TECHNICAL UNIVERSITY OF CATALONIA

Doctoral Program:

ADVANCED AUTOMATION AND ROBOTICS

Doctoral Thesis

ASALBP: the Alternative Subgraphs Assembly Line Balancing Problem. Formalization and Resolution Procedures

Liliana Capacho Betancourt

Thesis Advisor: Dr. Rafael Pastor Moreno

Institute of Industrial and Control Engineering

December 2007

I would like to dedicate this doctoral thesis to:
my mother Mariela

and to the beloved memory of my grandparents:
Enrique and Maria (as she liked to be called).

Acknowledgements

First of all, I would like to thank my supervisor Dr. Rafael Pastor for giving the opportunity of developing my PhD thesis under his supervision. I am grateful for his guidance and opportune suggestions. I also thank Dr. Pastor for carefully reviewing a significant number of written works including this dissertation and for his timely corrections. All the support provided helped me to conclude this doctoral thesis successfully.

I express my gratitude to all members of the Institute for Control and Industrial Engineering for creating a pleasant place to work and for providing me with all the resources necessary to carry out this work. A thank goes to Carma, Noemi and Marta for their continuous support and their assistance for solving the administrative matters. Also, I want to thank Vicenc for keeping under control the IT aspects of the Institute. Furthermore, I acknowledge the library staff Pilar and Montse.

I also want to thank Dr. Alexander Dolgui for giving me the opportunity to work with his research group, under his guidance, at the Division for Industrial Engineering and Computer Sciences, Ecole des Mines, Saint Etienne, France. I thank Dr. Dolgui as well as Dr. T. Jimenez for being co-examiners of this doctoral thesis. A special thank goes to Olga Gunshinskaya for her friendship, for offering her useful advice and for sharing with me long hours writing research papers!. I also want to thank Sana, Natasha, Midhi, Amellie, Hanane, Viviana and Vincent for making easier and more pleasant my stay in Saint Etienne.

Risking omitting someone, I will dare to mention some people that have positively influenced me over these last years and that made my life more enjoyable while I was carrying out my doctoral studies. I thank them all for their support, for giving me a hand when facing difficult moments and for the great times we spent together. In particular I thank Emmanuel, Ray, Ericka, Gerrit, Israel, Orlando, Adolfo, Albert, Viviana, Marcelo, Mayra, Luis, Meralys, Ronald, Antonio, Arturo, Cristina, Claudinha, Sandrita, Claudia, Duarte and Mario. ... I will miss our meetings and our international dinners!.

I express my gratitude to my special friends Enric (Henk Jan): I thank my little Dutch brother for taking care of me, I really enjoyed living with you these last years; Greg: we share plenty of funny moments!; Juan Manuel (Juamma): thanks for your support and for providing us with your remarkable writings; and Liliana: thanks for being an unconditional friend and for your support. I had many great times and a lot of fun with you and Luis, I will be always grateful to both of you!.

A deep thank goes to my Portuguese sister Ritinha for sharing many special moments, long and interesting conversations. Thanks for giving me the opportunity to meet you lovely family, now also my family -my truthful gratitude goes to the complete Marquez family! They took a good care of me any time I was in Portugal.

I also want to thank Ramon and Gloria and Mr. Pompeyo, the kind owners of the flat whom not only provided me with a home, for most of my stay in Barcelona, but also with a friendship.

Finally and most importantly, I would like to thank my family: specially my mother Mariela and my father Adolfo; my brothers Ricardo, Carlos and Fito; and my sisters Nancy and Maricela. They always supported me and, from far away, they motivated and encouraged me to continue towards the completion of this thesis. This achievement is theirs. I thank them all for believing in me, sometimes more than what I believe myself. Coming back home was always a good motivator. I am deeply grateful to my mother for all her love, unconditional support and for teaching me to be persevering and to have the determination to accomplish any proposed goal.

This doctoral thesis has been financed by a Venezuelan grant by the University of Los Andes. I am thankful to my Alma Mater for giving such an opportunity. Additionally, this project has been supported by the project DELIMER DPI2004-03472 from the Spanish Ministry of Science and Technology.

Summary

Nowadays assembly line balancing problems are commonly found in most industrial and manufacturing systems. Basically, these problems seek to assign a set of assembly tasks to an ordered sequence of workstations in such a way that precedence constraints are maintained and a given efficiency measure (e.g. the number of workstations or the cycle time) is optimized.

Because of the computational complexity of balancing problems, research works traditionally considered numerous simplifying assumptions in which, for example, a single model of a unique product were processed in a single line; moreover, problems were mainly restricted by precedence and cycle time constrains. Nevertheless, the current availability of computing resources and the enterprises need to adapt to rapid changes in production and manufacturing processes have encouraged researchers and decision-makers to address more realistic problems. Some examples include problems that involve mixed models, parallel workstations and parallel lines, multiple objectives and also further restrictions such as workstation processing capacity and resource allocation constraints.

This doctoral thesis addresses a novel assembly line balancing problem, entitled here ASALBP: *the Alternative Subgraphs Assembly Line Balancing Problem*, which considers alternative variants for different parts of an assembly or manufacturing process. Each variant can be represented by a precedence subgraph that establishes the tasks required to process a particular product, their precedence requirements and their processing times.

Therefore, to efficiently solve the Alternative Subgraphs Assembly Line Balancing Problem two subproblems need to be solved simultaneously: (1) the decision problem that selects one assembly variant for each part that admit alternatives and (2) the balancing problem that assigns the tasks to the workstations.

The analysis of the state-of-the-art carried out revealed that the Alternative Subgraphs Assembly Line Balancing Problem has not been addressed before in literature studies, which leaded to the characterization and definition of this new problem. Moreover, due to the impossibility of representing assembly variants in a standard precedence graph, the *S-Graph* is proposed here as a diagramming tool to represent all available assembly alternatives in a unique graph.

Different approaches are used here to address the ASALBP. The problem is formalize and optimally solved by means of two mathematical programming models. An approximate approach is used to address industrial-scale problems. Furthermore, local optimization procedures are proposed aiming at improving the quality of the solutions provided by all heuristic methods developed here.

Resumen

Hoy en día, los problemas de equilibrado de líneas de montaje se encuentran comúnmente en la mayoría de sistemas industriales y de manufactura. Básicamente, estos problemas consisten en asignar un conjunto de tareas a una secuencia ordenada de estaciones de trabajo, de manera que se respeten las restricciones de precedencia y se optimice una medida de eficiencia dada (como, por ejemplo, el número de estaciones de trabajo o el tiempo ciclo).

Dada la complejidad de los problemas de equilibrado de líneas, en los trabajos de investigación tradicionalmente se consideraban numerosas simplificaciones en las que, por ejemplo, una sola línea serial procesaba un único modelo de un solo producto. Además, los problemas estaban principalmente restringidos por las relaciones de precedencia y el tiempo ciclo. Sin embargo, la disponibilidad de recursos computacionales de hoy en día, así como la necesidad de las empresas a adaptarse a los rápidos cambios en los procesos de producción, han motivado tanto a investigadores como a gerentes a tratar problemas más realistas. Algunos ejemplos incluyen problemas que procesan modelos mixtos, estaciones de trabajo y líneas en paralelo, consideran múltiples objetivos y restricciones adicionales, como la capacidad de proceso de las estaciones de trabajo y la ubicación de los recursos en la línea.

Esta tesis doctoral trata un nuevo problema de equilibrado de líneas, que ha sido titulado ASALBP: *the Alternative Subgraphs Assembly Line Balancing Problem*, en el que se consideran variantes alternativas para diferentes partes de un proceso de montaje o de manufactura. Cada alternativa puede ser representada por un subgrafo de precedencias, que determina las tareas requeridas para procesar un producto particular, las restricciones de precedencia y los tiempos de proceso.

Para resolver eficientemente el ASALBP, se deben resolver dos problemas simultáneamente: (1) el problema de decisión para seleccionar un subgrafo de montaje para cada parte que admite alternativas y (2) el problema de equilibrado para asignar las tareas a las estaciones de trabajo.

El análisis del estado del arte revela que este problema no ha sido estudiado previamente en la literatura, lo que ha conducido a la caracterización y a la definición de un nuevo problema. Por otra parte, dado que no es posible representar las variantes de montaje en un diagrama de precedencias estándar, se propone el *S-grafo* como una herramienta de diagramación, para representar en un único grafo todas las alternativas de montaje.

Para resolver el ASALBP se usan varios enfoques. El problema se formaliza y se resuelve de manera óptima a través de dos modelos de programación matemática. Un enfoque aproximativo es usado para resolver problemas de tamaño industrial. Además, se proponen procedimientos de optimización local con el objetivo de mejorar la calidad de las soluciones obtenidas por los métodos heurísticos desarrollados en este trabajo.

Table of Content

Index of Figures

Chapter 4. Mathematical Models of the ASALBP

Chapter 5. Approximate Methods to Solve the ASALBP

Index of Tables

Chapter 1

Introduction

1.1 Presentation and Justification

Assembly lines are nowadays commonplace in many production and manufacturing systems, particularly those entailing a large volume of a single product. They maximize the division of labour, thereby maximizing system productivity (Amen (2001)). Therefore, the configuration of the line and the distribution of work along the line are fundamental to the system's efficiency. A complex optimization problem arises when technological constraints and a given objective are also taken into account: the line balancing problem.

In an *Assembly Line Balancing Problem* (ALBP) a set of tasks have to be assigned to an ordered sequence of workstations in such a way that precedence constraints are maintained and a given efficiency measure is optimized, such as, for example, the number of workstations or the workstation time (i.e. the cycle time). In the simplest case, referred to in the literature as SALBP: Simple Assembly Line Balancing Problem (e.g., Baybars (1986), Scholl and Becker (2006)), a serial line processes a single model of one product. Basically, the problem is restricted by technological precedence relations and the cycle time constrains. On the other hand, GALBP: Generalized Assembly Line Balancing Problems are considered to be those that take into account other attributes and system restrictions. A great diversity of GALBP has been considered in the literature, which include, for example, mixed-models, parallel workstations, U-Shaped lines, unequally equipped workstation and multiple objectives (see, for example, Becker and Scholl (2006)).

A common feature of most assembly line balancing problems is that they consider a unique and predetermined precedence graph that represents all possible precedence relations among the tasks. However, in real-life problems, several parts of an assembly process can admit alternative precedence subgraphs that represent their corresponding assembly variants. This is true in the assembly or disassembly of many industrial products for which several valid plans may be available. Examples of this situation include car assembling (Scholl *et al.* (2007)), the decoration of motorbike fairings (Capacho and Pastor (2005)), the production of commercial hand-held drills (Senin *et al.* (2000)), the manufacturing of toys from moulded plastic parts or by metal stamping (Das and Nagendra (1997)) or in the disassembly process of complex products (Gungor and Gupta, 1997).

Alternatives have essentially been a primary concern for the planning process and, due to its importance, several approaches have been proposed to integrate this strategic task into the balancing process (e.g., Tseng and Tang (2006), Gaalman *et al.* (1999)).

The huge complexity of problems involving assembly alternatives has led to the use of a two-stage based approach. In the initial stage, the system designer selects one of the possible variants according to criteria such as total processing time, cost, resource allocation, and task parallelism (e.g., Lambert (2006) and Senin *et al.* (2000)). Once the assembly alternatives have been selected, and a precedence graph is available (i.e. the assembly planning problem has been already solved), the line is then balanced in the second stage.

However, by following this two-stage procedure it cannot be guaranteed that an optimal solution of the global problem can be obtained, because the decisions taken by the system designer restrict the problem and cause information loss; i.e., a priori selection of an alternative leaves the effects of the other possibilities unexplored. For instance, if the system designer uses total processing time as decision criterion, the alternative with largest total processing time will be discarded notwithstanding it may provide the best solution of the problem (i.e., it requires the minimum number of workstations or minimum cycle time).

Therefore, it seems reasonable to consider that to solve efficiently an ALBP that involves processing alternatives all possibilities must be considered within the balancing process. For this purpose, in this thesis both the variant selection problem and the balancing problem are jointly considered instead of independently.

The *Alternative Subgraphs Assembly Line Balancing Problem* (ASALBP), the new problem firstly introduced, defined and studied in this doctoral thesis, considers the possibility of alternative assembly variants. Each variant is represented by a subgraph which determines the required assembly tasks, their precedence relations and their processing times. In this way, the SALBP hypothesis which states that tasks must be processed only once is relaxed; i.e. a particular set of tasks is performed only if the assembly process to which the tasks belong to is selected.

Therefore, apart from considering cycle time restrictions, subgraph constraints have to be taken into account to assure that tasks belonging to a particular subassembly are processed considering a unique assembly subgraph (i.e., the same assembly variant). Furthermore, it is also considered that task processing times may not be fixed, yet all known, but dependent on the assembly subgraphs. Therefore, total processing time may vary from one processing alternative to another.

A premise embraced by the problem addressed in this doctoral thesis considers that better solutions can be obtained when all available assembly variants are taken into account in the balancing process, rather than when selecting a priory an assembly alternative, and then balancing the line considering only the selected alternative. Therefore, solving the Alternative Subgraphs Assembly Line Balancing Problem implies simultaneously solving both the decision problem, to select one assembly subgraph for each subassembly that allows alternatives, and the balancing problem, to assign the tasks to the workstations.

Considering alternative precedence subgraphs imposes a higher level of difficulty on an assembly line balancing problem as it is verified the *NP-hard* condition of the problem -given that the simple case (SALBP) is *NP-hard* (see e.g. Wee and Magazine (1982)). However, as real industrial processes may involve assembly alternatives, the possibility of considering alternative subgraphs not only enables more practical and realistic instances of ALBP to be addressed, but also may favour an assignation of tasks to the workstations in order to optimize a given objective. Regarding the conventional terminology (e.g. Baybars (1986) or Scholl (1999)), when the objective is to minimize the number of workstations given an upper bound on the cycle time, the problem is referred to as ASALBP-1. If the objective is to minimize the cycle time given the number of workstations, the problem is called ASALBP-2.

1.2 Objectives

This doctoral thesis addresses a new assembly line balancing problem that has not been previously considered in the literature. Therefore, the core objectives of this work are to define, to formalize and to solve this complex problem.

In order to accomplish the main objectives, the following specific objectives are considered.

