-Model		Integer-Model	
Parameter	Value	#Cases	#Opt	Sol.Time	#Opt	Sol.Time
NP	1	84	78	296.0	83	45.5
	2	84	74	486.4	77	312.0
	3	84	69	748.0	76	361.2
Tasks	200	126	123	114.9	126	0.9
	500	126	98	905.5	110	478.2
CT	10,000	126	102	758.7	112	409.0
	11,000	126	119	261.6	124	70.1
TD	PB	84	84	5.2	84	0.7
	PM	84	57	1321.6	69	663.5
	BI	84	80	203.7	83	54.5

When one compares the results of both formulations, it is clear that the integer

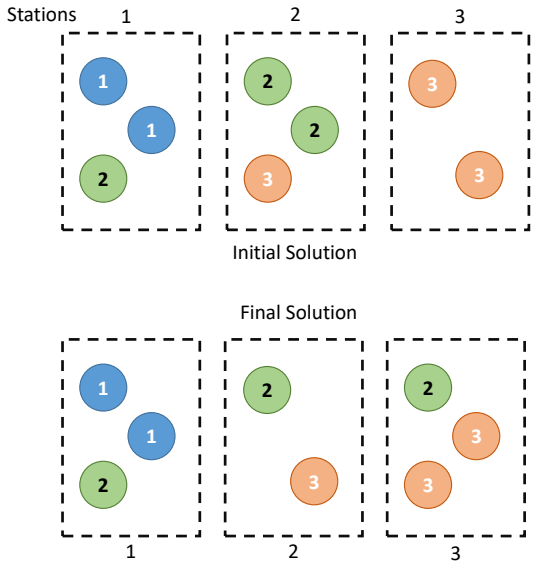

Figure 6.6: Example of a solution for the integer based model. Each color represents a type of task. Note that just one task changed station according to the integer formulation.

formulation dominates the binary for the given dataset. For any subset of the dataset, the integer formulation proved more optimal answers and achieved an overall better solution time.

By the performed test based on the dataset of Section 6.2 it can be inferred that the integer formulation is superior to the binary formulation **in the presence of multiple identical tasks**. This conclusion agrees with what has been found in Sikora, Lopes, Schibelbain and Magatão (2017).

6.4.3 Including the Final Stage

In this final comparison section we show the difference between a model that only sequences the implementation for a given final solution and another with no final fixed solution. Up to this section, the final answer is supposed to be known and obtained by an optimization method. The Final-Stage-Free formulation does not require a final solution: the model or method is free to find the best solution according to both the cycle time and the number of answers.

Solving both the final state balancing problem and the implementation sequencing can bring advantages, such as finding final answers with the least implementation effort.

On the other hand, solving both problems simultaneously can be significantly harder. Therefore, there is a trade-off in the solution quality for a given solution time limit. If there are enough available time the simultaneous approach reaches better solutions, but for a given time limit, solving the implementation problem for a fixed final solution might produce better results.

For this test, the small and medium instances of Section 6.2 were solved without a final solution for a number of periods of 2 and 3 (the final solution now counts as a period). The model was adapted not to consider the final solution as a parameter, that is, Equation 5.9 was ignored. Furthermore, a variable for the cycle time of the last period (CT_{NP}) is used to optimize the final state. The objective function is altered to the one in Expression 6.1. M is a big-M value, used to create the priority of reaching the best cycle time for the final solution, and, as a second objective, the least implementation effort.

$$Minimize : CT_{NP} \cdot M + Effort \tag{6.1}$$

The results detailed in Table 6.16 can be divided in three parts: instances with the same results; instances that improved; and instances with worse answers. The first subset (40 instances) achieved the same results for the cycle time and the number of changes. Which means that the decomposed problem already reached the optimality. The instances that presented improvement (91 instances) reached the same cycle time with less changes than what has been found in Section 6.2. That means that solving the balancing and implementation sequencing simultaneously allowed to reach a significant number of final solutions that are easier to achieve. Finally, a third subset of the dataset (37 instances) produced final state balancing with worse quality than the decomposed method. In those cases, the instance presented itself too hard to achieve the same result. From these 37 instances, in 9 cases the solver was unable to reach any feasible solution.

More detailed results can be seen in Table 6.16. The table contains the solution time for the instances solved with the simple (only implementation sequencing) and the composed approach (both implementation sequencing and balancing). It can be observed that the composed approach produced several better answers for any selected parameter (91 out of the 168 instances). However, the computational burden greatly increases by solving the two problems simultaneously: the solution time increased from 4.0 seconds on average for the simple approach to 1304.9 seconds by combining both problems.

Table 6.16: Comparison of the simple implementation problem and the composed implementation and balancing problem. The column $\#Eq$ contains the number of instances that presented the same results treating the simple or composed problem. The columns $\#B$ and $\#W$ contain the number of instances with better and worse answers obtained by the composed approach, respectively.

Parameter	Value	#Cases	#Opt	#Eq.	#B	#W	Composed Sol.Time	Simple Sol.Time
NP	2	84	59	19	50	15	1229.7	0.5
	3	84	54	21	41	22	1380.0	7.5
Tasks	20	84	84	35	49	0	6.6	0.5
	50	84	29	5	42	37	2603.1	7.6
CT	1000	84	57	20	43	21	1299.4	7.4
	1100	84	56	20	48	16	1310.3	0.6
TD	PB	56	45	23	27	6	905.9	0.5
	PM	56	32	3	41	12	1562.1	8.8
	BI	56	36	14	23	19	1446.5	2.7
PD	BN-0.2	24	12	6	7	11	1807.0	3.9
	BN-0.6	24	15	8	10	6	1443.8	0.6
	CH-0.2	24	14	6	12	6	1631.7	9.7
	CH-0.6	24	18	11	8	5	1059.8	7.6
	MX-0.2	24	12	3	14	7	1806.6	5.3
	MX-0.6	24	18	0	22	2	1360.2	0.6
	MX-0.9	24	24	6	18	0	24.8	0.5
Total	all	168	113	40	91	37	1304.9	4.0

In Section 6.3.7, we have seen the effects of the problem parameters on the solution difficulty. For the simple problem, the number of tasks, the number of periods, and the cycle time proved to be the most influential parameters. Table 6.16 implies that the composed approach is mostly sensitive to the number of tasks. Both the number of periods and the cycle time limitation did not have large differences on the number of optimal answers proved or the solution time. The attenuation of the importance of these parameters may be given to the balancing aspect of the problem. Since in the composed approach the sequencing module is only a part of the problem, these parameters are less important. The number of tasks, however, is also directly related to the difficulty of the line's final balancing.

The results presented in Table 6.16 contain only the number of instances with equal, better or worse results. Additional information on the reduced number of changes are presented herein. Aside the 9 failed instances, the composed approach resulted in implementations with an average of 7.1 changes per transition. The same instances required an average of 10.3 changes when solved by the simple approach. This shows that the implementation effort is strongly dependent of the final answer. If the problems can be solved simultaneously, these preliminary results show that a significant implementation effort can be saved.

6.5 Industrial Case Study and Decomposition Strategies

In this section we briefly describe an industrial application of the implementation sequencing. The inspiration for dividing the implementation in feasible intermediary states appeared in the optimization of a robotic welding line. The optimization model and the balancing case study description can be found in Lopes, Sikora, Molina, Schibelbain, Rodrigues and Magatão (2017). The case consists of a welding line of an automobile manufacturer in the outskirts of Curitiba, Brazil.

The welding line schematic is shown in Figure 6.7. The 42 robots are responsible for about 740 welding points distributed along 13 stations. In Lopes, Sikora, Molina, Schibelbain, Rodrigues and Magatão (2017), a model for the optimization of the division

of points considering interferences and movement times was presented and used to solve the problem. The direct industrial application of the results, however, proved to be difficult. The optimal answer requires 353 changes on task's allocations, requiring more time than the available for the implementation.

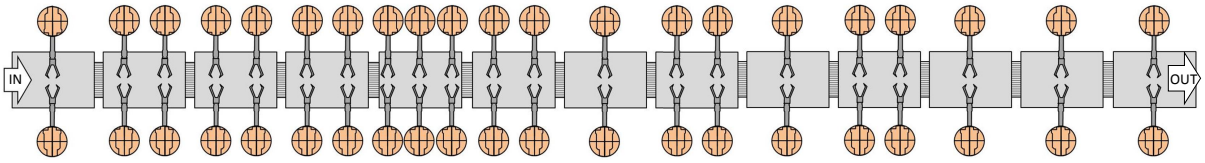

Figure 6.7: Layout of the line optimized containing 42 robots in 13 workstations.

The robot programmers are able to perform not more than 50 changes during one weekend intervention. Therefore, a sequencing of the changes was mandatory for the applicability of the optimal answer. The model from Lopes, Sikora, Molina, Schibelbain, Rodrigues and Magatão (2017) was used to assure the stage feasibility while the methodology described in Chapter 4 was used to model the transitions. Once a maximal limit of changes is fixed, the objective of the problem is to minimize the number of necessary periods for the implementation. As described in Section 3.4, this problem variation consists in an ALIP-1 case.

The balancing core of the model is already computational difficult to be solved (Lopes, Sikora, Molina, Schibelbain, Rodrigues and Magatão, 2017). The sequencing problem containing enough intermediary periods to result in only 50 changes per implementation phase proved not to be solvable with the available hardware and software. The problem resulted in a model with more than 21,000 variables and 41,000 restrictions. Even with eight hours of computing, no single incumbent solution could be found.

The formulation can be used heuristically by decomposing the problem. Instead of modeling all intermediary stages at the same time, two decomposition strategies are proposed.

6.5.1 Iterative Decomposition Strategy

The first decomposition strategy is to sequence the implementation one step at a time in an iterative procedure. From the initial answer, the formulation can be adapted to look

for the stage whose assignment is the closest to the final solution with the limitation of 50 changes, for instance. The answer can be used as the initial answer or a reference stage. The process continues until final configuration can be reached in one implementation phase from the last reference stage obtained.

Table 6.17 shows the results of the iterative procedure applied to the industrial case. In total, 8 transitions are necessary to implement the 353 changes in steps up to 50 changes. This answer presents the minimum number of transitions because there is no way to divide 353 changes in 7 parts of less than 50 changes ($\lceil 353/50 \rceil = 8$). The iterative procedure resulted in a total of 366 changes (13 extra changes) and was solved in 337 seconds.

Table 6.17: Results for the iterative procedure applied to the decomposition of the industrial case of the implementation problem.