1. **State of the Art** of assembly systems focusing on problems considering processing alternatives.

2. **Definition and characterization of a new assembly line balancing problem**: the Alternatives Subgraphs Assembly Line Balancing Problem (ASALBP). This problem is defined and characterized, giving some numerical examples to illustrate its relevance.

3. **Mathematical Formulation of the ASALBP**. In order to formalize this new problem, two different mathematical programming formulations are developed. Such models are used to optimally solve small- and medium-scale ASALBP instances.

4. **Design and Implementation of Approximate Procedures**. The NP-hard condition of the ASALBP limits the potential of mathematical programming models when industrial size problems are considered. In order to deal with large-scale problems, a heuristic approach based on constructive procedures is considered. Furthermore, several local optimization procedures based on two different neighbourhood search strategies are developed.

5. **Benchmark generation**. Since the ASALBP is a new assembly line balancing problem, benchmark problems must be generated.

6. **Evaluation and Comparison of the Performance of the Developed Solution Procedures.** In order to evaluate the performance of the proposed mathematical models and the solution procedures, a computational experiment is designed based on the sets of benchmark problems generated in this thesis. All procedures are applied to small-, medium- and large-scaled problems instances. Conclusions are drawn from this evaluation as well as proposals for future research work.

1.3 Structure of the Thesis

This thesis consists of six chapters and is structured as follows.

Chapter 1 introduces the problem addressed in this thesis and outlines the aims of this work.

Chapter 2 presents the State-of-the-art. It discusses the main concepts related to assembly systems and gives an overview of the problems that have been addressed in literatures studies, including the proposed solutions procedures. Combinatorial optimization problems that involve assembly alternatives are also discussed in this chapter.

Chapter 3 introduces, defines and characterizes *the Alternative Subgraphs Assembly Line balancing problem* (ASALBP). Furthermore, some examples are provided in order to illustrate the benefits that can be obtained by considering assembly alternatives in the balancing process. The *S-Graph,* a diagramming tool proposed to depict all assembly alternatives in a unique precedence graph, is introduced here.

Chapter 4 presents the mathematical formulation of *the Alternative Subgraphs Assembly Line balancing problem.* Two mathematical programming models are proposed, and their performance is evaluated by using the IPL solver CPLEX© (a commercial optimization software).

Chapter 5 deals with the approximate approach. This chapter describes both the heuristics methods and the local optimizations procedures proposed to solve the ASALBP. The computational experiment carried out to evaluate and compare the proposed methods is also described here.

Finally, **Chapter 6** presents the conclusions and further research proposals.

Chapter 2

State of the Art

2.1 Introduction

This chapter introduces the basic concepts and criterions habitually used in the literature to classify assembly lines. It describes classical assembly line balancing problems and presents some classification schemes that have been proposed for problem identification. Furthermore, it gives an overview of the variety of problems and solutions procedures that have been considered in research studies. Finally, some optimization problems involving alternative configurations are presented in order to outline the problem under study in this doctoral thesis.

2.2 Assembly Lines

In its basic form, an assembly line consists of a sequence of m workstations, usually connected by transportation mechanism such as a conveyor belt, through which the product units flow. Each workstation repeatedly performs a set of tasks in order to produced or manufacture a specific product. Tasks require certain time to be processed and are related amongst one another according to the existing technological constraints.

Undoubtedly, the most famous example of an assembly line is the production plant of Henry Ford. T-model components were manufactured in the first moving line using the ideas of work division to decrease the production cost per unit and to allow massive production. However, the work division ideas and this kind of configurations date from much earlier times. The Venetian Arsenal (considered the world first factory) for instance, developed methods of mass-producing warships which were much faster and required less wood. At the peak of its efficiency in the early 16th century, the Arsenal was able to produce nearly one ship per day on a production-line basis not seen again until the Industrial Revolution. In 1799, Eli Whitney introduced the assembly lines in the American manufacturing system. In 1901 Ransom Eli Olds patented the first assembly line concept and his Olds Motor Vehicle Company was the first factory in America to mass-produce automobiles (Wikipedia (2003)). Was until 1913 when Henry Ford perfected the assembly line concept; nowadays, the first assembly line for building cars is attributed to him.

Although, assembly lines are most commonly found in the automotive industry, many other sectors are also organized in assembly lines. This is the case for most daily life goods, as, for example, the final assembly of electrical products such as coffee machines, washing machines, refrigerators, radio, TV and personal computers (Amen (2001)). More recently, assembly lines have gained importance in low volume production of customized products (Scholl *et al.* (2007)) as well as in service systems.

2.2.1 Basic Concepts

- **Processing tasks**: a processing task i (task, hereafter) is an indivisible working unit which has associated a processing time t_i. The total work required to manufacture a product in an assembly line is divided into a set of n tasks.
- **Workstations**: are the line component where tasks are processed, and can involve a human or robotic operator, certain equipment and some specialized processing mechanisms.

- **Cycle time** ct**:** is the time available in each workstation to complete the tasks required to process a unit of product -the production rate is equal to $1/ct$ units of product per time unit. The cycle time is also defined (e.g. Peeters (2006)) as the time interval between the processing of two consecutive units.
- **Precedence relations**: are defined by the technological precedence requirements that determine the partial order in which tasks can be performed in the assembly line. A task cannot be processed until all its immediate predecessors have already been processed. Precedence relations are normally represented by a precedence diagram.
- **Workstation load** S_j: is the subset of tasks assigned to workstation j.
- **Workstation time** $t(S_j)$**:** is the sum of the times t_i of all tasks assigned to workstation j.

$$t(S_j) = \sum_{i \in S_j} t_i \qquad [2.1]$$

- **Workstation idle time** It_j: is the difference between the cycle time and the workstation load.

$$It_j = ct - t(S_j),\ t(S_j) < ct \qquad [2.2]$$

- **Line balancing**: is the process of distributing the n tasks among the m workstations in such a way that precedence constraints and other constraints are satisfied; aiming at optimizing a given efficiency measure. Classical objectives seek to minimize m for a desired cycle time ct, or to minimize ct given m.

There exists a great variety of configurations involving assembly lines, which are characterized according to diverse criteria. Amongst others, these include the layout and shape of the line, the number of products and models being processed in the line, types of workstation and the variability of the task processing times.

Based on the research studies of Boysen *et al.* (2007a, 2007b), Becker and Scholl (2006), Hao (2005), Miralles (2004), Rekiek (2001) and Scholl (1999), the following classification (section 2.2.2) summarizes some of the most relevant attributes of assembly lines.

2.2.2 Classification of Assembly Lines

What follows classifies assembly lines according to: the number of products or models produced, tasks durations, shape or layout of the line, the flow of the workpieces and the level of automation of the line.

According to the number of products or models

- **Single-model line**: is the classical configuration in which a single model of a unique product type is produced (Figure 2.1).

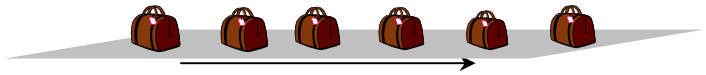

Fig. 2.1: Single-model line

- **Mixed-model line**: several variants of a basic product, referred to as models, are produced simultaneously in the line (see Figure 2.2). The production process does not involve setup times since all models require basically the same manufacturing tasks. Units of different models are produced in a mixed sequence.

Fig. 2.2: Mixed-model line

- **Multi-model line**: different models with significant differences amongst one another are processed in the line. Therefore, sequences of batches are processed, containing either the same model or a group of similar models, involving intermediate setup tasks (Figure 2.3).

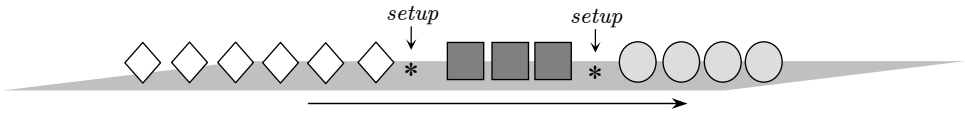

Fig. 2.3: Multi-model line

According to task durations

- **Deterministic**: all task processing times are fixed and known with certainty.
- **Stochastic line**: task processing times may be significantly affected from different sources of variability such as, for example, the ability or motivation of human operators. Therefore, the processing time of one or more tasks is considered to be probabilistic.

- **Dependent line**: tasks processing times are not fixed but dependent, for example, on the type of workstation to which the task is assigned, on the operator or on the processing sequence.
- **Dynamic line**: processing times vary over time and can be reduced in successive cycles due to improvements in the assembly process or due to learning effects (for example, when operators become familiar with the tasks).

According to the line shape or layout

- **Serial lines**: products units are processed throughout a group of workstations that are consecutively arranged in a straight line such as, for example, a conveyor belt (Figure 2.4).

Fig. 2.4: Serial line[1]

- **Two-sided lines**: consist of two serial lines in parallel, in which pairs of opposite workstations (left-hand side and right-hand side) process simultaneously the same workpiece. This configuration is commonly found in the automotive industry (Figure 2.5). Some tasks can be assigned only to one side (e.g. mount the left car wheel), some tasks can be assigned to either side (e.g. install the hood ornament), and some tasks must be assigned to both sides of the line simultaneously (e.g. install the rear seat) Bartholdi (1993).

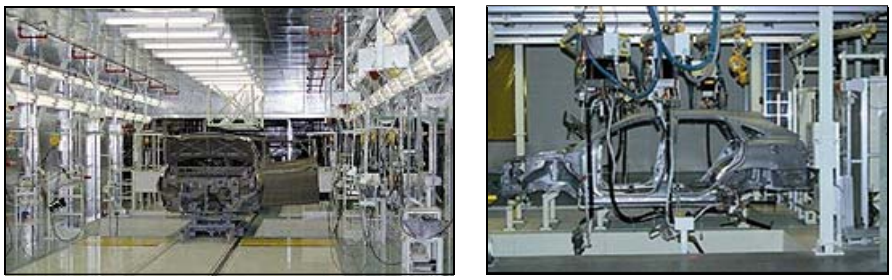

Fig. 2.5: Two-sided lines[2]

[1] An assembly line of VCR units of Sony.

- **Parallel workstations**: in this case two o more workstations are put in parallel; hence, the work pieces can be distributed between several workstations that perform an identical set of tasks.
- **Parallel lines**: this type of configuration can be considered when the production system involves multiple products, in which each line is designed for one product or family of similar products (Figure 2.6).

Fig. 2.6: Parallel lines

- **U-Shaped lines**: the workstations are arranges in a U-shaped line. Both tops of the line are closed to each other forming a U (Figure 2.7, Lee (2000)). The workstations may work during the same cycle on two or more workpieces at different positions on the line.

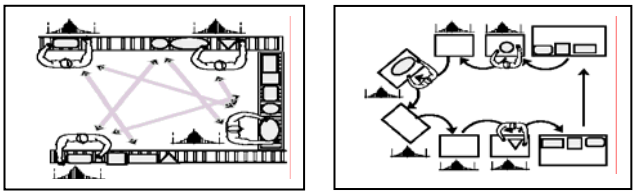

Fig. 2.7: U-shape lines

- **Circle/closed lines**: in this type of lines the workstations are arranged around a circular conveyor belt (or similar mechanism), as can be seen in Figure 2.8. A workpiece moves around being processed as it visits the workstations, until the last task have been performed.

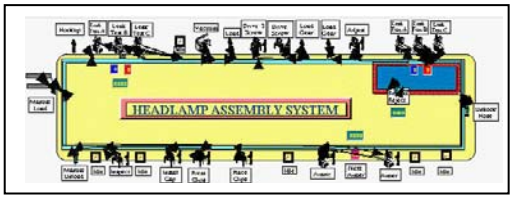

Fig. 2.8: Closed line

[2] The assembly line of the Toyota Lexus, Canada.

According to the workpieces flow

- **Synchronous lines**: in synchronous lines, also referred to as paced lines, all workstations have a common cycle time. Therefore, all workstations start processing at the same time and advance the workpieces simultaneously. Synchronous lines have a fixed production rate (Figure 2.9).

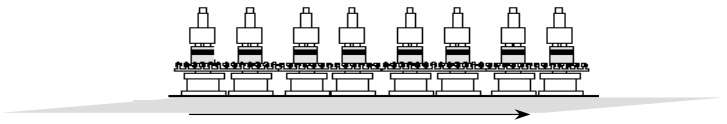

Fig. 2.9: Synchronous line

- **Asynchronous lines**: in these lines all workstations can work at different speeds; thus, workpieces are transferred whenever the required tasks are completed. The workstations are linked by buffers to store the workpieces that cannot advance to the next workstation due to it is processing another workpiece (Figure 2.10).

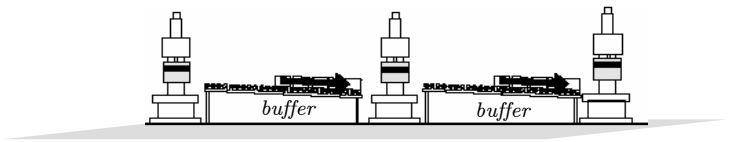

Fig. 2.10: Asynchronous line

- **Feeder lines**: feeder lines are supplementary lines that provide a main line with subassemblies. Figure 2.11 shows an example of an assembly process of an aeroplane, which consists of four lines feeding the main line.

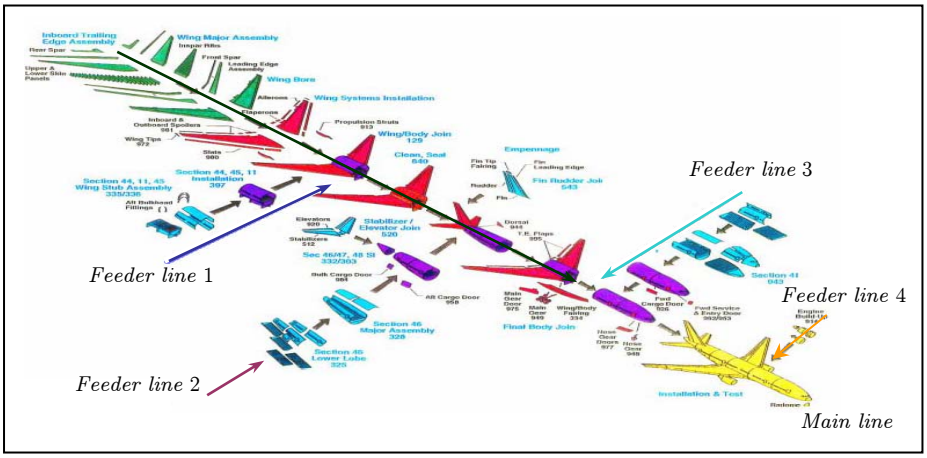

Fig. 2.11: Feeder lines

According to the level of automation

- **Manual lines**: in manual lines the tasks are performed by human operators. These lines are common when workpieces are fragile or are of special importance. Harley Davidson's motorcycles, for example, are 100% assembled by hand as shown in Figure 2.12.
- **Robotic lines**: robotic lines, commonplace in automotive industry, are lines fully automated and operated by robots (Figure 2.13).

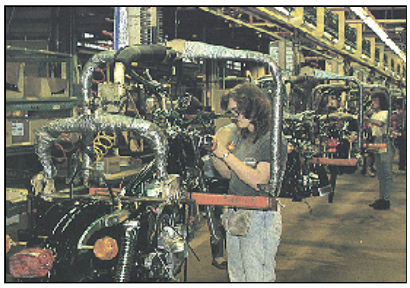

Fig. 2.12: Manual line

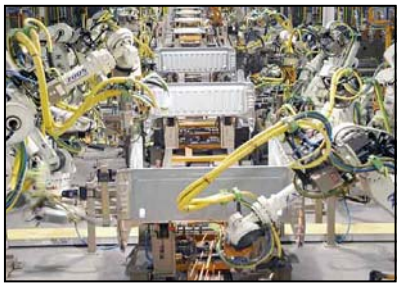

Fig. 2.13: Robotic line[3]

The characterization of the line, to a great extent, determines the type of balancing problem that is to be solved. For example a single-model, serial, paced, deterministic line merely entails the assignment of tasks to the workstations –the simplest balancing problem. However, for other line configurations the balancing problem comes together with additional decision problems. A mixed model line, for example, is connected with a problem of sequencing the models, whereas a multi-model line also implies a lot sizing problem. Parallel lines involve a decision problem concerning the number of lines that needs to be installed. Robotic lines, on the other hand, involve the assignment of both tasks and robots to the workstations. Asynchronous lines requires of the positioning and dimensioning of buffers; and whenever feeder lines are considered, the production rates of the available lines have to be synchronized.

It is evident that industrial systems involve a great variety of characteristics and problem variations. However, due to their complexity, most literature studies on production and manufacturing have addressed problems which do not consider many of the requirements and constraints present in real systems. Despite that, in the last years a considerable effort has been done towards

[3] http://encarta.msn.com/media˙701765960/Robot˙Assembly˙Line.html (visited on February 2004)

filling the gap between problems addressed in research works and real-world applications. What follows discusses main aspects of assembly line balancing problems, its variations and proposed solution procedures. Furthermore, problems involving processing alternatives are also discussed.

2.3 Assembly Line Balancing Problems

Assembly line balancing problems have been extensively studied, as can be seen in the reviews of Baybars (1986), Ghosh and Gagnon (1989), Erel and Sarin (1998), Rekiek *et al.* (2002), Dolgui (2006), Becker and Scholl (2006), Scholl and Becker (2006).

As previously mentioned, the Assembly Line Balancing Problem (ALBP) consists in assigning a set of indivisible tasks to an ordered sequence of workstations in such a way that precedence constraints are maintained, the workload of each workstation does not exceed the cycle time and a given efficiency measure is optimized. The term *balancing* arises from the fact that the workload of each workstation is to be balanced (Rekiek (2001)). Since a *perfect balance* (i.e., an identical load for all workstations) is rarely achieved, workstations idled time becomes a main optimization objective. On the other hand, as the assembly line global cost is influenced by the number of workstations, the classical objective of assembly line balancing problems is to minimize the number of workstations for a given cycle time, which is referred to as *time-oriented line balancing* (e.g. Amen (2001)). Furthermore, production rate can be maximized by minimizing the cycle time of a given number of workstations. Problems that seek to minimize costs are regarded to as *cost-oriented line balancing* (e.g. Becker and Scholl (2006), Scholl and Becker (2006), Amen (2000)). On the other hand, *profit-oriented* are those which implicitly consider the profit attained by the line.

Generally, minimizing the number of workstations or the cycle time is the primary objective of assembly line balancing problems. Nevertheless, most often, more that one efficiency measure is to be optimized. The followings are some of the minimization criteria that have been considered in literature studies: throughput time (i.e., the time interval between lunching a workpiece and finishing the finished product form the line), cost of machinery and tools (e.g. Bukchin and Tzur (2000)), inventory cost (e.g. Martin (1994)), dead time (i.e. the time that takes to transport a workpiece from one workstation to another) (e.g. Bard (1989)), cost of producing one unit of product, labour cost (e.g. Pinto *et al.* (1981)), number of buffers, line stoppage time, and variances in workstation times. Some maximization objectives include production rate

(which is equivalent to minimize the cycle time), line efficiency and profit (e.g. Becker and Scholl (2006)). A further objective considers that the workload of each workstation needs to be as similar as possible (e.g. Martinez and Duff (2004), Miralles *et al.* (2003)).

Considering the line system characteristics and the problem objectives, several attempts have been done to categorize balancing problems.

A well-known early classification of ALBP is the one proposed by Baybars (1986), which distinguish two classic problems: Simple Assembly Line Balancing Problem (SALBP) and Generalized Assembly Line Balancing Problem (GALBP). In the former case, only one model of a single product is processed, and the problem is restricted by precedence relations and cycle time constraints. GALBP, on the other hand, compounds all problem variations which take into account further restrictions and problem attributes. Ghosh and Gagnon (1989) slightly extended the Baybar's proposal by considering the number of products being processed in the line and the variability of the task processing times.

Notwithstanding the classification proposed by Baybars (1986) has been habitually used as a guideline for many other proposals, it is yet too general and restricted to reflect the increasing variety of real-word balancing problems. Consequently, more detailed classifications schemes have been intended to facilitate the communication between researches and practitioners. Such proposals use a condensed notation which allows considering a significant number of aspects to describe real assembly systems. Some of the most relevant proposals include the following.

Plans (1999) presented in his doctoral thesis an exhaustive classification scheme based in a five-field codification to identify and characterize assembly balancing problems as well as its resolution procedures. The first field of such codification is used to characterize the line: identifies the type of line considered (i.e. simple, mixed or multiple) and defines the existence of parallel workstations or buffers. The second field specifies tasks durations, setup times and, when applies, operator transportation times. The third field specifies the constraints among the tasks (i.e. precedence, incompatibility, affinity or parallelism) and indicates whether or not all workstations are equally equipped. The fourth element indicates if movement of the product being assembled is allowed over the line (e.g. rotated); and the last field specifies the problem type and the optimization objectives.

A similar classification scheme is proposed by Hao (2005), in which a larger number of the characteristics of the problem being studied are taken into

account. In this case, ALBP are classified considering four main groups: (1) the product, which defines the range of products processed over the line, its launching discipline and its position while being processed; (2) the line, which defines the line layout, the type of workstations used, the degree of automation, its length, type of setups, and the pattern (related to the speed of the line and the allowance to stop the line processing when required) used to manage the line; (3) the operator, which describes the people capabilities to perform the tasks over the line; and the last group (4) defines the type of problem and its objectives.

More recently, Boysen *et al.* (2007b) proposed an approach intended to typify extensions of assembly systems by considering the following tripartite notation: $[\alpha|\beta|\gamma]$. The first element, α, uses a set of six attributes to determine whether a unique product or model is being considered, to establish the structure of the precedence graph, to identify processing times, assignment restrictions and to establish whether there exist processing alternatives. The second element, β, describes the workstations and the line: it defines the movement of the workpieces, the line layout, level of (line, workstation, tasks, and working places) parallelization, resource assignment restrictions, and other configuration aspects, such as buffers or feeders. Finally, γ establishes the objectives.

2.3.1 Simple Assembly Line Balancing Problems (SALBP)

As previously mentioned, SALBP consider very simple problems, entirely restricted by the technological precedence relations and the cycle time constraints.

A huge amount of research work has been devoted to this type of problem (e.g. Baybars (1986), Ghosh and Gagnon (1989), Scholl (1999) and Becker and Scholl (2006)).

Characteristics of the simple assembly line balancing problem

The following are the main assumptions of simple assembly line balancing problems (Baybars (1986), Scholl (1999)).

A serial assembly line processes a unique model of a single product with all input parameters known with certainty. Task processing times are deterministic and independent on the workstation at which they are performed and on the preceding or following tasks. None of the task processing times is greater than the cycle time and setup times are considered to be negligible. All

workstations are equally equipped and manned, therefore, any workstation can process (one at a time) any one of the tasks; furthermore, tasks can be assigned to any workstation, and they are not incompatible between each other. On the other hand, tasks must be process only once and cannot be split among workstations; therefore, each task has to be completely processed in one workstation only. Task cannot be processed in arbitrary sequences due to technological precedence requirements; though all must be processed; and no other assignments restrictions are considered apart from precedence cycle time constraints.

Versions of SALBP

According to the optimization objective considered, four versions of SALBP are distinguished (Scholl (1999)):

- **SALBP-1**: minimizes the number of workstations m given a cycle time ct.
- **SALBP-2**: aims at minimizing the cycle time ct given the number of workstations m.
- **SALBP-E**: seeks to maximize the line efficiency E, where $E=tsum/(m \cdot ct)$ and $tsum$ is the summation of all task processing times.
- **SALBP-F**: is a feasibility problem that tries to establish whether a feasible task assignment exists for a given cycle time ct and a number of workstations m.

Although the great majority of published research work done on SALBP focuses on SALBP-1, it has been argued (e.g. Miralles (2004)) that SALBP-2 appears to be more relevant than its counterpart SALBP-1, because SALBP-1 is suitable only when designing an assembly line and SALBP-2 appears every time an existing line requires to be (re)balanced.

Mathematical model of SALBP

There exist several mathematical formulations for the simple assembly line balancing problem, in particular for SALBP-1, which have been used as a reference to many other models. According to Ghosh and Gagnon (1989), the first analytical statement of this problem was made by Helgeson *et al.* in 1954 and published for the first time in mathematical form by Salveson in 1955. Other models include the one proposed by Bowman (1960) who was the first to incorporate integer variables. The model of Bowman was improved by White (1961) and then further improved by Thangavelu and Shety (1971), and Patterson and Albracht (1975). What follows present a basic mathematical programming model for SALBP-1 and for SALBP-2.

Notation

- **Indices**
  - i for tasks
  - j for workstations
- **Parameters**
  - n number of tasks ($i = 1,\ldots,n$)
  - m maximum number of workstations ($j = 1,\ldots,m$)
  - t_i processing time of task i ($i = 1,\ldots,n$)
  - ct cycle time (it is given for SALBP-1 and a decision variable for SALBP-2)
  - PD_i set of the immediate predecessors of task i ($i = 1,\ldots,n$)
- **Decision variables**

$$y_j \in \{0,1\} = \begin{cases} 1, & \text{if there is any task assigned to workstation } j \ (j = 1,\ldots,m) \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ij} \in \{0,1\} = \begin{cases} 1, & \text{if task } i \text{ is assigned to workstation } j \ (i = 1,\ldots,n;\ j = 1,\ldots,m) \\ 0, & \text{otherwise} \end{cases}$$

Mathematical Model of SALBP-1

The following model for SALBP-1 assigns the tasks to the workstations in order to minimize the number of workstations given the cycle time ct.

The objective function [2.1] consists in minimizing the number of workstations. Constraints [2.2] guarantee that every task i is assigned to one and only one workstation. Constraints [2.3] ensure that the summation of the processing times of the tasks assigned to workstation j does not exceed the cycle time. Constraints [2.4] impose the precedence constraints. Relations [2.5] oblige the workstations to be used in lexicographic order (i.e., tasks are assigned from the first to the last workstation).

$$Minimize\ z = \sum_{j=1}^{m} y_j \qquad [2.1]$$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i \qquad [2.2]$$

$$\sum_{i=1}^{n} t_i \cdot x_{ij} \le ct \cdot y_j \qquad \forall j \qquad [2.3]$$

$$\sum_{j=1}^{m} j \cdot x_{pj} \le \sum_{j=1}^{m} j \cdot x_{ij} \qquad \forall i, \forall p \in PD_i \qquad [2.4]$$

$$y_j \ge y_{j+1} \qquad j = 1,\ldots,m-1 \qquad [2.5]$$

Mathematical Model of SALBP-2

The formulation for SALBP-2 is similar to the previous formulation for SALBP-1 in which the cycle time is the variable to be optimized, i.e., objective function [2.6]. Furthermore, as the number of workstations is a given parameter, constraint [2.3] is replaced by [2.7] since all workstations existence variables y_j are equal to 1.

$Minimize\ z = tc$		[2.6]
$\sum_{j=1}^{m} x_{ij} = 1$	$\forall i$	[2.2]
$\sum_{i=1}^{n} t_i \cdot x_{ij} \leq ct$	$\forall j$	[2.7]
$\sum_{j=1}^{m} j \cdot x_{pj} \leq \sum_{j=1}^{m} j \cdot x_{ij}$	$\forall i, \forall p \in PD_i$	[2.4]

2.3.2 Generalized Assembly Line Balancing Problems (GALBP)

Habitually, generalized assembly line balancing problems are considered to be the problems in which one or more assumptions of the simple case are relaxed (e.g. Baybars (1986), Scholl and Becker (2006)). Some common GALBP examples include the following main known groups.

U-Shaped Assembly Line Balancing Problem (UALBP)

This kind of problems involves U-shaped lines. This configuration is considered to be more flexible because the line disposition allows for more possibilities on how to assign tasks to workstations. The reason for this is that tasks can be assigned when either its predecessor or its successors have already been assigned, whereas with serial lines a task can be assigned only when its predecessors have been assigned. Regarding the conventional terminology used for SALBP (e.g., Baybars (1986)), the following variants are distinguished: UALBP-1, UALBP-2 and UALBP-E, respectively. Examples of this type of problems can be found in Scholl and Klein (1999b), Miltenburg (1998, 2002), Miltenburg and Wijngaard (1994), Ajenblit and Wainwright (1998).