Iteration	Initial	1	2	3	4	5	6	7	Final	Total
Changes	-	50	50	50	50	50	50	50	16	366
Solution Time (s)	-	17	26	62	1	215	8	7	1	337
Distance to Final	353	303	253	205	155	112	64	16	0	-

6.5.2 Bipartition Decomposition Strategy

The second decomposition method consists in partitioning the problem in less intermediary states. Once a small number of periods tends to be less difficult to solve (as seen in Section 6.3.7), the implementation can be divided grossly in two, three or more phases. The answer of the gross division can then be used as initial and final stages in smaller and easier problems. One form of applying this procedure is by the bipartition of intervals. The first step calculates the middle point in the implementation. Further steps divide each of the halves in more parts. The procedure continues until the individual parts contain less than the required number of changes.

The bipartition procedure was also applied to the industrial case, resulting in an implementation with eight transition phases. Table 6.18 contains the results and the description of each step. In the first step, an intermediary condition between the initial and final stages is obtained **(1/2)**. The intermediary state **(1/2)** is 177 changes away from the initial stage and 176 changes away from the final stage. The second and third steps

further divides the transitions between **(Initial - 1/2)** and **(1/2 - Final)** stages. Step 2 divides **(Initial - 1/2)** into **(Initial - 1/4)** and **(1/4 - 1/2)** with 89 and 90 changes respectively. The resultant problem was unable to be solved in less than 1800 seconds. The best incumbent found is used as the obtained answer. The procedure continues until all transitions present 50 or less changes between stages. In total, seven steps are used to divide the implementation in eight phases. The bipartition procedure reached an answer with 368 changes in 2428 seconds.

Table 6.18: Results for the bipartition procedure applied to the decomposition of the industrial case of the implementation problem. Each line represent a bipartition step. The lines show the initial and final states from the division, the intermediary state and the resulting changes between the states. *Step 2 has reached the maximal solution time without proving the optimal answer (1800″).

Step	From	States			To	Sol. Time (s)
1	**Initial**	177 changes	**1/2**	176 changes	**Final**	42
2	**Initial**	89 changes	**1/4**	90 changes	**1/2**	1800*
3	**1/2**	88 changes	**3/4**	88 changes	**Final**	7
4	**Initial**	48 changes	**1/8**	48 changes	**1/4**	465
5	**1/4**	45 changes	**3/8**	45 changes	**1/2**	5
6	**1/2**	46 changes	**5/8**	46 changes	**3/4**	104
7	**3/4**	44 changes	**7/8**	44 changes	**Final**	5
Total	**Initial**	368 changes (up to 50 at a time)			**Final**	2428

The decomposition procedures present less computation intensive methods of solving the implementation problem. Although both methods use MILP formulations, their solutions are obtained heuristically. That is, there is no guarantee their answers are optimal. For the proposed study case, however, the optimality can be proven by the lower bound calculation: 353 can only be divided in a minimal of 8 transitions with up to 50 changes. The complete problem could not be solved with a universal solver, but the decomposing strategies proved to be much less time consuming than solving the problem at once.

The formulation and solution of the implementation problem allow the feasibility of the industrial implementation. Without the stepwise division of the necessary changes, an implementation would require weeks of line stoppage to be performed. With the decomposition procedure, answers with enough segmentation were obtained so that the implementation can be realizable in eight weekends that do not necessarily need to be consecutive ones, since the line is within allowable cycle time limits after every intermediary implementation step.

Chapter 7

Conclusions

7.1 Conclusions/Contributions

Since the birth of Operations Research, the study of optimization have been intimately linked to industrial applications. The main concern of decision scientists is on defining and solving problems. The implementation of results depends not only on the best answer found by an algorithm, but also on investments, commercial and political desires in the industry, and the efficient communication between academy and companies. A project approval or the correct implementation may depend on more variables than the clear, defined, and behaved models we create making science. This master thesis intended to point out that some aspects of the implementation of the answers of operations research methods can be treated as optimization subjects. In this manuscript, the models and tests focus on the implementation of a new balancing on assembly lines, but an effort is made to present the equations and formulations as general as possible.

A panorama of the publications on the Assembly Line Balancing Problem (ALBP) is detailed in Chapter 2. In Sections 2.2 to 2.4, the definition and origin of the problem is explained along with the simplification hypotheses that define the Simple Assembly Line Balancing Problem (SALBP). Based on the most recent surveys and publications, the solutions methods (Section 2.5) and several variations and ramifications of the problem are discussed (Section 2.6). From the published material, it can be noticed that little attention from the academy is given to the application and implementation of optimal or

good solutions in the industrial practice, detailed in Section 2.7.

Contribution 1: The presentation of a literature review on ALBP showing the gap between the developed models and solution procedures and the industrial implementation of optimal balancing.

Chapter 3 describes that the Project Scheduling Problem is focused on non-operative systems. This approach is reasonable for projects beginning from scratch: during the implementation phase, the system output is neglected. Re-projecting or altering already established systems may still produce some output during the implementation phase. It can be profitable, and sometimes is the only feasible manner, to schedule the implementation of problems while considering the intermediary stages throughout the system passes during the implementation. This new paradigm is named Implementation Problem and can be observed in several systems as production, transportation, and maintenance. Section 3.1 described how the Implementation Problem can be seen as an extension of the Project Schedule. As it can be seen in Figure 3.5, the Implementation Schedule also considers the tasks of an implementation along with the operational conditions of the system.

Contribution 2: The identification and description of the Implementation Problem, a generalization of the Project Scheduling Problem.

The particularities of implementing a new balancing configuration on real assembly lines is presented in Chapter 3, Section 3.2. The production rhythm and the flow-shop structure bring some difficulties in changing efficiently the production systems. The considerations are mapped and used as boundary conditions to the particularization of the implementation problem applied to assembly lines. The simplification hypotheses, types of problems, variations, and extensions are described along with the Assembly Line Implementation Problem (ALIP) definition (Sections 3.3 and 3.4).

Contribution 3: The observation and description of industrial implementing conditions on assembly lines.

Contribution 4: The definition of the Assembly Line Implementation Problem, as well as its simplification hypotheses and variations.

A modeling guide is presented in Chapter 4 for the directions on how to model the Implementation Problem using Mixed-Linear Integer Programming (MILP). The variables assume a specific form and only in certain contexts the Implementation Problem can be defined. In this master thesis, the variables responsible to measure and control changes are named transition variables. The expressions used to link transition variables to the stages of the problem depend on the nature of the change and the type of variable used. Tables 4.3 to 4.6 contain the expressions used to define and measure a change between two states for every possible combination of types of variables. Furthermore, there are several possibilities for the modeling transitions or implementation phases. Capacity constraints, sequencing, and scheduling possibilities are discussed along with modeling examples.

Contribution 5: The development of a modeling guide for the formulation of Implementation Problem.

Contribution 6: The development of the expressions used to model changes between two state variables.

Contribution 7: The identification and presentation of some of the forms of modeling the transitions between two stages.

Chapter 5 presents MILP formulations of the Implementation Problem applied to assembly line balancing. Two versions of models are given for the simple version of ALIP (Sections 5.2.1 and 5.2.2), along with two models for the integer version of the problem (Sections 5.2.3 and 5.2.4). The possible objective functions and problem types are discussed in Section 5.3. Finally, the variable reduction for the ALBP variables is presented and a new variable reduction procedure for the sequencing variables is proposed (Section 5.4).

Contribution 8: The development of MILP formulations for the binary and integer variations of the Assembly Line Implementation Problem.

The results and tests for the formulations presented in Chapter 5 are described in Chapter 6. Section 6.1 brings didactic instances to exemplify the justification of applying the Implementation Problem on assembly lines. A dataset is developed based on the most recent SALBP dataset (Section 6.2). With the proposed dataset it is possible to observe the effect of the problem's parameters on solving the problem (Section 6.3). The analysis provided insights on the sensitivity of parameters such as the number of tasks, intermediary periods, and cycle time. This master thesis presented multiple models for the same problem, whose performances are compared in Section 6.4. Section 6.4.3 shows that solving both the implementation problem with the balancing problem simultaneously can produce even better results than a decomposition approach (treating the final state optimization and the implementation as different problems).

Contribution 9: The development of a structured dataset for the Assembly Line Implementation Problem.

Contribution 10: The analysis of the effect of each forming characteristic on the solving difficulty of a dataset and the comparison of the proposed formulations.

Section 6.5 presented an industrial case study based on a real assembly line of an automotive manufacturer in the outskirts of Curitiba. The standard modeling was unable to solve the problem based on the problem and model from Lopes, Sikora, Molina, Schibelbain, Rodrigues and Magatão (2017). Due to its large size and the necessity of division in several transition periods, the solver was unable to reach useful answers. Therefore, two decomposition procedures based on the MILP formulation (iterative and bipartition methods) are presented and used to solve heuristically the proposed case study of a welding line consisting of 42 robots and 13 workstations. The decomposition strategies allowed the division of the implementation effort of 353 changes in 8 smaller steps of 50 changes. The complete model for the case has more than 21,000 variables and 41,000 restrictions.

Contribution 11: The presentation of an industrial based case study for the Assembly Line Implementation Problem.

Contribution 12: The development of the iterative and bipartition heuristic solution methods for the Assembly Line Implementation Problem.

7.2 Future Research

The master thesis introduces the Implementation Problem and begins with the MILP formulation and some analysis on the structure of the problem. The manuscript is focused on the presentation and definition of this class of problem and on the derived MILP modeling. Although the intention of the presentation is to be as generic as possible in the modeling of the Implementation Problems in the Modeling Guide, only the models for the Assembly Line Balancing Problem are defined. During the development of the master thesis, some directions for further works that could not be part of the manuscript have been identified. Some extension possibilities for new developments on the Implementation Problems are:

- Although the generic Implementation Problem have been described, other applications are to be found. The corresponding version of the ALIP for the vehicle routing or maintenance problems are example of the potential of optimizing the implementation on other systems.
- Only the most basic versions for the ALIP problem are herein presented. Other types of problems are not yet defined and formulated, specially variations whose transitions require solving bin-packing or scheduling problems.
- The presented results were obtained using a universal solver. No solution method for ALIP is here presented or exists, to the best of the author's knowledge. Specifically designed algorithms can produce better and faster solutions for the proposed problems.
- Although the proposed dataset contemplates several variable parameters (number of tasks, number of periods, cycle time, precedence diagram, etc.), only one instance was generated for each combination of parameters. A more complete analysis on the parameters' influence can be obtained in larger and more detailed datasets.

- The decomposition methods (iterative and bipartition) were used for the solution of the industrial case study only. More studies based on dataset and the development of more sophisticated decomposition techniques are important to improve the resolvability of larger instances of implementation problems.

- No practical analysis have been perform to estimate the implementation cost of tasks. The instances were solved considering the number of changes (by assigning all changing costs to 1). Investigations on real assembly lines are necessary to map the real costs of reassigning tasks to other stations.

- For the solved instances, the cost of all changes is supposed to be the same. Further tests could identify the solving behavior and problem solving difficulty for other values of change cost.

- The Implementation Problem defined and solved in the master thesis is supposed to have deterministic parameters. It can be difficult to estimate with certainty the cost or amount of time necessary to reallocate a task if no historical data exists. Moreover, even with a good estimative, these changes do not necessarily require a fixed amount of time. The stochastic version of the problem (considering stochastic distributions for cost of moving tasks) may be closer to the reality in assembly lines.

Appendix A

Proofs

This appendix contains the development and/or proof of the expressions given in Tables 4.3 - 4.6.

A.1 Binary-Binary relations

This sections continues with the other two expressions (third and fourth from Table 4.3) that are not proven in Chapter 4. They relate to the binary-binary relationship with the equivalence rule.

Firstly, the logical statements of the binary-binary relationships are transformed into atomic statements linked with a logical *and*.

$$\underbrace{x_t \wedge \neg x_{(t-1)}}_{\text{new assignment}} \leftrightarrow y_t^+ \equiv \underbrace{\left((x_t \wedge \neg x_{(t-1)}) \rightarrow y_t^+\right) \wedge \left(\neg(x_t \wedge \neg x_{(t-1)}) \rightarrow \neg y_t^+\right)}_{\text{equivalence}} \equiv$$

$$\equiv \left((x_t \wedge \neg x_{(t-1)}) \rightarrow y_t^+\right) \wedge \left(\underbrace{(\neg x_t \vee x_{(t-1)})}_{\text{De Morgan's law}} \rightarrow \neg y_t^+\right) \equiv$$

$$\equiv \underbrace{\left((x_t \wedge \neg x_{(t-1)}) \rightarrow y_t^+\right) \wedge \underbrace{\left(\neg x_t \rightarrow \neg y_t^+\right) \wedge \left(x_{(t-1)} \rightarrow \neg y_t^+\right)}_{\text{implication}}}_{\text{equivalent statement}} \quad \text{(A.1)}$$

$$\underbrace{x_t \otimes x_{(t-1)}}_{\text{assignment change}} \leftrightarrow y_t \equiv \big((x_t \wedge \neg x_{(t-1)}) \vee (\neg x_t \wedge x_{(t-1)})\big) \leftrightarrow y_t \equiv$$

$$\equiv \underbrace{\underbrace{\big[\big((x_t \wedge \neg x_{(t-1)}) \vee (\neg x_t \wedge x_{(t-1)})\big) \rightarrow y_t\big]}_{i_1} \wedge \underbrace{\big[\neg\big((x_t \wedge \neg x_{(t-1)}) \vee (\neg x_t \wedge x_{(t-1)})\big) \rightarrow \neg y_t\big]}_{i_2}}_{\text{equivalence}}$$

$$i_1 \equiv \underbrace{(x_t \wedge \neg x_{(t-1)} \rightarrow y_t) \wedge (\neg x_t \wedge x_{(t-1)} \rightarrow y_t)}_{\text{implication}}$$

$$i_2 \equiv \underbrace{\big(\neg(x_t \wedge \neg x_{(t-1)}) \wedge \neg(\neg x_t \wedge x_{(t-1)})\big)}_{\text{De Morgan's law}} \rightarrow \neg y_t \equiv \underbrace{(\neg x_t \vee x_{(t-1)})}_{\text{De Morgan's law}} \wedge \underbrace{(x_t \vee \neg x_{(t-1)})}_{\text{De Morgan's law}} \rightarrow \neg y_t \equiv$$

$$\equiv \underbrace{\big((\neg x_t \vee x_{(t-1)}) \wedge x_t\big) \vee \big((\neg x_t \vee x_{(t-1)}) \wedge (\neg x_{(t-1)})\big)}_{\text{distributive law}} \rightarrow \neg y_t \equiv$$

$$\equiv \Bigg[\underbrace{\overbrace{(\neg x_t \wedge x_t)}^{0} \vee (x_{(t-1)} \wedge x_t)}_{\text{distributive law}} \vee \underbrace{(\neg x_t \vee (\neg x_{(t-1)})) \vee \overbrace{(x_{(t-1)} \wedge (\neg x_{(t-1)}))}^{0}}_{\text{distributive law}}\Bigg] \rightarrow \neg y_t \equiv$$

$$\equiv \big[(x_{(t-1)} \wedge x_t) \vee (\neg x_t \vee (\neg x_{(t-1)}))\big] \rightarrow \neg y_t \equiv$$

$$\equiv \underbrace{\big[(x_{(t-1)} \wedge x_t) \rightarrow \neg y_t\big] \wedge \big[(\neg x_t \vee (\neg x_{(t-1)})) \rightarrow \neg y_t\big]}_{\textit{implication}} \quad \text{(A.2)}$$

Both Expressions A.1 and A.2 are now composed of logical propositions united by logical *and* functions. In the translation to MILP expressions, every proposition will correspond to one linear expression. The MILP formulation for the variables y_t^+ (Proposition A.1.1) and y_t (Proposition A.1.2) with equivalence relations to the changes are given herein.

Proposition A.1.1.

$$\underbrace{\big((x_t \wedge \neg x_{(t-1)}) \rightarrow y_t^+\big) \wedge \big(\neg x_t \rightarrow \neg y_t^+\big) \wedge \big(x_{(t-1)} \rightarrow \neg y_t^+\big)}_{\textit{if and only if antecedent then consequent}} \equiv$$

$$\equiv \begin{cases} x_t + (1 - x_{(t-1)}) - y_t^+ \leq 1 \\ (1 - x_t) \leq (1 - y_t^+) \\ x_{(t-1)} \leq (1 - y_t^+) \end{cases} \equiv \underbrace{\begin{cases} y_t^+ \geq x_t - x_{(t-1)} \\ y_t^+ \leq x_t \\ y_t^+ \leq 1 - x_{(t-1)} \end{cases}}_{\textit{MILP expression}}$$

Proof. In proposition A.1.1, x_t and $x_{(t-1)}$ are binary variables, that is, they can assume either values 0 or 1. The truth table of the expression is expressed bellow, along with the resulting MILP expressions. The columns E_1, E_2, and E_3 represent the value y_t^+ has to assume for the three terms of the logical statement: $E_1 = (x_t \wedge \neg x_{(t-1)}) \rightarrow y_t^+$, $E_2 = \neg x_t \rightarrow \neg y_t^+$, and $E_3 = x_{(t-1)} \rightarrow \neg y_t^+$.

x_t	$x_{(t-1)}$	E_1	E_2	E_3	$E_1 \wedge E_2 \wedge E_3$	MILP Exp.	Conclusion
0	0	free	0	free	0	$y_t^+ \geq 0$ $y_t^+ \leq 0$ $y_t^+ \leq 1$	$\therefore y_t^+ = 0$
0	1	free	0	0	0	$y_t^+ \geq -1$ $y_t^+ \leq 0$ $y_t^+ \leq 0$	$\therefore y_t^+ = 0$
1	0	1	free	free	1	$y_t^+ \geq 1$ $y_t^+ \leq 1$ $y_t^+ \leq 1$	$\therefore y_t = 1$
1	1	free	free	0	0	$y_t^+ \geq 0$ $y_t^+ \leq 1$ $y_t^+ \leq 0$	$\therefore y_t = 0$

Therefore, setting x_t and $x_{(t-1)}$ the values (1;0) results in $y_t^+ = 1$, which is equivalent to identify a positive increment on x variables between two periods. Setting x_t and $x_{(t-1)}$ any other value {(0;1), (1;0), and (1;1)} results in $y_t^+ = 0$. This way, y_t^+ is true if and only if a new assignment is made. Thus, proposition A.1.1 holds. □

Proposition A.1.2.

$$\underbrace{\big((x_t \wedge \neg x_{(t-1)}) \rightarrow y_t\big) \wedge \big((\neg x_t \wedge x_{(t-1)}) \rightarrow y_t\big)}_{\textit{if and only if antecedent then consequent}} \wedge$$

$$\wedge \underbrace{\big((x_{(t-1)} \wedge x_t) \rightarrow \neg y_t\big) \wedge \big((\neg x_{(t-1)} \wedge (\neg x_t)) \rightarrow \neg y_t\big)}_{\textit{if and only if antecedent then consequent}} \equiv$$

$$\equiv \begin{cases} x_t + (1 - x_{(t-1)}) - y_t \leq 1 \\ (1 - x_t) + x_{(t-1)} - y_t \leq 1 \\ x_{(t-1)} + x_t - (1 - y_t) \leq 1 \\ (1 - x_t) + (1 - x_{(t-1)}) - (1 - y_t) \leq 1 \end{cases} \equiv \underbrace{\begin{cases} y_t \geq x_t - x_{(t-1)} \\ y_t \geq x_{(t-1)} - x_t \\ y_t \leq 2 - x_t - x_{(t-1)} \\ y_t \leq x_t + x_{(t-1)} \end{cases}}_{\textit{MILP expression}}$$

Proof. In proposition A.1.2, x_t and $x_{(t-1)}$ are binary variables, that is, they can assume either values 0 or 1. The truth table of the expression is expressed bellow, along with the resulting MILP expressions. The columns E_1, E_2, E_3, and E_4 represent the value y_t has to assume for the four terms of the logical statement: $E_1 = (x_t \wedge (\neg x_{(t-1)})) \rightarrow y_t$, $E_2 = (\neg x_t \wedge x_{(t-1)}) \rightarrow y_t$, $E_3 = (x_{(t-1)} \wedge x_t) \rightarrow \neg y_t$, and $E_4 = (\neg x_t \wedge \neg(x_{(t-1)})) \rightarrow \neg y_t$.

x_t	$x_{(t-1)}$	E_1	E_2	E_3	E_4	$E_1 \wedge ... \wedge E_4$	MILP Exp.	Conclusion
0	0	free	free	free	0	0	$\begin{cases} y_t \geq 0 \\ y_t \geq 0 \\ y_t \leq 2 \\ y_t \leq 0 \end{cases}$	$\therefore y_t = 0$
0	1	free	1	free	free	1	$\begin{cases} y_t \geq -1 \\ y_t \geq 1 \\ y_t \leq 1 \\ y_t \leq 1 \end{cases}$	$\therefore y_t = 1$
1	0	1	free	free	free	1	$\begin{cases} y_t \geq 1 \\ y_t \geq -1 \\ y_t \leq 1 \\ y_t \leq 1 \end{cases}$	$\therefore y_t = 1$
1	1	free	free	0	free	0	$\begin{cases} y_t \geq 0 \\ y_t \geq 0 \\ y_t \leq 0 \\ y_t \leq 2 \end{cases}$	$\therefore y_t = 0$

Therefore, setting x_t and $x_{(t-1)}$ the values $\{(0;1)$ and $(1;0)\}$ imposes $y_t = 1$, which is equivalent to a change of assignment. Setting x_t and $x_{(t-1)}$ the values $\{(0;0)$ and $(1;1)\}$ makes $y_t = 0$. This way, y_t^+ is true if and only if an assignment changes. Thus, proposition A.1.2 holds. □

A.2 Integer-Binary or Continuous-Binary relations

This section proves the expressions for the integer-binary or continuous-binary link between state and transition variables. The second, third, and fourth expressions from Table 4.4, that are not proven in Chapter 4, are proven here.