Mixed-model Assembly Line Balancing Problem (MALBP)

This problem appears when a mixed-model line is considered. Different models of the same product are inter-mixed to be assembled on the same line. Therefore, apart from assigning the tasks to the workstations the sequence of different models has to be determined. The problem versions MALPB-1,

MALBP-2 and MALBP-E are also valid. Many literature studies have addressed this problem, see, for example, Kubiak and Suresh (1991), Bard *et al.* (1992), Bukchin (1998), Merengo *et al.* (1999), Bukchin *et al.* (2002), Karabati and Sayin (2003), Ponnambalam *et al.* (2003), Spina *et al.* (2003), Bukchin and Rabinowitch (2005).

***Robotic Assembly Line Balancing Problem* (*RALBP*)**

In this case, a robotic line is considered, therefore, both the assembly tasks and the set of robots have to be assigned to workstations (e.g. Rubinovitz and Bukchin (1993), Tsai and Yao (1993), Hong and Cho (1999)).

***Multi-objective Assembly Line Balancing Problem* (*MOALBP*)**

These problems consider several optimization objectives simultaneously. Agpak and Gokcen (2005), for example, deal with a problem that seeks to minimize both the number of workstations and the total assembling cost or the amount of resources. According to Rekiek *et al.* (2002) most GALBP are multi-objective (e.g. Kim *et al.* (1996), Malakooti and Kumar (1996), McMullen and Frazier (1998a), Bukchin and Masin (2004)).

Many other problems have been also addressed in the literature in which a great diversity of aspects of the real problem has been taken into account.

Regarding the characteristics of the line and the layout of the system

Bukchin and Rubinovitz (2003), for example, addressed a problem involving parallel workstations; multiple workstations are considered by Buxey (1974); Pinto *et al.* (1975) tackled a problem involving parallel tasks. Other problems include two-sided lines, commonly found when heavy work pieces such as cars or aeroplanes are involved (e.g. Kim *et al.* (2000), Bartholdi (1993)); buffered or parallel lines commonplace in a multi-model context (e.g. Suer (1998)), multi-product lines (e.g. Pastor *et al.* (2002), Berger *et al.* (1992)); multiple assembly lines as the N-UALBP of Miltenburg (1998); and complex layouts involving lines with different shapes (e.g. Bukchin *et al.* (2006)).

Additional restrictions

Research works have also addressed problems that consider additional restrictions apart from cycle time and precede constraints. Park *et al.* (1997), for example, considered a problem involving incompatibilities; therefore certain tasks cannot be processed in the same workstation. Other examples include workstation capacity constrained problems as in Moon *et al.* (2002); resource constrained (e.g. Agpak and Gokcen (2005)); and workstations that are not equally equipped (e.g. Nicosia *et al.* (2002)).

Task durations

With reference to duration of the tasks, literature studies include problems that involve processing times that are dependent on the sequence (e.g. Spina *et al.* (2003)) or on the operator (e.g. Corominas *et al.* (2006)), which are stochastic (e.g. Sarin *et al.* (1999)) or fuzzy (e.g. Gen *et al.* (1996)).

Processing alternatives and equipment selection

Alternatives configurations have also been considered in literature studies, which are mainly related to equipment selection. In this case, processing alternatives are determined through task requirements concerning either machines or manpower (e.g. Pinto *et al.* (1983), Sawik (2002), Agpak and Gokcen (2005) and Gamberini *et al.* (2005)).

Bukchin and Tzur (2000) addressed a problem that considers equipment alternatives, with every workstation provided with one equipment chosen from a set of equipment types. Each equipment type has an individual cost that affects task processing times. Therefore, the problem implies, on the one hand, the selection a proper equipment type to be assigned to workstations; and, on the other hand, the assignment of tasks considering workstation restrictions. In this problem tasks are subject to fixed precedence restrictions; in the same way, processing times are considered to be fixed. A similar problem related to equipment selection was undertaken by Bukchin and Rubinovitz (2003) which further considered parallel workstations.

Pinto *et al.* (1983) dealt with a problem involving processing alternatives. According to Bukchin and Tzur (2000), this problem is related to the selection of limited equipment, which may be added to the existing equipment in the workstation. In this problem precedence relations between tasks are always maintained.

Processing alternatives have also been considered in other optimization problems, such as the scheduling of tasks in flexible manufacturing systems. Ahn and Kusiak (1990), for example, dealt with a case in which the assumption that one process plan is available for each job is relaxed. They analyzed the effects of process plans on scheduling performance and concluded that the quality of schedules, regarding makespan and utilization of resources, improves when alternative processing plans are considered.

Sawik (2002) tackled a problem of balancing and scheduling several product types which are produced in a flexible assembly line; i.e., a line that involves workstations of various types in series, each one capable of simultaneously producing a mix of product types. Two solution approaches are proposed: (1)

is a sequential approach that at first assigns the tasks to the workstations regardless of the model type, and then afterwards determines the sequence for each product type; and (2) a monolithic approach that simultaneously considers the balancing and the scheduling problem. In both cases, each product must be successively routed to the workstations where the required tasks have been assigned subject to the precedence relations defined by its assembly plan, any of which is unique and prefixed beforehand.

The problem addressed in this doctoral thesis, *the Assembly Subgraphs Assembly Line Balancing Problem* (ASALBP), considers the possibility of assembly alternatives, each of which consists of a particular task processing order that is represented by a precedence subgraph. Consequently, precedence relations are not fixed but dependent on the assembly subgraphs. Furthermore, each processing alternative involves a subset of tasks which may be different for each assembly variant. In addition, task processing times are not fixed but are also dependent on the assembly subgraphs.

The development of this thesis engenders the definition and formalization of this new problem as exposed in the research works of Capacho and Pastor (2005, 2006). Per se, previous to this thesis, the ASALB problem remained unexplored.

In a recent work, Scholl *et al.* (2007) introduced the sequence-dependent assembly line balancing problem (SDALBP), which extends the basic problem by considering sequence-dependent task times. In that paper, the authors adapt solution approaches for SALBP to SDALBP, generate test data and perform some preliminary computational experiments. SDALBP can be considered a special case of ASALBP, in which assembly alternatives are represented by time increments that are added to the task time, defining the interference of performing one task after certain other task. For instance, an increment sd_{ij} corresponds to the additional time that a task j requires to be performed given that a task i has been performed before it.

In the ASALBP the alternatives are explicitly defined by independent precedence subgraphs which represent different processing alternatives; i.e. assembly variants. Therefore, the alternatives can be defined not only trough different task processing times but also trough completely different sets of precedence requirements. Furthermore, assembly processes involving different set of tasks are also allowed, which are not at all contemplated in the SDALB Problem.

2.4 Procedures to Solve Assembly Line Balancing Problems

Numerous procedures have been developed to solve assembly line balancing problems. Due to the NP-*hard* nature of this type of combinatorial problem, few exact methods have been developed to solve SALBP, in particularly SALBP-1. Habitually, although guaranteeing an optimum solution, exact methods have a problem size limitation, measured in terms of computing time; therefore, they can only be applied to problem instances with small or medium number of assembly tasks. Approximate methods (i.e., heuristics and metaheuristics) have been developed in order to overcome such a limitation, and aiming at providing good solutions that are as near as possible of the optimal solution.

Amongst the more relevant review papers concerning both exact and approximate procedures to solve assembly line balancing problems are the following proposals: Erel and Sarin (1998) and Baybars (1986) which present exact methods developed for the simple case (SALBP); Talbot *et al.* (1986), on the other hand, dealt with heuristics techniques. Scholl and Voss (1996) also discuss heuristic approaches focussed on SALBP. Branch and bound methods are compared by Scholl and Klein (1999a).

An analysis of the optimization methods for assembly lines design is provided by Rekiek *et al.* (2002). Erel and Sarin (1998) provide a survey on the procedures to solve ALBP. The most up to dated states of the art on both exact and heuristics methods can be found in Scholl and Becker (2006) for the simple case and in Becker and Scholl (2006) for the generalized case.

2.4.1 Exact Procedures

Generally, (*mixed*) *integer linear programming* models have been used to formally describe assembly line balancing problems, which may facilitate designers and decision makers to have a better understanding of different assembly systems. However, most often solving such models optimally has not practical relevance because standard solvers proved to be inefficient when considering real-world scaled problems (Scholl *et al.* (2007)). Therefore, most exact methods considered in the literature to solve ALBP are based on dynamic programming and branch-and-bound procedures.

Dynamic Programming (DP) procedures basically transform the problem into a multi-stage decision process by breaking it into smaller subproblems, which in turn are solved recursively; then the optimal solutions of the subproblems are used to construct the optimal solution of the original problem. The first dynamic programming procedure was developed by Jackson (1956) and modified by Held *et al.* (1963). The main drawback of these procedures is their large memory requirements. This limitation was improved by the procedures proposed by Schrage and Baker (1978), Lawler (1979) and Kao and Queyranne (1982). Although the latter DP proposals have resulted in greater computational efficiency, time and storage requirements continues to be a mayor inconvenient of this type of procedures.

Branch-and-bound (B&B) is an enumeration technique developed by Little *et al.* (1963), which finds the optimal solution by exploring subsets of feasible solutions. Sub-regions are formed by *branching* the solution space. A *bounding* process is recursively used to find lower or upper bounds of the optimal solution within each sub-region, using different searching strategies (e.g., depth first search, minimal lower bound, best first search or minimal local lower bound). Computational comparisons (e.g. Scholl and Klein (1999a)) have revealed that *branch-and-bound* (B&B) procedures outperform DP. B&B procedures are further discussed by Scholl (1999) and Becker and Scholl (2006). Pastor (1999) presents a classification of such procedures as well as different search and bounding strategies.

Some effective B&B methods developed to solve SALBP-1 include FABLE proposed by Johnson (1988), EUREKA by Hoffmann (1992), and SALOME of Scholl and Klein (1997). Similarly, contributions on *Branch-and-cut* algorithms are provided by Pinnoi and Wilhelm (1998) and Bockmayr and Pisaruk (2001). On the other hand, most exact methods used to solve SALBP-2 are based on repeatedly solving SALBP-F with m workstations and various trial cycle times values within a given interval (Klein and Scholl (1996)). Only two B&B procedures solve SALBP-2 directly: TBB2 and SALOME2 developed by Klein and Scholl (1996).

In a much lesser extent, mainly justified by their problem size limitation, exact methods have been also used to solve GALBP. Urban (1998), for example, presented an integer programming formulation for UALBP-1, solving problem instances with CPLEX (a commercial optimization software). Sarin *et al.* (1999) developed a B&B procedure for a problem with stochastic processing

times. Scholl and Klein (1999b) developed a B&B procedure to address an UALBP. Dynamic programming formulations, on the other hand, were proposed by Miltenburg (1998) to solve a case with N U-lines and by Nicosia *et al.* (2002) to solve a problem involving different workstations.

2.4.2 Approximate Procedures

There exist a great variety of approximate methods proposed in the literature to solve assembly line balancing problems (e.g. Talbot *et al.* (1986), Amen (2000, 2001), Scholl and Voss (1996)); most of which are constructive methods, enumeration procedures and improving techniques. Two main groups are distinguished: heuristic and metaheuristic methods.

Heuristic methods

A common methodology used is the *greedy* approach, where, at each step of the procedure, one element of the solution is chosen according to a given criteria until a complete solution is obtained. The simplest method randomly generates solutions, evaluates each one of them and keeps the best of all solutions obtained (Silver (2002)).

Basically, *constructive methods* are based on priority rules, most of which are measured considering the number of predecessors and successors, and the task processing times. One of the first proposed heuristic was *Ranked Positional Weight* (*RPW*) by Helgeson and Bernie (1961), in which tasks are ranked in descending order of the *positional weight* (the summation of the task time and the processing times of all its successors). Other well-known priority rules include maximum task time, maximum total number of successors, minimum earliest and latest workstation and minimum slack. Some heuristics combine several priority rules; such as, for example, *TTS* which considers the maximum task time divided by the total number of successors.

Most efficient priority rules are described in detail in Talbot *et al.* (1986), Hackman *et al.* (1989), Boctor (1995), Scholl (1999) and Gosh and Gagnon (1989).

Priority-rule based methods create a ranked list of the *assignable tasks*. A task is assignable if all of its predecessors have already been assigned and if its time plus the current workstation time does not exceed the cycle time. Then, tasks are selected and assigned to the workstations considering one of the two following strategies.

Station-oriented: this strategy starts with one workstation and then others are consecutively considered one at a time. In each iteration tasks are orderly selected from the ranked list and assigned to the current workstation. Once the current workstation is fully loaded (the ranked list is empty) a new workstation is opened.

Task-oriented: in this strategy, the first task in the rank list (the one with the highest priority) is selected and assigned to the earliest workstation to which the task can be assigned. Task-oriented methods are further divided into *immediate-update-first* or *general-first-fit methods* depending on whether the ranked list is immediately updated after a task has been assigned or after all tasks in the ranked list have already been assigned, respectively.

Computational experiments (e.g. Scholl and Voss (1996)) have shown that, in general, station-oriented provide better results than task-oriented methods.

Constructive methods that consider a unique rule to generate a single feasible solution are also regarded to as *single-pass* methods (Rekiek (2001)). Their counterpart, *multi-pass* methods (also called multi-start methods, e.g. Martí and Moreno (2003), Fernandes and Ribeiro (2005)) generate multiple feasible solutions, applying repeatedly different priority rules, returning the best of all solutions obtained when a stopping criterion is satisfied. In the latter case, a *random search* strategy must be also considered in which one of the assignable tasks is selected randomly, instead of selected the best considering a particular priority rule. Arcus (1966) proposed COMSOAL, the first multi-pass procedure applied to SALBP. In this procedure, the next task to be assigned to the current workstation is randomly selected from the set of assignable tasks; furthermore, it is considered that all tasks have the same probability of being selected. Several solutions are generated keeping the one with lowest level of idle time. Other procedures have been proposed based on the ideas behind COMSOAL, e.g. DePuy and Whitehouse (2000) for a resource allocation problem, and Dolgui *et al.* (2005) for transfer lines balancing.

Heuristics based on priority rules and enumeration procedures have also been proposed by Lapierre and Ruiz (2004) involving an industrial case.

The solution obtained by constructive methods can be improved by using *local search procedures.* These procedures start with a feasible solution which is progressively improved. Different strategies are used to generate neighbour solutions (e.g. two tasks are interchanged between each other) and then such solutions are evaluated based on a given objective. If one solution is better

than the current solution, it becomes the new solution and its neighbourhood is investigated until no further improvement can be obtained.

Metaheuristics

Falling in a local optimum is a main drawback of classical heuristic methods. Therefore, in the last years a group of methods, referred to as *metaheuristics*, have been developed to overcome such a limitation.

The term metaheuristic was first introduced by Glover (1996). These procedures are based in constructive methods to find an initial solution (or a population of initial solutions) and local search algorithms to move to an improved neighbour solution. In contrast to local search approaches, metaheuristics do not stop when no improving neighbour solutions can be found. They allow movements to worsening solutions in order to avoid premature convergence to a local optimum solution. Metaheuristics use different concepts derived from artificial intelligence, evolutionary algorithms inspired from mechanisms of natural evolution (Pierreval *et al.* 2003).

Further details on metaheuristics can be found in Reeves (1993, 1997), Osman and Laporte (1996) and Gottlieb *et al.* (2003). Most common metaheuristics include the following.

GRASP (*Greedy Randomized Adaptive Search Procedure*) is an iterative process in which each iteration consists of two phases: the construction phase, which generates an initial solution; and the improving phase, which uses a local optimization procedure to find a local optimum. The initial solution is generated by probabilistically selecting the next element to be incorporated in a partial solution from a *restricted candidate list* (RCL). The RCL is composed of the best elements considering a given greedy function (Armentano and Bassi (2006)). It has been proven (e.g. Feo *et al.* (1994), Festa and Resende (2004)) that GRASP produces good quality solutions for hard combinatorial optimization problems, including line balancing problems. Andres *et al.* (2006), for example, proposed a GRASP procedure to solve a balancing and scheduling problem considering sequence-dependent setup times.

Tabu search (TS) is a local search metaheuristic based on memory structures that prevents returning and keeping trap in a local optimum solution. To escape from a local optimum moves to worse solutions are allowed. A tabu list is used to avoid cycling back to recently visited solutions. The size of the list, a key parameter, determines the number of iterations during which a given solution is prevented to reoccur. The procedure finishes, for example, when a number of search movements has been performed and no further improvement

has been achieved. TS procedures have been proposed to solve assembly line balancing problems: SALBP-1 (e.g. Chiang (1998)), SALBP-2 (e.g. Scholl and Voss (1996)) and GALBP (e.g. Voss (1994), Pastor *et al.* (2002)). Further details on TS are found in Glover (1990) and Glover and Laguna (1993, 1997).

A*nt colonies algorithms*, first proposed by Dorigo *et al.* (1996), basically model the behaviour of ants searching an optimal path (e.g. for food or real ants) which connects two different positions (Gottlieb *et al.* (2003)). The selection of paths is stochastic and it is influenced by both the quantity of pheromone that other ants have put on a path (i.e. *desirability*) and the local values of the objective function that can be determined if the path is selected (i.e. *visibility*). The level of desirability is updated according to the paths that ants use the most (Pierreval *et al.* (2003)). Procedures based on ant colonies have been considered by Baykasoglu *et al.* (2003) and Bautista and Pereira (2002) to solve SALBP-1; and by Bautista and Pereira (2003) to solve an UALBP. McMullen and Tarasewich (2006) also considered ant colony optimization to address a multi-objective assembly line balancing problem.

Simulated Annealing (SA) is a technique inspired from the physical annealing of solids. It models how the molecular structure of metals is disordered at high temperatures and ordered and crystalline at low temperatures. A problem instance is formulated in such a way that it resembles disordered material. The temperature is gradually lowered such that ordered states correspond to good solutions of the problem. SA methods avoid getting trap in a local optimum by allowing *uphill* moves based on a model of the annealing process in the physical world (Flake (1999)).

SA algorithms applied to assembly line balancing problems include, for example, the proposal of Suresh and Sahu (1994) for solving a stochastic variant of SALBP-1, and the one by McMullen and Frazier (1998a) for a GALBP involving parallel stations, stochastic task times and multiple objectives.

Genetic algorithms (GA), an idea pioneered by John Holland, closely simulate biological evolution as they map programs and data into DNA-like structures that express some notion of fitness (Goldberg (1989)). GA use a set of initial solutions, i.e. individuals, each of which represents a point in the search space of potential solutions to a particular problem. A given number of individuals conforms a population of potential solutions. The population is evolved by employing *crossover* and *mutation* operators along with an objective function (i.e. the *fitness function*) that determines how likely individual are to be reproduced (Flake (1999)).

Genetic algorithms have been proposed, for example, by Ji *et al.* (2001) to determine the cycle time for printed circuit board assembly lines (SALBP-2); Ruvinovitz and Levitin (1995) to solve a RALBP; Kim *et al.* (1996) to solve a MOALBP; Ponnambalam *et al.* (2003) to solve a MALBP; and Feyzbakhsh and Matsui (1999) to optimal design flexible assembly systems.

Other approaches that have been also considered in research studies include the followings (see Pierreval *et al.* (2003) for more details on evolutionary algorithms applied to ALBP): *expert systems*, e.g. Phonganant *et al.* (2001) to solve a MALBP; and *fuzzy logic*, e.g. Gen *et al.* (1996). Erel and Gokcen (1999) also proposed a procedure using the *shortest route model* to solve a MALBP. Park *et al.* (1997) consider an algorithm based on *networks theory* to solve a problem with incompatibilities among the tasks. Tools for system modelling and analysis have been also used in combination with metaheuristics. For example, Mendes *et al.* (2005) use a simulated annealing procedure to derive configurations in a mixed-model assembly line, and then such configurations are fine-tune via a *simulation* model. McMullen and Frazier (1998b) use simulation as a mean to compare the results of applying different line balancing strategies considering paralleling of workers within work centres. Moberly and Wyman (1973) also use simulation to compare a set of assembly line balancing configurations.

Heuristics based on Petri nets have been also considered for solving assembly line balancing problems; Kilincci and Bayhan (2006, 2007), for example, proposed an algorithm based on Petri nets to solve SALBP-1.

Fluid models (eg. Avram *et al.* (1995)), on the other hand, have been proposed to analyze the behaviour of stochastic networks and large scale production systems involving a large number of tasks. Dai and Weiss (2002) and Weiss (1999), for example, considered a fluid approach for solving scheduling problems in manufacturing systems. In this type of problems it is assumed that work is composed of homogeneous fluid instead of discrete tasks, being the events occurring in the system associated with rate changes in fluid flows.

2.5 Conclusions

This doctoral thesis addresses a new generalized problem: the Assembly Subgraphs Assembly Line Balancing Problem (ASALBP). Such a problem considers the possibility of assembly alternatives, any of which consists of a particular task processing order and is represented by a precedence subgraph. Precedence relations are dependent on the subgraph selected, which, in turn, determines task processing times. Furthermore, assembly variants may involve different and independent set of tasks that are executed only when the alternative which they belong to is selected.

A comprehensive literature review have been carried out, and after analysing research works concerning generalized assembly line balancing problems, the following conclusion can be drawn: the problem that considers assembly variants, which may involve different sets of tasks with processing times and precedence requirements dependent on the assembly alternatives, has not been addressed before.

It is important to mention that Pinto *et al.* (1983) commented about the possibility of having variable precedence relations: "*In practice it is possible that a particular processing alternative can change the nature of the precedence requirements such that the requirements for the replacing tasks are not the same as the union for the requirement of the replaced tasks... Such special situations are not dealt with here*" (p. 823). However, as stated, this possibility is neither formalized nor developed.

On the other hand, in a recent work Scholl *et al.* (2007) highlighted the importance of having flexible precedence constraints to describe assembly alternatives. They presented a special case of the ASALBP, in which assembly alternatives are represented by time increments that are added to the task time and which define the interference of performing one task after certain other task. Precedence constraints, however, are kept fixed.

Consequently, in this doctoral thesis this unedited problem (ASALBP) is defined and formalized via two mathematical models. Moreover, considering the simplicity of the constructive methods and the fact that they have been successfully applied to assembly line balancing problems, a significant number of constructive methods, based on an adaptation of well-known priority rules and random search strategies, are proposed here. Furthermore, since it has been proven that workstation-oriented methods perform better than task-oriented ones, the proposed constructive procedures follow that assignment approach.

Chapter 3

ASALBP: The Alternative Subgraphs Assembly Line Balancing Problem

3.1 Introduction

As previously mentioned, this doctoral thesis tackles a new generalized assembly line balancing problem, which has been entitled ASALBP: *the Alternative Subgraphs Assembly Line Balancing Problem*.

The novel characteristic of such a problem is that it considers the possibility of having alternative assembly variants (represented by different assembly subgraphs) which determine how assembly tasks are to be performed. In ASALBP assembly variants may be defined by different task processing times and by different task precedence relations. Furthermore, as industrial problems may involve different assembly processes, assembly variants may also be defined by different and mutually exclusive sets of tasks. Therefore, task processing times, the precedence relations of certain tasks, and the tasks themselves are considered to be dependent on the available assembly variants. Then, apart from the problem of assigning the tasks to the workstations, a decision problem needs to be solved in order to fully determine the assembly or manufacturing process; i.e., one subgraph has to be selected for each subassembly of the system that allows alternatives.

3.2 Definition of the Problem

The *Alternative Subgraphs Assembly Line Balancing Problem* can be stated as follows:

There exists a set of tasks for which several alternative assembly variants (also called assembly routes) are available; the tasks have to be assigned to a group of workstations. Each variant for each subassembly is represented by an individual subgraph, which determines the required assembly/manufacturing tasks (hence the assembly variants may be defined by different and mutually exclusive sets of tasks) and the precedence relations among them. Furthermore, task processing times are considered to be dependent on the assembly subgraph. Therefore, total processing time may vary from one assembly alternative to another.

Tasks processing times are generally considered to be fixed, however in many real applications this is not the case. For example, task times depend on the nature of the tasks, the skills of the operators and the reliability of the machines (Rekiek (2001)). Furthermore, the duration of a task can be determined by the complexity of performing a given task considering the current state of the system; i.e., it depends on the processing sequence. For example, it gets more difficult (it requires more time) to decorate the fairing of a motorbike after they have already been assembled onto the motorbike than when they are unassembled.

Taking these assumptions into account, two problems have to be solved simultaneously: the decision problem, to select one assembly subgraph for each subassembly that allows alternatives; and the balancing problem, to assign the tasks to the workstations.

Regarding conventional assembly line balancing terminology (see, for example, Baybars (1986) and Scholl (1999)), an ASALBP that aims to minimize the number of workstations for a given upper bound on cycle time is referred to as ASALBP-1. If the objective is to minimize the cycle time for a given number of workstations, the problem is called ASALBP-2.

According to the classification of assembly line balancing problems proposed by Boysen *et al.* (2007a, 2007b), ASALBP is identified as [$\mathrm{pa}^{\mathrm{subgraph}}$— —]; where $\mathrm{pa}^{\mathrm{subgraph}}$ characterizes the precedence graph and indicates that processing alternatives exist, which alter complete parts of the production process, so that whole subgraphs are substitutable.

The ASALB Problem contains the following main characteristics:

- The ASALBP considers a serial assembly line designed for a single model of a unique product, for which all alternative assembly variants are completely known in advanced.
- None of the task processing times are larger than the cycle time.
- Tasks have to be processed completely in one workstation only, i.e., they cannot be divided between workstations.
- Workstations can process only one task at a time.
- Tasks cannot be processed in an arbitrary order due to the existence of precedence constraints.
- Several sets of precedence constraints are available, instead of a unique one, which represent the precedence relations among the tasks of the available assembly subgraphs.
- All tasks belonging to a particular subgraph have to be performed according the specifications of the same assembly variant.
- Tasks processing times are dependent on the assembly subgraph selected, but independent on the workstation where they are processed.
- Setup times are considered to be negligible.
- All workstations are equally equipped and manned; therefore, any task can be assigned to any workstation.
- Tasks are not incompatible between each other; therefore, any combination of tasks can be assigned to any of the workstations.
- Tasks must be processed at most once. Therefore, only those tasks belonging to the selected assembly subgraphs (or those that do not allow alternatives) must be performed. The remaining tasks will not be considered in the assembly process and, therefore, will not be carried out.

The following example illustrates the ASALB Problem.

Example 3.1: the final phase in the process of assembling a motorbike

This example considers the final phase in the process of assembling a motorbike, which consists of three main sets of tasks: Z, which is the decoration of the motorbike's fairing (it involves several subtasks, such as sticking different colour stickers and text labels onto the fairing); J, which entails attaching the fairing to the motorbike; and K, which involves making the final adjustments.

These three sets of tasks can be processed in two different ways (see Figure 3.1), which determine two alternative assembly variants of this process.

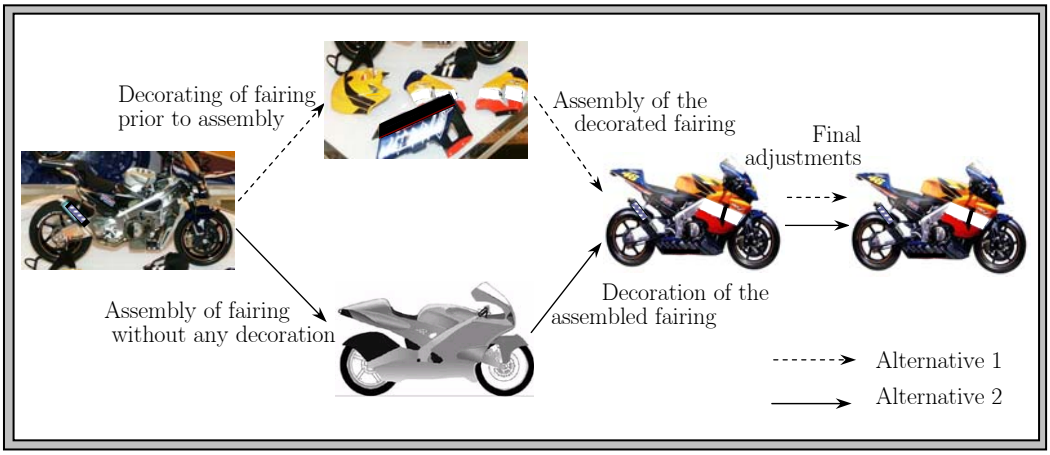

Figure 3.1: Final phase in the process of assembling a motorbike

- **Alternative 1** implies the decoration (task Z) of the unassembled fairing first, then attaching (task J) the fairing to the motorbike, and then making the final adjustments (task K).
- **Alternative 2** consists in assembling the fairing first (task J), then decorating (task Z) the fairing provided they have already been assembled onto the motorbike, and lastly making the final adjustments (task K).