The first expression is the implication form for the binary variable y_t. If there is a change in the value of x_{t-1} to x_t (either positive or negative), y_t must assume 1.

Proposition A.2.1.

$$\big(|x_t - x_{(t-1)}| > 0\big) \rightarrow y_t \quad \equiv \quad \begin{cases} Uy_t \geq x_t - x_{(t-1)} \\ Uy_t \geq x_{(t-1)} - x_t \end{cases}$$

Proof. In proposition A.2.1, the logical expression depends on the value of $|x_t - x_{(t-1)}|$. The expression can be divided in three cases: when $x_t > x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| > 0$), when $x_t < x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| > 0$), and when $x_t = x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| = 0$). Testing all cases in the MILP expression results in:

$$x_t > x_{(t-1)} \rightarrow \begin{cases} Uy_t \geq x_t - x_{(t-1)} > 0 \\ Uy_t \geq x_{(t-1)} - x_t < 0 \end{cases} \therefore \; y_t = 1 \therefore \text{ proposition holds}$$

$$x_t < x_{(t-1)} \rightarrow \begin{cases} Uy_t \geq x_t - x_{(t-1)} < 0 \\ Uy_t \geq x_{(t-1)} - x_t > 0 \end{cases} \therefore \; y_t = 1 \therefore \text{ proposition holds}$$

$$x_t = x_{(t-1)} \rightarrow \begin{cases} Uy_t \geq x_t - x_{(t-1)} = 0 \\ Uy_t \geq x_{(t-1)} - x_t = 0 \end{cases} \therefore \; y_t \geq 0 \therefore \text{ tautology}$$

Therefore, setting $x_t = x_{(t-1)}$ imposes the relation $Uy_t \geq 0$, which is valid for any valid value the variable y_t can assume. Setting $x_t > x_{(t-1)}$ or $x_t < x_{(t-1)}$ makes the constraint $Uy_t > 0$ (with $y_t = 1$) valid. Thus, proposition A.2.1 holds. □

The third expression treats the equivalence relation between $x_t - x_{t-1}$ and y_t^+. The transition variable y_t^+ must assume 1 if $x_t > x_{t-1}$.

Proposition A.2.2.

$$\big(x_t - x_{(t-1)} > 0\big) \leftrightarrow y_t^+ \quad \equiv \quad \begin{cases} Uy_t^+ \geq x_t - x_{(t-1)} \\ (L - \varepsilon)(1 - y_t^+) + \varepsilon \leq x_t - x_{(t-1)} \end{cases}$$

Proof. In proposition A.2.2, the logical expression depends on the value of $x_t - x_{(t-1)}$. The expression can be divided in two cases: when $x_t > x_{(t-1)}$ and when $x_t \leq x_{(t-1)}$. Testing both cases in the MILP expression results in:

$$x_t > x_{(t-1)} \rightarrow \begin{cases} Uy_t \geq x_t - x_{(t-1)} > 0 \\ (L-\varepsilon)(1-y_t^+) + \varepsilon \leq x_t - x_{(t-1)} > 0 \end{cases} \therefore \begin{cases} y_t^+ > 0 \\ y_t^+ \leq 1 \end{cases} \therefore$$

$$\therefore \; y_t^+ = 1 \; \therefore \text{ proposition holds}$$

$$x_t \leq x_{(t-1)} \rightarrow \begin{cases} Uy_t \geq x_t - x_{(t-1)} \leq 0 \\ (L-\varepsilon)(1-y_t^+) + \varepsilon \leq x_t - x_{(t-1)} \leq 0 \end{cases} \therefore \begin{cases} y_t^+ \geq 0 \\ y_t^+ < 1 \end{cases}$$

$$\therefore \; y_t^+ = 0 \; \therefore \text{ proposition holds}$$

Therefore, setting $x_t \leq x_{(t-1)}$ imposes the relation $y_t^+ = 0$, while setting $x_t > x_{(t-1)}$ makes $y_t^+ = 1$. Thus, proposition A.2.2 holds. □

The last integer/continuous-binary relation is the equivalence form of the link between $|x_t - x_{t-1}|$ and y_t. (either positive or negative), y_t must assume 1.

Proposition A.2.3.

$$\big(|x_t - x_{(t-1)}| > 0\big) \leftrightarrow y_t \quad \equiv \quad \begin{cases} U\delta_t^+ \geq x_t - x_{(t-1)} \\ (L-\varepsilon)(1-\delta_t^+) + \varepsilon \leq x_t - x_{(t-1)} \\ U\delta_t^- \geq x_{(t-1)} - x_t \\ (L-\varepsilon)(1-\delta_t^-) + \varepsilon \leq x_{(t-1)} - x_t \\ y_t = \delta_t^+ + \delta_t^- \end{cases}$$

Proof. In proposition A.2.3, the logical expression depends on the value of $|x_t - x_{(t-1)}|$. The expression can be divided in three cases: when $x_t > x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| > 0$), when $x_t < x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| > 0$), and when $x_t = x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| = 0$). Two auxiliary binary variables are used (δ_t^+ and δ_t^-). Testing all cases

in the MILP expression results in:

$$
x_t > x_{(t-1)} \rightarrow \begin{cases} U\delta_t^+ \geq x_t - x_{(t-1)} > 0 \\ (L-\varepsilon)(1-\delta_t^+) + \varepsilon \leq x_t - x_{(t-1)} > 0 \\ U\delta_t^- \geq x_{(t-1)} - x_t < 0 \\ (L-\varepsilon)(1-\delta_t^-) + \varepsilon \leq x_{(t-1)} - x_t < 0 \\ y_t = \delta_t^+ + \delta_t^- \end{cases} \therefore \begin{cases} \delta_t^+ > 0 \\ \delta_t^+ \leq 1 \\ \delta_t^- \geq 0 \\ \delta_t^- < 1 \\ y_t = \delta_t^+ + \delta_t^- \end{cases} \therefore
$$

$$
\therefore \begin{cases} \delta_t^+ = 1 \\ \delta_t^- = 0 \\ y_t = \delta_t^+ + \delta_t^- = 1 \end{cases} \quad \therefore \; y_t = 1 \; \therefore \text{ proposition holds}
$$

$$
x_t < x_{(t-1)} \rightarrow \begin{cases} U\delta_t^+ \geq x_t - x_{(t-1)} < 0 \\ (L-\varepsilon)(1-\delta_t^+) + \varepsilon \leq x_t - x_{(t-1)} < 0 \\ U\delta_t^- \geq x_{(t-1)} - x_t > 0 \\ (L-\varepsilon)(1-\delta_t^-) + \varepsilon \leq x_{(t-1)} - x_t > 0 \\ y_t = \delta_t^+ + \delta_t^- \end{cases} \therefore \begin{cases} \delta_t^+ \geq 0 \\ \delta_t^+ < 1 \\ \delta_t^- > 0 \\ \delta_t^- \leq 1 \\ y_t = \delta_t^+ + \delta_t^- \end{cases} \therefore
$$

$$
\therefore \begin{cases} \delta_t^+ = 0 \\ \delta_t^- = 1 \\ y_t = \delta_t^+ + \delta_t^- = 1 \end{cases} \quad \therefore \; y_t = 1 \; \therefore \text{ proposition holds}
$$

$$
x_t = x_{(t-1)} \rightarrow \begin{cases} U\delta_t^+ \geq x_t - x_{(t-1)} = 0 \\ (L-\varepsilon)(1-\delta_t^+) + \varepsilon \leq x_t - x_{(t-1)} = 0 \\ U\delta_t^- \geq x_{(t-1)} - x_t = 0 \\ (L-\varepsilon)(1-\delta_t^-) + \varepsilon \leq x_{(t-1)} - x_t = 0 \\ y_t = \delta_t^+ + \delta_t^- \end{cases} \therefore \begin{cases} \delta_t^+ \geq 0 \\ \delta_t^+ < 1 \\ \delta_t^- \geq 0 \\ \delta_t^- < 1 \\ y_t = \delta_t^+ + \delta_t^- \end{cases} \therefore
$$

$$
\therefore \begin{cases} \delta_t^+ = 0 \\ \delta_t^- = 0 \\ y_t = \delta_t^+ + \delta_t^- = 0 \end{cases} \quad \therefore \; y_t = 0 \; \therefore \text{ proposition holds}
$$

Therefore, setting $x_t > x_{(t-1)}$ or $x_t < x_{(t-1)}$ imposes $y_t = 1$, while setting $x_t = x_{(t-1)}$ makes $y_t = 0$. Thus, proposition A.2.3 holds. □

A.3 Integer-Integer or Continuous-Continuous relations

Table 4.5 contains the expressions for either the Integer-Integer or Continuous-Continuous relations between the state and transition variables. In total, eight expressions can be derived from the combination of the non-negative y_t^+ or y_t variables, implication or equivalence relations, and inequality or equality consequents.

The expressions for the Continuous-Integer relations can be found in Table 4.6. Although the expressions are different to the ones from Table 4.5, their proof is equivalent. Therefore, only the expressions of Table 4.5 are here described.

Propositions A.3.1 to A.3.8 represent the eight expressions from Table 4.5. Propositions A.3.1 to A.3.4 deal with the y_t^+ variable, while Propositions A.3.5 to A.3.8 are related to y_t.

For this section, we define Δ as a strictly positive value to aid in the proofs. We assume $|x_t - x_{(t-1)}| = \Delta$ when $x_t \neq x_{(t-1)}$. Therefore, when $x_t > x_{(t-1)}$, $x_t - x_{(t-1)} = \Delta$, when $x_t < x_{(t-1)}$, $x_t - x_{(t-1)} = -\Delta$, and when $x_t = x_{(t-1)}$, $x_t - x_{(t-1)} = 0$. Another supposition is that if $x_t > x_{(t-1)}$, then $x_t - x_{(t-1)}$ is at least as great as a small value ε.

Proposition A.3.1.

$$x_t - x_{(t-1)} > 0 \rightarrow y_t^+ \geq x_t - x_{(t-1)} \quad \equiv \quad y_t^+ \geq x_t - x_{(t-1)}$$

Proof. In proposition A.3.1, the logical expression depends on the value of $x_t - x_{(t-1)}$. The expression can be divided in two cases: when $x_t > x_{(t-1)}$ and when $x_t \leq x_{(t-1)}$. Testing both cases in the MILP expression results in:

$$x_t > x_{(t-1)} \rightarrow y_t^+ \geq x_t - x_{(t-1)} = \Delta \quad \therefore \quad y_t^+ \geq \Delta \quad \therefore \quad \text{proposition holds}$$
$$x_t \leq x_{(t-1)} \rightarrow y_t^+ \geq x_t - x_{(t-1)} \leq 0 \quad \therefore \quad y_t^+ \geq 0 \quad \therefore \quad \text{tautology}$$

Therefore, setting $x_t \leq x_{(t-1)}$ imposes the relation $y_t^+ \geq 0$, which is valid for any valid value variable y_t^+ can assume. Setting $x_t > x_{(t-1)}$ makes $y_t^+ \geq \Delta$. Thus, proposition A.3.1 holds. □

Proposition A.3.2.

$$x_t - x_{(t-1)} > 0 \rightarrow y_t^+ = x_t - x_{(t-1)} \quad \equiv \quad \begin{cases} U\delta \geq x_t - x_{(t-1)} \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq x_t - x_{(t-1)} + U(1-\delta) \end{cases}$$

Proof. In proposition A.3.2, the logical expression depends on the value of $x_t - x_{(t-1)}$. The expression can be divided in two cases: when $x_t > x_{(t-1)}$ and when $x_t \leq x_{(t-1)}$. A binary auxiliary variable (δ) is used in the formulation. Testing both cases in the MILP expression results in:

$$x_t > x_{(t-1)} \rightarrow \begin{cases} U\delta \geq x_t - x_{(t-1)} > 0 \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq x_t - x_{(t-1)} + U(1-\delta) \end{cases} \therefore \begin{cases} \delta = 1 \\ y_t^+ \geq x_t - x_{(t-1)} = \Delta \\ y_t^+ \leq x_t - x_{(t-1)} = \Delta \end{cases} \therefore$$

$$\therefore y_t^+ = \Delta \therefore \text{ proposition holds}$$

$$x_t \leq x_{(t-1)} \rightarrow \begin{cases} U\delta \geq x_t - x_{(t-1)} \leq 0 \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \leq 0 \\ y_t^+ \leq x_t - x_{(t-1)} + U(1-\delta) \end{cases} \therefore \begin{cases} \delta \geq 0 \\ y_t^+ \geq 0 \\ y_t^+ \leq -\Delta + U(1-\delta) \rightarrow \delta = 0 \end{cases} \therefore$$

$$\therefore \begin{cases} \delta = 0 \\ y_t^+ \geq 0 \\ y_t^+ \leq U - \Delta \end{cases} \quad \therefore \ 0 \leq y_t^+ \leq U - \Delta \therefore \text{ tautology}$$

Therefore, setting $x_t \leq x_{(t-1)}$ imposes the relation $0 \leq y_t^+ \leq U - \Delta$, which is valid for any valid value less than $U - \Delta$ the variable y_t^+ can assume. Setting $x_t > x_{(t-1)}$ makes the constraint $y_t^+ = \Delta$ valid. Thus, proposition A.3.2 holds. □

Proposition A.3.3.

$$\begin{array}{c} \textit{If } \left(x_t - x_{(t-1)} > 0\right) \textit{ then } \left(y_t^+ \geq x_t - x_{(t-1)}\right) \\ \textit{else } \left(y_t^+ = 0\right) \end{array} \quad \equiv \quad \begin{cases} U\delta \geq x_t - x_{(t-1)} \\ (L-\varepsilon)(1-\delta) + \varepsilon \leq x_t - x_{(t-1)} \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq U\delta \end{cases}$$

Proof. In proposition A.3.3, the logical expression depends on the value of $x_t - x_{(t-1)}$. The expression can be divided in two cases: when $x_t > x_{(t-1)}$ and when $x_t \leq x_{(t-1)}$. A

binary auxiliary variable (δ) is used in the formulation. Testing both cases in the MILP expression results in:

$$x_t > x_{(t-1)} \rightarrow \begin{cases} U\delta \geq x_t - x_{(t-1)} > 0 \\ (L-\varepsilon)(1-\delta) + \varepsilon \leq x_t - x_{(t-1)} > 0 \ (\geq \varepsilon) \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq U\delta \end{cases} \therefore \begin{cases} \delta = 1 \\ \text{tautology} \\ y_t^+ \geq x_t - x_{(t-1)} \\ y_t^+ \leq U \end{cases} \therefore$$

$$\therefore y_t^+ \geq \Delta \ \therefore \text{ proposition holds}$$

$$x_t \leq x_{(t-1)} \rightarrow \begin{cases} U\delta \geq x_t - x_{(t-1)} \leq 0 \\ (L-\varepsilon)(1-\delta) + \varepsilon \leq x_t - x_{(t-1)} \leq 0 \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \leq 0 \\ y_t^+ \leq U\delta \end{cases} \therefore \begin{cases} \text{tautology} \\ \delta = 0 \\ y_t^+ \geq 0 \\ y_t^+ \leq 0 \end{cases} \therefore$$

$$\therefore y_t^+ = 0 \ \therefore \text{ proposition holds}$$

Therefore, setting $x_t \leq x_{(t-1)}$ imposes $y_t^+ = 0$, while setting $x_t > x_{(t-1)}$ makes the constraint $y_t^+ \geq \Delta$ valid. Thus, proposition A.3.3 holds. □

Proposition A.3.4.

$$\begin{gathered} \textit{If } \left(x_t - x_{(t-1)} > 0\right) \textit{ then} \\ \left(y_t^+ = x_t - x_{(t-1)}\right) \textit{ else } \left(y_t^+ = 0\right) \equiv \\ \equiv \textit{If } \left(x_t - x_{(t-1)} \geq 0\right) \textit{ then} \\ \left(y_t^+ = x_t - x_{(t-1)}\right) \textit{ else } \left(y_t^+ = 0\right) \end{gathered} \quad \equiv \quad \begin{cases} U\delta \geq x_t - x_{(t-1)} \\ L(1-\delta) + \varepsilon \leq x_t - x_{(t-1)} \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq x_t - x_{(t-1)} + U(1-\delta) \\ y_t^+ \leq U\delta \end{cases}$$

Proof. In proposition A.3.4, the logical expression depends on the value of $x_t - x_{(t-1)}$. The expression can be divided in two cases: when $x_t > x_{(t-1)}$ and when $x_t \leq x_{(t-1)}$. A binary auxiliary variable (δ) is used in the formulation. Testing both cases in the MILP

expression results in:

$$x_t > x_{(t-1)} \rightarrow \begin{cases} U\delta \geq x_t - x_{(t-1)} > 0 \\ L(1-\delta) + \varepsilon \leq x_t - x_{(t-1)} > 0 (\geq \varepsilon) \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq x_t - x_{(t-1)} + U(1-\delta) \\ y_t^+ \leq U\delta \end{cases} \therefore \begin{cases} \delta = 1 \\ \text{tautology} \\ y_t^+ \geq x_t - x_{(t-1)} \\ y_t^+ \leq x_t - x_{(t-1)} \\ y_t^+ \leq U \end{cases} \therefore$$

$$\therefore y_t^+ = \Delta \ \therefore \text{ proposition holds}$$

$$x_t \leq x_{(t-1)} \rightarrow \begin{cases} U\delta \geq x_t - x_{(t-1)} \leq 0 \\ L(1-\delta) + \varepsilon \leq x_t - x_{(t-1)} \leq 0 \\ y_t^+ \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t^+ \leq x_t - x_{(t-1)} + U(1-\delta) \\ y_t^+ \leq U\delta \end{cases} \therefore \begin{cases} \text{tautology} \\ \delta = 0 \\ y_t^+ \geq x_t - x_{(t-1)} + L \\ y_t^+ \leq x_t - x_{(t-1)} + U \\ y_t^+ \leq 0 \end{cases} \therefore$$

$$\therefore y_t^+ = 0 \ \therefore \text{ proposition holds}$$

Therefore, setting $x_t \leq x_{(t-1)}$ imposes $y_t^+ = 0$, while setting $x_t > x_{(t-1)}$ makes the constraint $y_t^+ = \Delta$ valid. Thus, proposition A.3.4 holds. □

Proposition A.3.5.

$$|x_t - x_{(t-1)}| > 0 \rightarrow y_t \geq |x_t - x_{(t-1)}| \quad \equiv \quad \begin{cases} y_t \geq x_t - x_{(t-1)} \\ y_t \geq x_{(t-1)} - x_t \end{cases}$$

Proof. In proposition A.3.5, the logical expression depends on the value of $|x_t - x_{(t-1)}|$. The expression can be divided in three cases: when $x_t > x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| > 0$), when $x_t < x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| > 0$), and when $x_t = x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| = 0$). Testing all cases in the MILP expression results in:

$$x_t > x_{(t-1)} \rightarrow \begin{cases} y_t \geq x_t - x_{(t-1)} > 0 \\ y_t \geq x_{(t-1)} - x_t < 0 \end{cases} \therefore \ y_t \geq \Delta \therefore \text{ proposition holds}$$

$$x_t < x_{(t-1)} \rightarrow \begin{cases} y_t \geq x_t - x_{(t-1)} < 0 \\ y_t \geq x_{(t-1)} - x_t > 0 \end{cases} \therefore \ y_t \geq \Delta \therefore \text{ proposition holds}$$

$$x_t = x_{(t-1)} \rightarrow \begin{cases} y_t \geq x_t - x_{(t-1)} = 0 \\ y_t \geq x_{(t-1)} - x_t = 0 \end{cases} \therefore \ y_t \geq 0 \therefore \text{ tautology}$$

Therefore, setting $x_t = x_{(t-1)}$ imposes the relation $y_t \geq 0$, which is valid for any valid value the variable y_t can assume. Setting $x_t > x_{(t-1)}$ or $x_t < x_{(t-1)}$ makes the constraint $y_t \geq \Delta$ valid. Thus, proposition A.3.5 holds. □

Proposition A.3.6.

$$|x_t - x_{(t-1)}| > 0 \rightarrow y_t = |x_t - x_{(t-1)}| \quad \equiv \quad \begin{cases} U\delta^+ \geq x_t - x_{(t-1)} \\ U\delta^- \geq x_{(t-1)} - x_t \\ y_t \geq x_t - x_{(t-1)} + L(1-\delta^+) \\ y_t \leq x_t - x_{(t-1)} + U(1-\delta^+) \\ y_t \geq x_{(t-1)} - x_t + L(1-\delta^-) \\ y_t \leq x_{(t-1)} - x_t + U(1-\delta^-) \end{cases}$$

Proof. In proposition A.3.6, the logical expression depends on the value of $|x_t - x_{(t-1)}|$. The expression can be divided in three cases: when $x_t > x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| > 0$), when $x_t < x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| > 0$), and when $x_t = x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| = 0$). The formulation uses two binary auxiliary variables (δ^+ and δ^-).