As can be seen in Figure 3.2, each of these two assembly alternatives can be represented by using a standard precedence graph.

In this thesis, a precedence graph consists of nodes to represent the tasks required by each assembly alternative and connecting arcs which indicate the corresponding task precedence relations; furthermore, task processing times are represented as node weights.

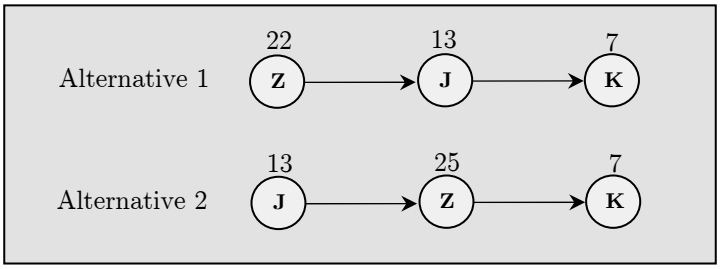

Figure 3.2: Assembly alternatives for the example of the motorbike

As can be observed in Figure 3.2, tasks Z and J allow two assembly alternatives whilst task K can be performed only after the execution of both Z and J have been completed.

Furthermore, tasks processing times are allowed to be dependent of the assembly alternatives. In this example, the processing time of task Z depends on the order in which it is processed: it requires 22 time units if it is performed before task J and 25 time units when it is performed afterwards (i.e., it takes longer time to decorate the fairing when they are already assembled). Task J, on the other hand, lasts 13 time units regardless of the assembly sequence. Task K always is processed at the end of the process and has a processing time of 7 time units.

Assembly alternatives can also be represented by using precedence subgraphs which gather the tasks processed according to the same assembly variant. Using the standard diagramming representation, it is not possible to depict alternative precedence subgraphs. In order to overcome the limitation of the standard precedence graphs, a diagramming tool, entitled *S-Graph* (discussed in detail in section 3.4), has been proposed to represent in a unique graph all available assembly alternatives. Figure 3.3 shows the *S-Graph* for the example of Figure 3.1.

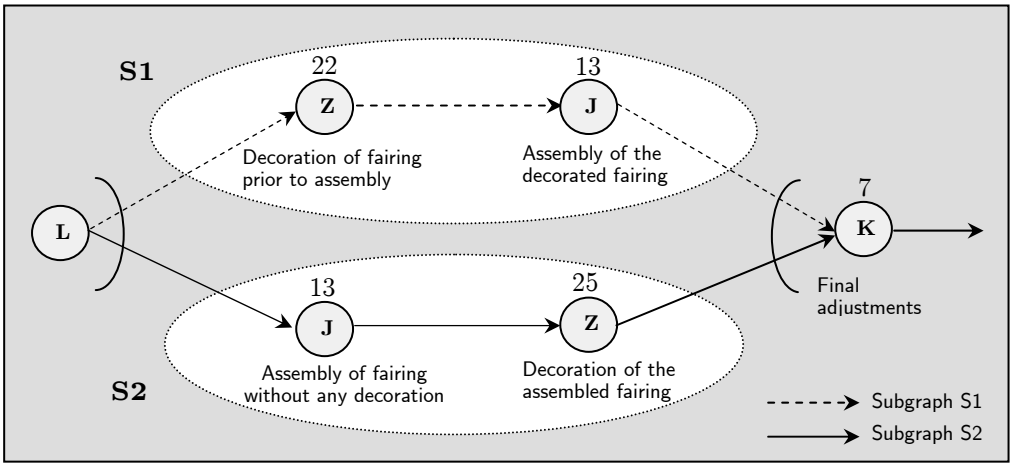

Figure 3.3: S-Graph of the final phase of the process of assembling a motorbike

As can be seen in Figure 3.3, there are two subgraphs representing the assembling alternatives for tasks Z and J: the first subgraph, S1, consists in performing task Z before task J which implies a total processing time of 35 time units; the second subgraph, S2, consists in performing task J first and then task Z with a total processing time equal to 38 time units.

It is valid to mention that task L of Figure 3.3 is considered to be a major task belonging to an intermediate phase in the process of assembling a motorbike.

As mentioned in the introduction, a two-stage procedure is normally used to solve a problem that involves assembly alternatives. In the initial stage, the system designer either decides, a priori, all the task durations (by fixing a precedence subgraph, which is equivalent to imposing additional precedence relations other than the existing technological ones), or selects (using a given criterion) one assembly subgraph from amongst the available alternatives. Once the alternative has been selected, the line is then balanced in a second stage considering that particular choice. By following such an approach, it cannot be guaranteed that the global problem can be solved optimally. However, better solutions can be obtained if the problem of selecting an assembly alternative and the balancing problem are solved simultaneously, rather than independently.

The following example helps, on the one hand, to clarify the ideas previously introduced and, on the other hand, to illustrate how the assignment of tasks to the workstations can be favoured by considering assembly alternatives.

Example 3.2: optimally solving ASALBP

This example considers again the aforementioned final phase of the process of assembling a motorbike (see Figure 3.1). Additionally, task Z, which consists of the decoration of the fairing, has been further divided into four subtasks (F, G, H and I) that involve fixing to the fairing different colour stickers and text labels. Table 3.1 shows the description of the disaggregated tasks, and, for each of the two resulting assembly alternatives, the task processing times, the task predecessors, and total processing time.

Table 3.1: Data for the example 3.2

Task		Alter. 1 (Subgraph S1)		Alter. 2 (Subgraph S2)	
		Processing time	Predecessors	Processing time	Predecessors
Z	F: Decoration of fairing with yellow stickers	5	L	6	J
	G: Decoration of fairing with blue stickers	5	L	7	J
	H: Decoration of fairing with text labels	8	L	8	J
	I: Decoration of fairing with black stickers	4	L	4	J
J	Assembly of fairing	13	F, G, H, I	13	L
K	Final adjustment	7	J	7	F, G, H, I
	Total processing time	42		45	

As can be seen in Table 3.1, some of the decorating tasks require longer processing times if they are performed on the attached fairing instead of on the unattached fairing.

By balancing each of the two resulting problems optimally, one for each alternative subgraph, and aiming to minimize the number of workstations, given a cycle time upper bound that is equal to 17 time units, the solutions presented in Table 3.2 are obtained. These results include task assignments (and workstation time), total processing time and number of workstations required per alternative.

Table 3.2: Results for ASALBP-1 for the example 3.2

Alternative subgraph	**Workstation load (workstation time)**				**Total processing time**	**Number of stations**
	I	**II**	**III**	**IV**		
1	F, H, I (17)	G (5)	J (13)	K (7)	42	4
2	J, I (17)	G, H (15)	F, K (13)	-	45	3

By following the argument on decision criteria used to select assembly variants discussed previously, it seems reasonable to consider S1 as a promising alternative for the assembly process because it entails less total processing time (42 time units), and would thus be chosen a priori over S2. However, as observed in Table 3.2, despite implying a greater total processing time (45 time units), S2 provides the best solution to the problem because it requires three workstations instead of the four required by S1. Therefore, if S1 had been selected a priori, then a better solution would have been discarded.

Similar results can be obtained for an ASALBP-2. Table 3.3 shows the results of optimally balancing the two resulting problems of Table 3.1 by considering three workstations. In this case, Alternative 2 provides the best solution to the problem since it requires a cycle time of 17 units instead of the 18 required by Alternative 1.

Table 3.3: Results for ASALBP-2 for the example 3.2

Alternative subgraph	**Workstation load (workstation time)**			**Total processing time**	**Cycle time**
	I	**II**	**III**		
1	F, G, H (18)	I, J (17)	K (7)	42	18
2	J, I (17)	F, G (13)	H, K (15)	45	17

The results previously obtained showed that considering alternative assembly variants may favour the assignment of tasks to the workstations which minimizes the number of workstations or the cycle time.

The balancing process may also be benefited even when the available assembly alternatives involve fixed task processing times (i.e., independent on the precedence subgraphs). This case is illustrated in the next example.

Example 3.3: solving optimally ASALBP with fixed times

Considering again the example of Figure 3.1 but now assuming that task processing times are independent on the tasks processing sequence (and equal to 5 units for both tasks F and task G).

Table 3.4 presents the results of optimally balancing each of the two ensuing problems and aiming at minimizing the number of workstations for a cycle time equal to 17 time units. Table 3.5, on the other hand, shows the results when the objective is to minimize the cycle time considering 3 workstations.

Table 3.4: Results for ASALBP-1 with Fixed Times

Alternative subgraph	**Workstation load (workstation time)**				**Total processing time**	**Number of workstations**
	I	**II**	**III**	**IV**		
1	F, H, I (17)	G (5)	J (13)	K (7)	42	4
2	J, I (17)	G, H (13)	F, K (12)	-	42	3

Table 3.5: Results for ASALBP-2 with Fixed Times

Alternative subgraph	**Workstation load (workstation time)**			**Total processing time**	**Cycle time**
	I	**II**	**III**		
1	F, G, H (18)	I, J (17)	K (7)	42	18
2	J, I (17)	F, G (10)	H, K (15)	42	17

As can be seen in Table 3.4 and Table 3.5, the possibility of having alternative assembly subgraphs may favour an assignation of tasks to workstations, even when the processing times are not dependent on the tasks processing sequence; i.e., independent on the assembly subgraphs.

Therefore, it can be expected that economical benefits can be achieved by simultaneously considering the decision problem that selects the assembly subgraphs, and the balancing problem that assigns the tasks to the workstations, underlining in this way the relevance of the ASALBP.

3.3 The *S-Graph*: a diagramming scheme to depict assembly alternatives

In this doctoral thesis a diagramming tool, which has been entitled *S-Graph*, has been proposed with the aim of representing in a unique graph all available assembly alternatives (i.e., precedence subgraphs), which cannot be depicted in a standard precedence graph.

Figure 3.4 shows the *S-Graph* for the example of the process of assembling a motorbike considering also the intermediate phase of such a process, which consists of attaching two parts of a piece, including the axle, to the motorbike's main body. The intermediate phase can be carried out in two different ways which are represented in the *S-Graph* by the subgraphs S3 and S4, respectively. The assembly alternatives for the final phase of the process of assembling a motorbike, as previously described, are represented by subgraphs S1 and S2.

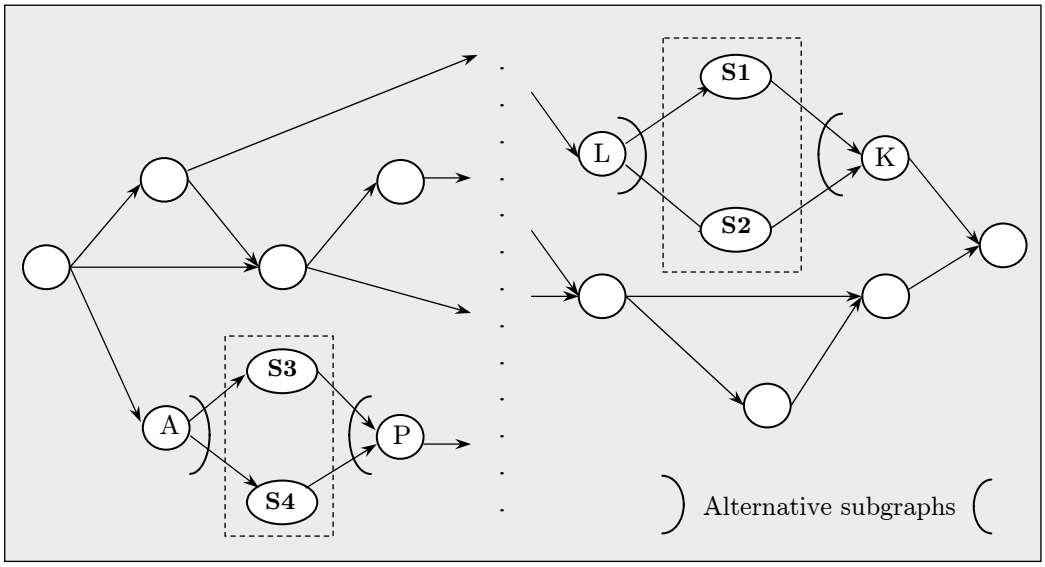

Figure 3.4: Precedence S-Graph for the assembly process of the motorbike

As can be seen in Figure 3.4, the assembly alternatives in the *S-Graph* are specified by the arcs entering or exiting the subgraphs, which are indicated by the semicircles drawn on the corresponding arcs. In this way, the *S-Graph* allows to represent, via individual subgraphs, assembly variants which imply different precedence requirements, different processing times and/or different sets of assembly tasks.

In order to make a more comprehensive definition of the *S-Graph* as an alternative precedence diagramming tool, two aspects need to be discussed.

On the one hand, it is assumed that assembly alternatives do not overlap between each other; therefore, each alternative for each available subassembly is represented by a unique and independent precedence subgraph. On the other hand, fictitious tasks, with nil processing time, are used to facilitate the representation of two subassemblies with processing alternatives that are consecutive (this case is represented in Figure 3.5 by the fictitious task α).