Testing all cases in the MILP expression results in:

$$x_t > x_{(t-1)} \rightarrow \begin{cases} U\delta^+ \geq x_t - x_{(t-1)} > 0 \\ U\delta^- \geq x_{(t-1)} - x_t < 0 \\ y_t \geq x_t - x_{(t-1)} + L(1-\delta^+) \\ y_t \leq x_t - x_{(t-1)} + U(1-\delta^+) \\ y_t \geq x_{(t-1)} - x_t + L(1-\delta^-) \\ y_t \leq x_{(t-1)} - x_t + U(1-\delta^-) \end{cases} \therefore \begin{cases} \delta^+ = 1 \\ \delta^- \geq 0 \\ y_t \geq \Delta \\ y_t \leq \Delta \\ y_t \geq -\Delta + L(1-\delta^-) \\ y_t \leq -\Delta + U(1-\delta^-) \rightarrow \delta^- = 0 \end{cases} \therefore$$

$$\therefore y_t = \Delta \therefore \text{ proposition holds}$$

$$x_t < x_{(t-1)} \rightarrow \begin{cases} U\delta^+ \geq x_t - x_{(t-1)} < 0 \\ U\delta^- \geq x_{(t-1)} - x_t > 0 \\ y_t \geq x_t - x_{(t-1)} + L(1-\delta^+) \\ y_t \leq x_t - x_{(t-1)} + U(1-\delta^+) \\ y_t \geq x_{(t-1)} - x_t + L(1-\delta^-) \\ y_t \leq x_{(t-1)} - x_t + U(1-\delta^-) \end{cases} \therefore \begin{cases} \delta^+ \geq 0 \\ \delta^- = 1 \\ y_t \geq -\Delta + L(1-\delta^+) \\ y_t \leq -\Delta + U(1-\delta^+) \rightarrow \delta^+ = 0 \\ y_t \geq \Delta \\ y_t \leq \Delta \end{cases} \therefore$$

$$\therefore y_t = \Delta \therefore \text{ proposition holds}$$

$$x_t = x_{(t-1)} \rightarrow \begin{cases} U\delta^+ \geq x_t - x_{(t-1)} = 0 \\ U\delta^- \geq x_{(t-1)} - x_t = 0 \\ y_t \geq x_t - x_{(t-1)} + L(1-\delta^+) \\ y_t \leq x_t - x_{(t-1)} + U(1-\delta^+) \\ y_t \geq x_{(t-1)} - x_t + L(1-\delta^-) \\ y_t \leq x_{(t-1)} - x_t + U(1-\delta^-) \end{cases} \therefore \begin{cases} \delta^+ \geq 0 \rightarrow \text{tautology} \\ \delta^- \geq 0 \rightarrow \text{tautology} \\ y_t \geq L(1-\delta^+) \rightarrow \text{tautology} \\ y_t \leq U(1-\delta^+) \\ y_t \geq L(1-\delta^-) \rightarrow \text{tautology} \\ y_t \leq U(1-\delta^-) \end{cases} \therefore$$

$$\therefore \begin{cases} \delta^+, \delta^- \text{ free} \\ y_t \leq U(1-\delta^+) \quad \therefore\ y_t \leq U(\text{assuming either } \delta^+ \text{ or } \delta^- = 1) \therefore \text{ tautology} \\ y_t \leq U(1-\delta^-) \end{cases}$$

Therefore, setting $x_t = x_{(t-1)}$ imposes the relation $y_t \leq U$, which is valid for any valid value the variable y_t can assume. Setting $x_t > x_{(t-1)}$ or $x_t < x_{(t-1)}$ makes the constraint $y_t = \Delta$ valid. Thus, proposition A.3.6 holds. □

Proposition A.3.7.

$$\begin{array}{c} \textit{If } \big(|x_t - x_{(t-1)}| > 0\big) \textit{ then} \\ \big(y_t \geq |x_t - x_{(t-1)}|\big) \textit{ else } \big(y_t = 0\big) \end{array} \equiv \begin{cases} (L-\varepsilon)(1-\delta^+) + \varepsilon \leq x_t - x_{(t-1)} \\ (L-\varepsilon)(1-\delta^-) + \varepsilon \leq x_{(t-1)} - x_t \\ y_t \leq U\delta^+ + U\delta^- \\ y_t \geq x_t - x_{(t-1)} \\ y_t \geq x_{(t-1)} - x_t \end{cases}$$

Proof. In proposition A.3.7, the logical expression depends on the value of $|x_t - x_{(t-1)}|$. The expression can be divided in three cases: when $x_t > x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| > 0$), when $x_t < x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| > 0$), and when $x_t = x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| = 0$). The formulation uses two binary auxiliary variables (δ^+ and δ^-). Testing all cases in the MILP expression results in:

$$x_t > x_{(t-1)} \rightarrow \begin{cases} (L-\varepsilon)(1-\delta^+) + \varepsilon \leq x_t - x_{(t-1)} > 0 (\geq \varepsilon) \\ (L-\varepsilon)(1-\delta^-) + \varepsilon \leq x_{(t-1)} - x_t < 0 \\ y_t \leq U\delta^+ + U\delta^- \\ y_t \geq x_t - x_{(t-1)} > 0 \\ y_t \geq x_{(t-1)} - x_t < 0 \end{cases} \therefore \begin{cases} \text{tautology} \\ \delta^- = 0 \\ y_t \leq U\delta^+ \rightarrow \delta^+ = 1 \\ y_t \geq \Delta \\ \text{tautology} \end{cases}$$

$\therefore y_t \geq \Delta \ \therefore$ proposition holds

$$x_t < x_{(t-1)} \rightarrow \begin{cases} (L-\varepsilon)(1-\delta^+) + \varepsilon \leq x_t - x_{(t-1)} < 0 \\ (L-\varepsilon)(1-\delta^-) + \varepsilon \leq x_{(t-1)} - x_t > 0 (\geq \varepsilon) \\ y_t \leq U\delta^+ + U\delta^- \\ y_t \geq x_t - x_{(t-1)} < 0 \\ y_t \geq x_{(t-1)} - x_t > 0 \end{cases} \therefore \begin{cases} \delta^+ = 0 \\ \text{tautology} \\ y_t \leq U\delta^- \rightarrow \delta^- = 1 \\ \text{tautology} \\ y_t \geq \Delta \end{cases}$$

$\therefore y_t \geq \Delta \ \therefore$ proposition holds

$$x_t = x_{(t-1)} \rightarrow \begin{cases} (L-\varepsilon)(1-\delta^+) + \varepsilon \leq x_t - x_{(t-1)} = 0 \\ (L-\varepsilon)(1-\delta^-) + \varepsilon \leq x_{(t-1)} - x_t = 0 \\ y_t \leq U\delta^+ + U\delta^- \\ y_t \geq x_t - x_{(t-1)} = 0 \\ y_t \geq x_{(t-1)} - x_t = 0 \end{cases} \therefore \begin{cases} \delta^+ = 0 \\ \delta^- = 0 \\ y_t \leq 0 \\ \text{tautology} \\ \text{tautology} \end{cases}$$

$\therefore y_t = 0 \ \therefore$ proposition holds

Therefore, setting $x_t = x_{(t-1)}$ imposes $y_t = 0$, while setting $x_t > x_{(t-1)}$ or $x_t < x_{(t-1)}$ makes the constraint $y_t \geq \Delta$ valid. Thus, proposition A.3.7 holds. □

Proposition A.3.8.

$$\begin{array}{c} \textit{If } \big(|x_t - x_{(t-1)}| > 0\big) \textit{ then} \\ \big(y_t = |x_t - x_{(t-1)}|\big) \textit{ else } \big(y_t = 0\big) \equiv \\ \equiv \big(y_t = |x_t - x_{(t-1)}|\big) \end{array} \quad \equiv \quad \begin{cases} U\delta \geq x_t - x_{(t-1)} \\ L(1-\delta) \leq x_t - x_{(t-1)} \\ y_t \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t \leq x_t - x_{(t-1)} + U(1-\delta) \\ y_t \geq x_{(t-1)} - x_t + L\delta \\ y_t \leq x_{(t-1)} - x_t + U\delta \end{cases}$$

Proof. In proposition A.3.8, the logical expression depends on the value of $|x_t - x_{(t-1)}|$. The expression can be divided in three cases: when $x_t > x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| > 0$), when $x_t < x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| > 0$), and when $x_t = x_{(t-1)}$ (and therefore $|x_t - x_{(t-1)}| = 0$). The formulation uses one binary auxiliary variable (δ). Testing all cases

in the MILP expression results in:

$$x_t > x_{(t-1)} \rightarrow \begin{cases} U\delta \geq x_t - x_{(t-1)} > 0 \\ L(1-\delta) \leq x_t - x_{(t-1)} > 0 \\ y_t \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t \leq x_t - x_{(t-1)} + U(1-\delta) \\ y_t \geq x_{(t-1)} - x_t + L\delta \\ y_t \leq x_{(t-1)} - x_t + U\delta \end{cases} \therefore \begin{cases} \delta = 1 \\ \text{tautology} \\ y_t \geq \Delta \\ y_t \leq \Delta \\ y_t \geq -\Delta + L \\ y_t \leq -\Delta + U \end{cases} \therefore \begin{cases} \delta = 1 \\ \text{tautology} \\ y_t \geq \Delta \\ y_t \leq \Delta \\ \text{tautology} \\ \text{tautology} \end{cases} \therefore$$

$$\therefore y_t = \Delta \ \therefore \ \text{proposition holds}$$

$$x_t < x_{(t-1)} \rightarrow \begin{cases} U\delta \geq x_t - x_{(t-1)} < 0 \\ L(1-\delta) \leq x_t - x_{(t-1)} < 0 \\ y_t \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t \leq x_t - x_{(t-1)} + U(1-\delta) \\ y_t \geq x_{(t-1)} - x_t + L\delta \\ y_t \leq x_{(t-1)} - x_t + U\delta \end{cases} \therefore \begin{cases} \text{tautology} \\ \delta = 0 \\ y_t \geq -\Delta + L \\ y_t \leq -\Delta + U \\ y_t \geq \Delta \\ y_t \leq \Delta \end{cases} \therefore \begin{cases} \text{tautology} \\ \delta = 0 \\ \text{tautology} \\ \text{tautology} \\ y_t \geq \Delta \\ y_t \leq \Delta \end{cases} \therefore$$

$$\therefore y_t = \Delta \ \therefore \ \text{proposition holds}$$

$$x_t = x_{(t-1)} \rightarrow \begin{cases} U\delta \geq x_t - x_{(t-1)} = 0 \\ L(1-\delta) \leq x_t - x_{(t-1)} = 0 \\ y_t \geq x_t - x_{(t-1)} + L(1-\delta) \\ y_t \leq x_t - x_{(t-1)} + U(1-\delta) \\ y_t \geq x_{(t-1)} - x_t + L\delta \\ y_t \leq x_{(t-1)} - x_t + U\delta \end{cases} \therefore \begin{cases} \delta \geq 0 \\ \delta \leq 1 \\ y_t \geq L(1-\delta) \\ y_t \leq U(1-\delta) \\ y_t \geq L\delta \\ y_t \leq U\delta \end{cases} \therefore \begin{cases} \text{tautology} \\ \text{tautology} \\ \text{tautology} \\ y_t \leq U(1-\delta) \\ \text{tautology} \\ y_t \leq U\delta \end{cases} \therefore$$

$$\therefore \begin{cases} y_t \leq U(1-\delta) \\ y_t \leq U\delta \end{cases} \therefore \ y_t = 0 \ \therefore \ \text{proposition holds}$$