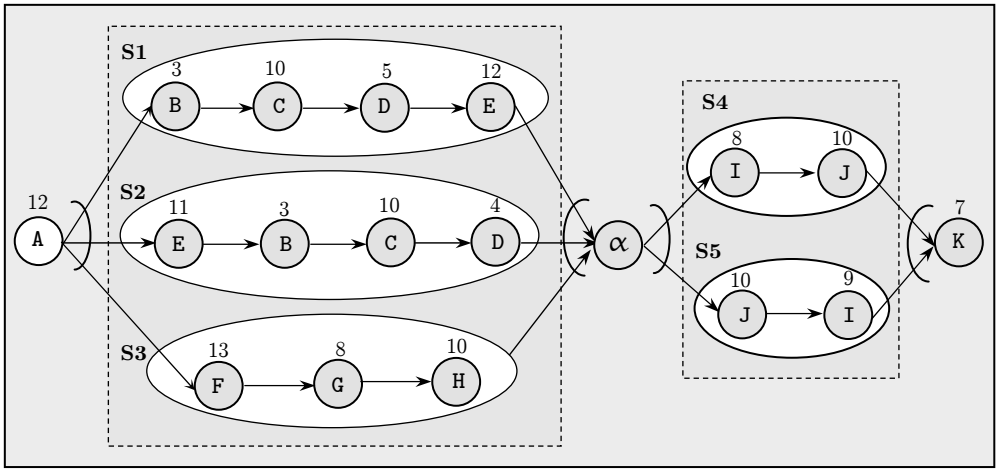

Figure 3.5: S-Graph including fictitious tasks

Figure 3.5 illustrates an example in which the available assembly alternatives also imply mutually exclusive sets of assembly tasks.

As can be observed in Figure 3.5, subgraphs S1, S2 and S3 represent the assembly alternatives for the first subassembly, in which S1 and S2 are assembly variants for the same set of tasks (B, C, D and E); and subgraph S3 represents a sub-process which involves a complete different set of tasks (F, G and H). Therefore, selecting a subgraph for the first subassembly not only determines precedence requirements and task processing times but also the required assembly tasks. The second subassembly represents the ASALBP case considered in previous examples, in which the assembly alternatives (i.e. S4 and S5) involve the same group of task (I and J).

Therefore, a solution for this problem will consist of a choice of two subgraphs (one per subassembly), a number of workstations, and the assignment of the corresponding tasks to the workstations.

The *S-Graph* of Figure 3.6 shows an example of a medium-scale ASALBP. This example, which is based on the precedence diagram of Kilbrid's benchmark problem, consists of 47 tasks and seven subgraphs that represent the assembly alternatives for three subassemblies.

The first subassembly allows two assembly variants which are represented in the *S-Graph* of Figure 3.6 by subgraphs S1 and S2; both alternatives involve tasks 1, 3, 5, 7 and 9. The second subassembly allows three assembly variants: subgraphs S3 and S4 for tasks 20, 21 and 22, and S5 for tasks 46 and 47. For the last subassembly there also available two assembly variants, which are represented by subgraphs S6 and S7 both involving tasks 42, 44 and 45.

In this example, the combination of the assembly subgraphs available for each subassembly, results in a total of 12 possible global assembly variants that are allowed for this assembly process ($12 = 2 \cdot 3 \cdot 2$).

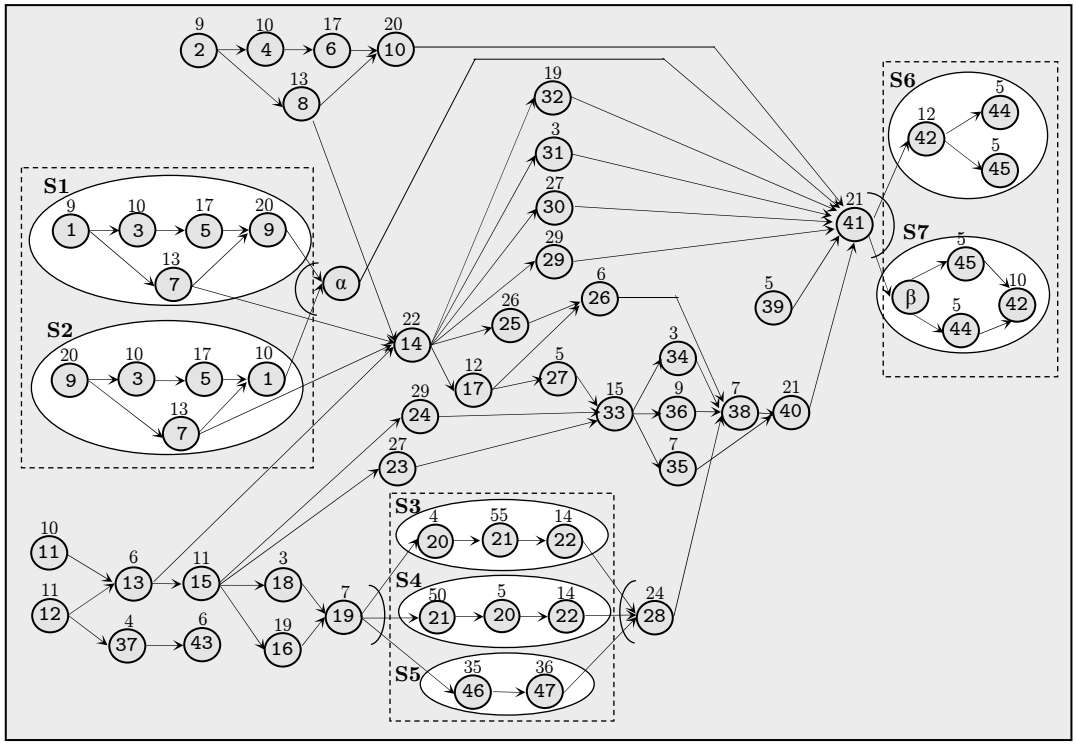

Figure 3.6: Precedence S-Graph for an example of 47 tasks.

The capability of the *S-Graphs* to depict feasible assembly variants in a unique graph may let practitioners to have a better understanding of the system as a whole.

Chapter 4

Mathematical Models of the ASALBP

4.1 Introduction

In its basic form, an assembly line balancing model consists of an objective function than minimizes the number of workstations (i.e. SALBP-1) or the cycle time (i.e. SALBP-2) and a set of constraints that guarantee that every task i is assigned to one and only one workstation, constraints which ensures that the total task processing time assigned to workstation j does not exceed the upper bound on the cycle time, and constraints that guarantee that the precedence relations among the tasks are maintained.

The ASALB Problem considers alternative assembly subgraphs, which in addition may involve different sets of assembly tasks; therefore, apart from cycle time and precedence constraints, subgraphs restrictions need to be taken into account in order to be able to solve this problem: on the one hand, it is necessary to ensure that only one assembly variant (i.e., a subgraph for each subassembly) is selected from amongst the possible ones; on the other hand, it must be guaranteed that only the tasks belonging to the selected subgraphs, and those that do not allow alternatives, are always performed. Furthermore, all tasks have to be performed considering its corresponding precedence constraints.

Accordingly, and in order to formalize and optimally solve the ASALBP, two linear mathematical programming (LMP) models have been developed, which simultaneously solve the decision problem to select an assembly variant and

the problem of assigning the corresponding tasks to the workstations. In the first model, assembly alternatives are represented by a complete precedence graph, which involves the entire set of assembly tasks. The assembly alternatives are thus obtained by making all possible combinations of the available subgraphs. In the second model, referred to as enhanced model, assembly alternatives are represented by an individual subgraph and, therefore, involve only the reduced set of tasks that affected by such a particular subgraph. As a result, the dimension of the model is considerably reduced comparing with the former model.

This chapter describes in detail both the preliminary and the enhanced model. It includes the main modelling assumptions considered in the formulation of the problem and the approaches considered to compute bounds on the number of workstations and other input parameters. The chapter ends by reporting the results of a computational experiment carried out to evaluate and compare the performance of both proposed mathematical models.

4.2 Modelling Assumptions

To facilitate the use of the terminology, in the both mathematical formulations assembly alternatives are referred to as assembly routes[1] undistinguished, any of which defines a known and feasible set of precedence relations among the tasks and the corresponding task processing times. Two different types of assembly routes are considered in the models, as follows.

4.2.1 Global Routes

Global routes are obtained by making all possible combinations of the alternative subgraphs of each available subassembly. Therefore, each global route is represented by a complete precedence graph which depicts the precedence relations of the whole set of tasks required to assemble a given product. In the S-Graph of Figure 3.6, introduced in the previous chapter, it can be observed that there are 12 possible subgraph combinations and, therefore, there are 12 global routes. Precedence graph of Figure 4.1 shows an example of one of these global routes, which is composed by the tasks belonging to subgraphs S1, S3, and S6 and the remaining tasks that do not allow processing alternatives.

[1] The term route has been used previously in other works (e.g., Sawik (2002)) to make reference to assembly plans.

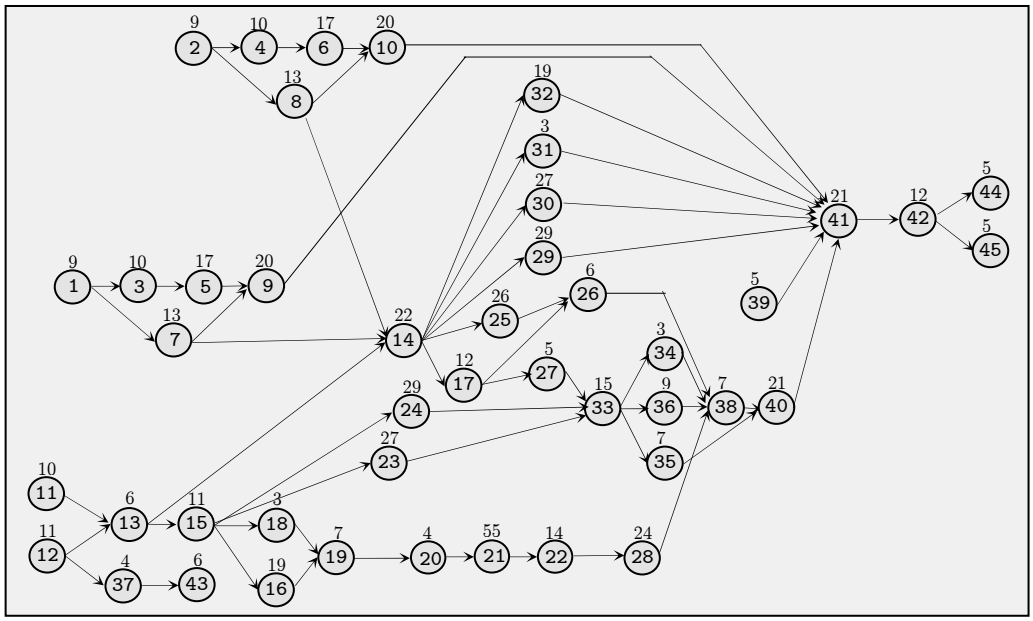

Figure 4.1: The precedence graph of a global route

4.2.2 Partial Routes

A partial route refers to a set of precedence relations that only affects a group of tasks which allow alternative assembly variants. In this case each route is understood as a partial processing alternative which is represented by a subgraph and, consequently, each one only involves a reduced subset of the assembly tasks. For instance, the example of the S-Graph of Figure 3.6 consists of 7 subgraphs; hence, there are 7 partial routes: two represent the processing alternatives for tasks 1, 3, 5, 7 and 9 (S1 and S2); there are two alternatives for tasks 20, 21 and 22 (S3 and S4); one partial route for tasks 46 and 47 (S5); and there are two partial routes for tasks 42, 44, and 45 (S6 and S7). Figure 4.2 shows one of the two processing alternatives available for tasks 1, 3, 5, 7 and 9, which corresponds to subgraph (partial route) S1. Additionally, a basic route, named R0, is considered for those tasks that cannot be performed through alternative routes. In the example of Figure 3.6 there are 34 of such tasks belonging to R0.

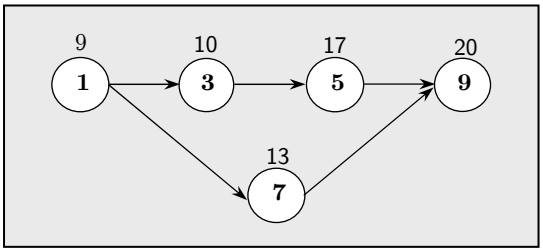

Figure 4.2: a partial route for an ASALBP example with 47 tasks

In a first attempt to mathematically formalize the *Alternative Subgraphs Assembly Line Balancing Problem*, a preliminary model was built in which global routes, represented by a complete precedence graph, were used to define each overall assembly variant. This implies, as it has been previously mentioned, that the whole set of assembling tasks is involved in each global route, including those tasks which do not admit processing alternatives (as shown in Figure 4.1). Consequently, a large number of task-workstation assignment variables need to be defined even when only a small number of assembly routes are available.

By analysing the preliminary model, it was observed that the dimension of the mathematical program could be reduced by defining route-independent assignment variables for those tasks not affected by subassemblies with alternatives, and by considering partial routes for all other tasks. Accordingly, an enhanced mathematical model was developed in which assembly variants are represented by individual subgraphs. In this way, it is possible to reduce the resulting number of variables involved within the model since task-workstation assignment variables are defined per partial route, which involves only a reduced subset of the assembly tasks.

It is valid to remark at this point that for the example of Figure 3.6, there exist 12 global routes all involving 47 tasks, whereas there are only 7 partial routes, each of which consists of at most 5 assembly tasks. Furthermore, the difference between the preliminary and the enhanced model regarding the size of the model to be solved is even greater because the number of assignment variables increases exponentially with increasing number of partial routes and assembly tasks.

On the other hand, considering partial routes complicates even more the modelling process, in particular, when it relates to the precedence constraints. This feature is commented in the following section.

4.2.3 Task precedence relations typology

When considering global routes, the immediate predecessors of a task are fixed for each individual global route; therefore, precedence constraints can be easily established. However, this is not the case when considering partial routes. The difficulty arises due to the fact that an immediate predecessor, or a task itself, may have processing alternatives, from amongst which one is to be selected. Therefore, all possible immediate predecessors of a task have to be considered.

In order to account for all possible precedence relations implied when considering partial routes, and to facilitate its formalization, tasks have been classified into two categories: fixed, which are those without alternatives routes, processed throughout the base route (R0); and mobile, which are those that contemplate alternative routes. Consider the example in Figure 4.3.