Therefore, setting $x_t = x_{(t-1)}$ imposes $y_t = 0$, while setting $x_t > x_{(t-1)}$ or $x_t < x_{(t-1)}$ makes the constraint $y_t = \Delta$ valid. Thus, proposition A.3.8 holds. □

Bibliography

Agnetis, A., Ciancimino, A., Lucertini, M. and Pizzichella, M. (2007). Balancing flexible lines for car components assembly, *International Journal of Production Research* **33**(2): 333–350.

Ajenblit, D. A. and Wainwright, R. L. (1998). Applying genetic algorithms to the U-shaped assembly line balancing problem, *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC*, IEEE, pp. 96–101.

Anderson, E. J. and Ferris, M. C. (1994). Genetic algorithms for combinatorial optimization: The assembly line balancing problem, *ORSA Journal on Computing* **6**: 161–173.

Andrés, C., Miralles, C. and Pastor, R. (2008). Balancing and scheduling tasks in assembly lines with sequence-dependent setup times, *European Journal of Operational Research* **187**(3): 1212–1223.

Arcus, A. L. (1965). A computer method of sequencing operations for assembly lines, *International Journal of Production Research* **4**(4): 259–277.

Baker, K. R. and Trietsch, D. (2009). *Principles of sequencing and scheduling*, 1st edn, John Wiley & Sons, Hoboken.

Bard, J. F. (1989). Assembly line balancing with parallel workstations and dead time, *International Journal of Production Research* **27**(6): 1005–1018.

Bartholdi, J. J. (1993). Balancing two-sided assembly lines: a case study, *International Journal of Production Research* **31**(10): 2447–2461.

Battaïa, O. and Dolgui, A. (2013). A taxonomy of line balancing problems and their solution approaches, *International Journal of Production Economics* **142**(2): 259–277.

Battini, D., Faccio, M., Ferrari, E., Persona, A. and Sgarbossa, F. (2007). Design configuration for a mixed-model assembly system in case of low product demand, *The International Journal of Advanced Manufacturing Technology* **34**(1-2): 188–200.

Bautista, J. and Pereira, J. (2002). Ant algorithms for assembly line balancing, *Lecture Notes in Computer Science* **2463**: 65–75.

Bautista, J. and Pereira, J. (2007). Ant algorithms for a time and space constrained assembly line balancing problem, *European Journal of Operational Research* **177**(3): 2016–2032.

Bautista, J. and Pereira, J. (2009). A dynamic programming based heuristic for the assembly line balancing problem, *European Journal of Operational Research* **194**(3): 787–794.

Bautista, J., Suarez, R., Mateo, M. and Companys, R. (2000). Local search heuristics for the assembly line balancing problem with incompatibilities between tasks, *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, San Francisco - CA, pp. 2404–2409.

Baybars, I. (1986). Survey of exact algorithms for the simple assembly line balancing problem, *Management Science* **32**(8): 909–932.

Baykasolu, A. and Özbakr, L. (2006). Stochastic U-line balancing using genetic algorithms, *The International Journal of Advanced Manufacturing Technology* **32**(1-2): 139–147.

Becker, C. and Scholl, A. (2006). A survey on problems and methods in generalized assembly line balancing, *European Journal of Operational Research* **168**: 694–715.

Blum, C. (2008). Beam-ACO for simple assembly line balancing, *INFORMS Journal on Computing* **20**(4): 618–627.

Bowman, E. H. (1960). Assembly-Line Balancing by Linear Programming, *Operations Research* **8**(3): 385–389.

Boysen, N., Fliedner, M. and Scholl, A. (2007). A classification of assembly line balancing problems, *European Journal of Operational Research* **183**(2): 674–693.

Boysen, N., Fliedner, M. and Scholl, A. (2008). Assembly line balancing: Which model to use when?, *International Journal of Production Economics* **111**(2): 509–528.

Buxey, G. M. (1974). Assembly Line Balancing With Multiple Stations, *Management Science* **20**(6): 1010–1021.

Chakravarty, A. K. (1988). Line balancing with task learning effects, *IIE Transactions (Institute of Industrial Engineers)* **20**(2): 186–193.

Chiang, W.-C. (1998). The application of a tabu search metaheuristic to the assembly line balancing problem, *Annals of Operations Research* **77**: 209–227.

Costa, A. M. and Miralles, C. (2009). Job rotation in assembly lines employing disabled workers, *International Journal of Production Economics* **120**(2): 625–632.

Daoud, S., Chehade, H. and Yalaoui, F. (2014). Solving a robotic assembly line balancing problem using efficient hybrid methods, *Journal of Heuristics* **20**: 235–259.

Falkenauer, E. (2005). Line balancing in the real world, *Proceedings of the International Conference on Product Lifecycle Management, PLM 05*, Lyon.

Falkenauer, E. and Delchambre, A. (1992). A genetic algorithm for bin packing and line balancing, *Proceedings - IEEE International Conference on Robotics and Automation*, Vol. 2, pp. 1186–1192.

Fattahi, P., Roshani, A. and Roshani, A. (2011). A mathematical model and ant colony algorithm for multi-manned assembly line balancing problem, *The International Journal of Advanced Manufacturing Technology* **53**(1): 363–378.

Fleszar, K. and Hindi, K. S. (2003). An enumerative heuristic and reduction methods for the assembly line balancing problem, *European Journal of Operational Research* **145**(3): 606–620.

Fulkerson, D. R. (1961). A Network Flow Computation for Project Cost Curves, *Management Science* **7**(2): 167–178.

Gamberini, R., Gebennini, E., Grassi, A. and Regattieri, A. (2009). A multiple single-pass heuristic algorithm solving the stochastic assembly line rebalancing problem, *International Journal of Production Research* **47**(8): 2141–2164.

Gersting, J. L. (2006). *Mathematical Structures for Computer Science*, W. H. Freeman & Co., New York, NY, USA.

Gonçalves, J. F. and De Almeida, J. R. (2002). A hybrid genetic algorithm for assembly line balancing, *Journal of Heuristics* **8**: 629–642.

Hartmann, S. and Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem, *European Journal of Operational Research* **207**(1): 1–14.

Heinrici, A. (1994). A comparison between simulated annealing and tabu search with an example from the production planning, *in* H. Dyckhoff (ed.), *Operations Research Proceedings 1994*, Springer, Berlin, pp. 498–503.

Held, M., Karp, R. and Shareshian, R. (1963). Assembly line balancing - Dynamic programming with precedence constraints, *Operations Research* **11**(3): 442–459.

Helgeson, W. B. and Birnie, D. P. (1961). Assembly Line Balancing Using the Ranked Positional Weight Technique, *Journal of Industrial Engineering* **12**: 394–398.

Hillier, F. S. and Lieberman, G. J. (2010). *Introduction to Operations Research*, 10th edn, McGraw-Hill Higher Education.

Hoffmann, T. (1963). Assembly line balancing with a precedence matrix, *Management Science* **9**(4): 551–562.

Hoffmann, T. R. (1992). Eureka: A hybrid system for assembly line balancing, *Management Science* **38**(1): 39–47.

Jackson, J. (1956). A computing procedure for a line balancing problem, *Management Science* **2**(3): 261–271.

Johnson, R. V. (1988). Optimally Balancing Large Assembly Lines with Fable, *Management Science* **34**(2): 240–253.

Kelley, J. E. J. (1961). Critical-Path Planning and Scheduling: Mathematical Basis, *Operations Research* **9**(3): 296–320.

Kim, H. and Park, S. (1995). Strong cutting plane algorithm for the robotic assembly line balancing problem, *International Journal of Production Research* **33**(8): 2311–2323.

Kim, Y. K., Kim, Y. and Kim, Y. J. (2000). Two-sided assembly line balancing: A genetic algorithm approach, *Production Planning & Control* **11**(1): 44–53.

Klein, R. and Scholl, A. (1996). Maximizing the Production Rate in Simple Assembly Line Balancing - A Branch and Bound Procedure, *European Journal of Operational Research* **91**: 367–385.

Kottas, J. F. and Lau, H. S. (1973). Cost-oriented approach to stochastic line balancing, *AIIE Transactions* **5**(2): 164–171.

Krajewski, L. J., Ritzman, L. P. and Malhotra, M. K. (2013). *Operations Management: Processes and Supply Chains*, 10th edn, Pearson.

Lapierre, S. D., Ruiz, A. and Soriano, P. (2006). Balancing assembly lines with tabu search, *European Journal of Operational Research*, Vol. 168, pp. 826–837.

Levitin, G., Rubinovitz, J. and Shnits, B. (2006). A genetic algorithm for robotic assembly line balancing, *European Journal of Operational Research* **168**: 811–825.

Lopes, T. C., Michels, A. S., Molina, R. G., Sikora, C. G. S. and Magatão, L. (2016). An MILP formulation for robotic assembly line design with parallel stations, dead time and equipment selection, *XLVIII Simpósio Brasileiro de Pesquisa Operacional*, Vitória - ES, pp. 3257–3258.

Lopes, T. C., Sikora, C. G. S. and Magatão, L. (2016). Buffer and Cyclical Product Sequence Aware Assembly Line Balancing Problem: Model and Steady-State Balancing Case Study, *XLVIII Simpósio Brasileiro de Pesquisa Operacional*, Vitória - ES, pp. 3459–3470.

Lopes, T. C., Sikora, C. G. S., Michels, A. S. and Magatão, L. (2017a). Balancing, Cyclical Scheduling and Buffer Allocation Mixed-Model Assembly Line Problem: Model and Case Studies, *International Journal of Production Research* **in review**.

Lopes, T. C., Sikora, C. G. S., Michels, A. S. and Magatão, L. (2017b). Mixed-Model Assembly Lines Balancing with Given Buffers and Product Sequence: Model, Formulation Comparisons and Case Study, *Annals of Operations Research* **in review**.

Lopes, T. C., Sikora, C. G. S., Molina, R. G., Schibelbain, D., Rodrigues, L. C. d. A. and Magatão, L. (2017). Balancing a Robotic Welding Manufacturing Line: Problem, Model and Case Study, *European Journal of Operational Research* **in review**.

Magatão, L. (2005). *Mixed integer linear programming and constraint logic programming: towards a unified modeling framework*, PhD thesis, CEFET-PR.

Makssoud, F., Battaïa, O., Dolgui, A., Mpofu, K. and Olabanji, O. (2015). Re-balancing problem for assembly lines: new mathematical model and exact solution method, *Assembly Automation* **35**(1): 16–21.

Malcolm, D. G., Roseboom, J. H., Clark, C. E. and Fazar, W. (1959). Application of a Technique for Research and Development Program Evaluation, *Operations Research* **7**(5): 646–669.

Meira, M. d. S., Sikora, C. G. S., Drevek, P. R., Lüders, R., Magatão, L. and Rodrigues, L. C. d. A. (2016). Optimização do Setup de uma Célula de Manufatura de Portas, *25th SAE BRASIL International Congress and Display*, SAE Technical Paper Series, São Paulo, pp. 1–8.

Michels, A. S., Dias, F. A., Sikora, C. G. S. and Magatão, L. (2016). Using a LP Model for the Design Analysis of an Integrated Renewable Energy System, *XLVIII Simpósio Brasileiro de Pesquisa Operacional*, Vitória - ES, pp. 1029–1040.

Michels, A. S., Lopes, T. C., Sikora, C. G. S. and Magatão, L. (2017). The Robotic Assembly Line Design (RALD) Problem: Model and case studies with practical extensions, *Computers & Industrial Engineering* **in review**.

Miltenburg, J. and Wijngaard, J. (1994). U-line line balancing problem, *Management Science* **40**(10): 1378–1388.

Miralles, C., García-Sabater, J. P., Andrés, C. and Cardós, M. (2008). Branch and bound procedures for solving the Assembly Line Worker Assignment and Balancing Problem: Application to Sheltered Work centres for Disabled, *Discrete Applied Mathematics* **156**(3): 352–367.

Moreira, M. C. O., Miralles, C. and Costa, A. M. (2015). Model and heuristics for the Assembly Line Worker Integration and Balancing Problem, *Computers & Operations Research* **54**: 64–73.

Nearchou, A. C. (2007). Balancing large assembly lines by a new heuristic based on differential evolution method, *The International Journal of Advanced Manufacturing Technology* **34**(9-10): 1016–1029.

Otto, A., Otto, C. and Scholl, A. (2013). Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing, *European Journal of Operational Research* **228**(1): 33–45.

Otto, A. and Scholl, A. (2011). Incorporating ergonomic risks into assembly line balancing, *European Journal of Operational Research* **212**(2): 277–286.

Otto, C. and Otto, A. (2014). Multiple-source learning precedence graph concept for the automotive industry, *European Journal of Operational Research* **234**: 253–265.

Özcan, U. (2010). Balancing stochastic two-sided assembly lines: A chance-constrained, piecewise-linear, mixed integer program and a simulated annealing algorithm, *European Journal of Operational Research* **205**(1): 81–97.

Öztürk, C., Tunali, S., Hnich, B. and Örnek, A. (2015). Cyclic scheduling of flexible mixed model assembly lines with paralel stations, *Journal of Manufacturing Systems* **36**(1): 147–158.

Pape, T. (2015). Heuristics and lower bounds for the simple assembly line balancing problem type 1: Overview, computational tests and improvements, *European Journal of Operational Research* **240**: 32–42.

Patterson, J. H. and Albracht, J. J. (1975). Assembly-Line Balancing: Zero-One Programming with Fibonacci Search, *Operations Research* **23**(1): 166–172.

Pritsker, A. A. B., Waiters, L. J. and Wolfe, P. M. (1969). Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach, *Management Science* **16**(1): 93–108.

Ritt, M. and Costa, A. M. (2015). Improved integer programming models for simple assembly line balancing and related problems, *International Transactions in Operational Research* .

Rubinovitz, J., Bukchin, J. and Lenz, E. (1993). RALB - A Heuristic Algorithm for Design and Balancing of Robotic Assembly Lines, *CIRP Annals - Manufacturing Technology* **42**(1): 497–500.

Sabuncuoglu, I., Erel, E. and Tanyer, M. (2000). Assembly line balancing using genetic algorithms, *Journal of Intelligent Manufacturing* **11**(3): 295–310.

Salveson, M. (1955). The assembly line balancing problem, *Journal of Industrial Engineering* **6**(3): 18–25.

Sawik, T. (2012). Batch versus cyclic scheduling of flexible flow shops by mixed-integer programming, *International Journal of Production Research* **50**(18): 5017–5034.

Schibelbain, D., Lopes, T. C., Sikora, C. G. S., Molina, R. G. and Magatão, L. (2017). Procédé de réduction de temps de cycles d'une ligne automatisée de soudage et installation de soudage correspondante.

Scholl, A. (1993). Data of Assembly Line Balancing Problems, *Technical report*, Darmstadt Technical University, Department of Business Administration, Economics and Law, Institute for Business Studies (BWL), Darmstadt.

Scholl, A. (1999). *Balancing and Sequencing of Assembly Lines*, second edn, Physica, Heidelberg.

Scholl, A. and Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing, *European Journal of Operational Research* **168**(3): 666–693.

Scholl, A., Boysen, N. and Fliedner, M. (2008). The sequence-dependent assembly line balancing problem, *OR Spectrum* **30**(3): 579–609.

Scholl, A., Boysen, N. and Fliedner, M. (2013). The assembly line balancing and scheduling problem with sequence-dependent setup times: problem extension, model formulation and efficient heuristics, *OR Spectrum* **35**: 291–320.

Scholl, A., Fliedner, M. and Boysen, N. (2010). Absalom: Balancing assembly lines with assignment restrictions, *European Journal of Operational Research* **200**(3): 688–701.

Scholl, A. and Klein, R. (1997). SALOME: A bidirectional branch-and-bound procedure for assembly line balancing, *INFORMS Journal on Computing* **9**(4): 319–334.

Scholl, A. and Klein, R. (1999). Balancing assembly lines effectively - A computational comparison, *European Journal of Operational Research* **114**(1): 50–58.

Scholl, A. and Voß, S. (1996). Simple assembly line balancing - heuristic approaches, *Journal of Heuristics* **2**(3): 217–244.

Schrage, L. (1970). Solving Resource-Constrained Network Problems by Implicit EnumerationNonpreemptive Case, *Operations Research* **18**(2): 263–278.

Sewell, E. C. and Jacobson, S. H. (2012). A Branch, Bound, and Remember Algorithm for the Simple Assembly Line Balancing Problem, *INFORMS Journal on Computing* **24**(3): 433–442.

Sikora, C. G. S., Lopes, T. C., Lopes, H. S. and Magatão, L. (2016). Genetic algorithm for type-2 assembly line balancing, *2015 Latin-America Congress on Computational Intelligence, LA-CCI 2015*.

Sikora, C. G. S., Lopes, T. C. and Magatão, L. (2015). Variable Sets Reduction for Assembly Line Balancing Problem: MILP Model and Case Studies, *XLVII Simpósio de Brasileiro de Pesquisa Operacional*, Porto de Galinhas, pp. 166–176.

Sikora, C. G. S., Lopes, T. C. and Magatão, L. (2016). Implementing Changes in the Balancing of Assembly Lines: a Sequencing Problem Addressed by MILP, *XLVIII Simpósio de Brasileiro de Pesquisa Operacional*, Vitória, pp. 73–83.

Sikora, C. G. S., Lopes, T. C. and Magatão, L. (2017). Traveling worker assembly line (re)balancing problem: model, reduction techniques, and real case studies, *European Journal of Operational Research* **259**(3): 949–971.

Sikora, C. G. S., Lopes, T. C., Schibelbain, D. and Magatão, L. (2017). Integer based formulation for the simple assembly line balancing problem with multiple identical tasks, *Computers & Industrial Engineering* **104**: 134–144.

Sternatz, J. (2014). Enhanced multi-Hoffmann heuristic for efficiently solving real-world assembly line balancing problems in automotive industry, *European Journal of Operational Research* **235**: 740–754.

Sungur, B. and Yavuz, Y. (2015). Assembly line balancing with hierarchical worker assignment, *Journal of Manufacturing Systems* **37**: 290–298.

Talbot, F. B., Patterson, J. H. and Gehrlein, W. V. (1986). A Comparative Evaluation of Heuristic Line Balancing Techniques, *Management Science* **32**(4): 430–454.

Thangavelu, S. R. and Shetty, C. M. (1971). Assembly Line Balancing by Zero-One Integer Programming, *AIIE Transactions* **3**(1): 61–68.

Thomopoulos, N. T. (1967). Line Balancing-Sequencing for Mixed-Model Assembly, *Management Science* **14**(2): 59–75.

Thomopoulos, N. T. (1970). Mixed Model Line Balancing with Smoothed Station Assignments, *Management Science* **16**(9): 593–603.

Wee, T. and Magazine, M. (1982). Assembly line balancing as generalized bin packing, *Operations Research Letters* **1**(2): 56–58.

White, W. W. (1961). Comments on a Paper by Bowman, *Operations Research* **9**(2): 274–276.

Wiest, J. D. (1964). Some Properties of Schedules for Large Projects with Limited Resources, *Operations Research* **12**(3): 395–418.

Yazgan, H. R., Beypinar, I., Boran, S. and Ocak, C. (2011). A new algorithm and multi-response Taguchi method to solve line balancing problem in an automotive industry, *The International Journal of Advanced Manufacturing Technology* **57**(1-4): 379–392.

Zha, J. and Yu, J.-j. (2014). A hybrid ant colony algorithm for U-line balancing and rebalancing in just-in-time production environment, *Journal of Manufacturing Systems* **33**(1): 93–102.