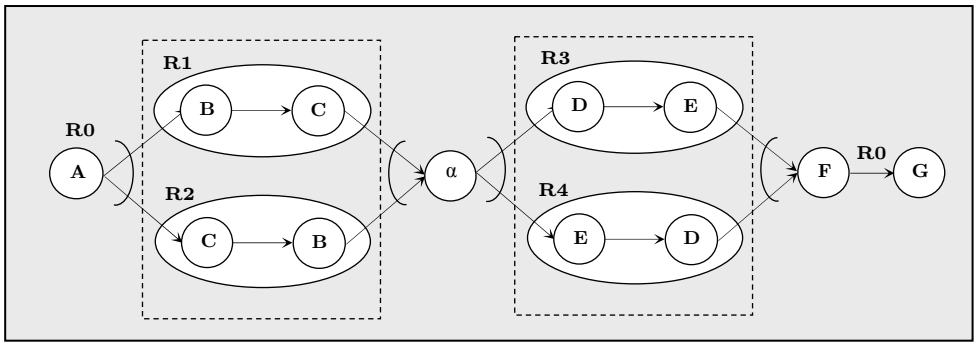

Figure 4.3. Precedence relations of fixed and mobile tasks.

As can be seen in Figure 4.3, tasks A, F and G are fixed, whereas tasks B, C, D and E are mobile due to they can be processed throughout several alternative routes: R1 and R2 for tasks B and C, and R3 and R4 for tasks D and E. On the other hand, α, a fictitious task with nil processing time, is used to represent in the S-Graph precedence relations involving a mobile task i with a mobile predecessor p, which are affected by different alternative routes. This case is represented in Figure 4.3 by tasks D and E, whose predecessors C and B are also mobile tasks; being both groups of tasks affected by different routes: R3 and R4 for D and E, and R1 and R2 for C and B.

Table 4.1 shows the five basic cases of task-predecessor relations, which arise in the example of Figure 4.3.

Table 4.1: Task-predecessor relation typology

	Case	i	p
1	A fixed task i has a fixed predecessor p	G	F
2	A fixed task i has a mobile predecessor p	F	E,D
3	A mobile task i has a fixed predecessor p	B	A
4	A mobile task i has a mobile predecessor p, with i and p belonging to the same route	C	B
5	A mobile task i has a mobile predecessor p, with i and p belonging to different routes	D	C,B

4.3 The Preliminary Model

As previously mentioned, in this model, hereafter referred to as M1, global routes are used to represent each overall assembly variant. Therefore, the model selects a unique global route which determined the precedence constraints and processing times of all required assembly tasks, at the same time assign the tasks to the workstations.

Notation for ASALBP-1

- **Indices**

i for tasks

j for workstations

r for routes

- **Parameters**

n number of tasks ($i = 1,...,n$)

m_{max} upper bound on the number of workstations ($j = 1,...,m_{max}$)

m_{min} lower bound on the number of workstations

nr number of alternative global routes ($r = 1,...,nr$)

t_{ir} duration of task i when processed through route r ($i = 1,...,n$; $r = 1,...,nr$); in some cases t_{ir} is independent on the route

C_{max} upper bound on the cycle time

PD_{ir} set of the immediate predecessors of task i, if task i is processed through route r ($i = 1,...,n;\ r = 1, ...,nr$)

E_{ir} earliest workstation that task i can be assigned to, if task i is processed through route r ($i = 1,...,n;\ r = 1,...,nr$)

L_{ir} latest workstation that task i can be assigned to, if task i is processed through route r ($i = 1,...,n;\ r = 1,...,nr$)

T_{jr} set of tasks potentially assignable to workstation j, if the tasks are processed through route r $\left\{i \mid j \in \left[E_{ir}, L_{ir}\right]\right\}$, ($j = 1,...,m_{max};\ r = 1,...,nr$)

- **Decision variables**

$x_{ijr} \in \{0,1\}$ 1 if task i is assigned to workstation j and processed through route r ($i = 1,...,n;\ r = 1, ...,nr;\ j \in \left[E_{ir}, L_{ir}\right]$)

$y_j \in \{0,1\}$ 1 if there is any task assigned to workstation j ($j = m_{min}+1,...,m_{max}$)

Mathematical formulation for the ASALBP-1: to minimize the number of workstations given C_{max}.

$$Minimize\ z = \sum_{j=m_{min}+1}^{m_{max}} j \cdot y_j \qquad [4.1]$$

$$\sum_{r=1}^{nr} \sum_{j=E_{ir}}^{L_{ir}} x_{ijr} = 1 \qquad \forall i \qquad [4.2]$$

$$\sum_{r=1}^{nr} \sum_{\forall i \in T_{jr}} t_{ir} \cdot x_{ijr} \leq C_{max} \qquad j = 1,\ldots,m_{min} \qquad [4.3]$$

$$\sum_{r=1}^{nr} \sum_{\forall i \in T_{jr}} t_{ir} \cdot x_{ijr} \leq C_{max} \cdot y_j \qquad j = m_{min}+1,\ldots,m_{max} \qquad [4.4]$$

$$\sum_{j=E_{pr}}^{L_{pr}} j \cdot x_{pjr} \leq \sum_{j=E_{ir}}^{L_{ir}} j \cdot x_{ijr} \qquad \forall r, \forall i, \forall p \in PD_{ir} \qquad [4.5]$$

$$\sum_{j=E_{1r}}^{L_{1r}} x_{1jr} \leq \sum_{j=E_{ir}}^{L_{ir}} x_{ijr} \qquad \forall r;\ i = 2,\ldots,n \qquad [4.6]$$

$$x_{ijr} \in \{0,1\} \qquad \forall i, \forall r, \forall j \in [E_{ir}, L_{ir}] \qquad [4.7]$$

$$y_j \in \{0,1\} \qquad j = m_{min}+1,\ldots,m_{max} \qquad [4.8]$$

The objective function [4.1] consists in minimizing the number of workstations. Constraints [4.2] guarantee that every task i is assigned to one and only one workstation. Constraints [4.3] and [4.4] ensure that the total task processing time assigned to workstation j does not exceed the upper bound on the cycle time. Constraints [4.5] impose the precedence conditions. Route uniqueness constraints [4.6] ensure that all the tasks are assigned to the same route. Finally, [4.7] and [4.8] express the binary conditions of the variables.

If one analyzes the previous model, it can be observed that, if the precedence graph is connected, then constraints [4.6] can be removed, due to the fact that constraints [4.5] are sufficient to guarantee route uniqueness. Constraints [4.5] oblige all tasks to be assigned to the same route as their immediate predecessors. In a connected graph, all the tasks are related to one another, direct or indirectly, through their predecessors and successors; therefore, all the tasks are assigned to the same route. In any case, a connected graph can be obtained by defining an initial (or final) fictitious task.

The mathematical formulation of ASALBP-2 can be easily obtained by changing the objective function the formulation for ASALBP-1 by using cycle time ct as the variable that is to be minimized.

4.4 The Enhanced Model

This model, hereafter referred to as M2, considers partial routes to represent the assembly variants that are allowed for each available subassembly. This model selects a single partial route for each available subassembly. Therefore, apart from those for tasks, workstations and routes, an index is required to identify the groups of partial routes that are alternative between each other since only one of those is to be selected. The notation used in this model is presented next. It is valid to mention that tasks processed through route R0 are those which do not admit processing alternatives.

Notation for ASALBP-1

- **Indices**

i for tasks

j for workstations

r for partial routes

q for subsets of partial routes that are alternatives among one another

- **Parameters**

n number of tasks ($i = 1,\ldots,n$)

nr number of partial routes ($r = 0,\ldots,nr$)

nsr number of different sets of partial routes (subgraphs) such that the routes within a set are alternatives to each other (q=1,…,nsr). In the example of Figure 4.3 there are 2 such subsets (nsr=2)

m_{min} lower bound on the number of workstations

m_{max} upper bound on the number of workstations ($j = 1,\ldots,m_{max}$)

R_i set of all routes through which task i can be processed ($i = 1,\ldots,n$)

C_{max} upper bound on the cycle time

t_{ir} duration of task i when processed through route r ($i = 1,\ldots,n;\ r \in R_i$)

TR_r Set of tasks that are affected by route r ($r = 0,\ldots,nr$)

P_{ir} Set of the possible immediate predecessors of task i, if task i is processed through route r ($i = 1,\ldots,n;\ r \in R_i$)

PT_i Set of all possible immediate predecessors of task i ($PT_i = U_{\forall r \in R_i} P_{ir}$)

$E_{ir,}\ L_{ir}$ Earliest and latest station that task i can be assigned to, if task i is processed through route r ($i = 1,\ldots,n;\ r \in R_i$).

SCR_q Subset q of routes that are alternative among one another (q=1,…,nsr). For the example in Figure 4.3, there are two of such subsets: SCR_1 involving R1 and R2 and SCR_2 involving R3 and R4.

❖ **Decision binary variables**

$x_{ijr} \in \{0,1\}$ 1 if task i is assigned to workstation j and processed through route r $(i=1,...,n; \forall r \in R_i;\ \forall\ j \in [E_{ir}, L_{ir}])$

$y_j \in \{0,1\}$ 1 if there is any task assigned to workstation j $(j=m_{min}+1,\ldots,m_{max})$

$ar_r \in \{0,1\}$ 1 if there is any task processed through route r $(r = 1,\ldots,nr)$

Mathematical Model for the ASALBP-1: to minimize the number of workstations given C_{max}.

$$Minimize\ Z = \sum_{j=m_{min}+1}^{m_{max}} j \cdot y_j \qquad [4.9]$$

$$\sum_{j=E_{i0}}^{L_{i0}} x_{ij0} = 1 \qquad \forall i \mid i \in TR_0 \qquad [4.10]$$

$$\sum_{\forall r \in R_i} \sum_{j=E_{ir}}^{L_{ir}} x_{ijr} = \sum_{\forall r \in R_i} ar_r \qquad \forall i \mid i \notin TR_0 \qquad [4.11]$$

$$\sum_{r=0}^{nr} \sum_{\forall i \mid (r \in R_i) \wedge (j \in [E_{ir}, L_{ir}])} t_{ir} \cdot x_{ijr} \le C_{max} \qquad j = 1, \ldots, m_{min} \qquad [4.12]$$

$$\sum_{r=0}^{nr} \sum_{\forall i \mid (r \in R_i) \wedge (j \in [E_{ir}, L_{ir}])} t_{ir} \cdot x_{ijr} \le C_{max} \cdot y_j \qquad j = m_{min}+1, \ldots, m_{max} \qquad [4.13]$$

$$\sum_{j=E_{p0}}^{L_{p0}} j \cdot x_{pj0} \le \sum_{j=E_{i0}}^{L_{i0}} j \cdot x_{ij0} \qquad \forall i \in TR_0, \forall p \in PT_i \mid p \in TR_0 \qquad [4.14]$$

$$\sum_{\forall s \in R_p} \sum_{j=E_{ps}}^{L_{ps}} j \cdot x_{pjs} \le \sum_{j=E_{i0}}^{L_{i0}} j \cdot x_{ij0} \qquad \forall i \in TR_0, \forall p \in PT_i \mid p \notin TR_0 \qquad [4.15]$$

$$\sum_{j=E_{p0}}^{L_{p0}} j \cdot x_{pj0} \le \sum_{\forall r \in R_i} \sum_{j=E_{ir}}^{L_{ir}} j \cdot x_{ijr} + m_{max} \cdot (1 - \sum_{\forall r \in R_i} ar_r) \qquad \forall i \notin TR_0, \forall p \in PT_i \mid p \in TR_0 \qquad [4.16]$$

$$\sum_{j=E_{pr}}^{L_{pr}} j \cdot x_{pjr} \le \sum_{j=E_{ir}}^{L_{ir}} j \cdot x_{ijr} \qquad \forall i \notin TR_0, \forall r \in R_i, \forall p \in P_{ir} \mid [p \notin TR_0 \wedge r \in R_p] \qquad [4.17]$$

$$\sum_{\forall s \in R_p} \sum_{j=E_{ps}}^{L_{ps}} j \cdot x_{pjs} \le \sum_{\forall r \in R_i} \sum_{j=E_{ir}}^{L_{ir}} j \cdot x_{ijr} \qquad \forall i \notin TR_0, \forall p \in PT_i \mid [p \notin TR_0 \wedge (R_i \bigcap R_p = \varnothing)] \qquad [4.18]$$

$$\sum_{r \in SCR_q} ar_r = 1 \qquad q = 1, \ldots, nsr \qquad [4.19]$$

$$\sum_{\forall i \in TR_r} \sum_{j=E_{ir}}^{L_{ir}} x_{ijr} = ar_r \cdot |TR_r| \qquad r = 1, \ldots, nr \qquad [4.20]$$

$$x_{ijr} \in \{0,1\} \qquad \forall i, \forall r \in R_i, \forall j \in [E_{ir}, L_{ir}] \qquad [4.21]$$

$$y_j \in \{0,1\} \qquad j = m_{min}+1, \ldots, m_{max} \qquad [4.22]$$

$$ar_r \in \{0,1\} \qquad r = 1, \ldots, nr \qquad [4.23]$$

The objective function [4.9] minimizes the number of workstations for a given upper bound on the cycle time. The constraints are: [4.10] and [4.11], which ensure that all tasks belonging to a selected partial route are assigned to one and only one workstation, and otherwise tasks are not assigned; [4.12] and [4.13] ensure that the total processing time assigned to workstation j does not exceed the cycle time; [4.14] to [4.18] are the precedence constraints, and correspond to the five different cases presented in Table 4.1, which guarantee that none task is assigned to an earlier workstation than an immediate predecessor; [4.19] are the route uniqueness constraints that ensure that one and only one route for each subassembly is selected from among the possible routes; and [4.20] guarantees that tasks belonging to a particular precedence subgraph are assigned to the same route. Finally, [4.21], [4.22] and [4.23] express the binary conditions of the variables.

Similarly to the preliminary case, the mathematical formulation for the ASALBP-2 version can also be easily obtained by using the enhanced formulation but using the cycle time ct as the variable to minimize instead of the number of workstations.

4.5 Computation of input parameters

This section presents the approaches used in this work to determine, on the one hand, the earliest and latest workstations to which a task can be assigned; and on the other hand, the lower and upper bounds on the number of workstations that help to reduce the number of the variables and constraints of both proposed mathematical models.

4.5.1 Earliest and latest workstations

The methods used to determine the values of the earliest and latest workstation to which a task i can be assigned are based on a well-know approach traditionally applied to SALBP (e.g. Talbot *et al.* (1986), Klein and Scholl (1996) and Scholl (1999)).

According to this, a task i can not be assigned to a workstation before the total time of all its predecessors has been already assigned, but should be assigned before the remaining available time is less than the total time of all its followers. Nonetheless, this concept should be adapted in order to contemplate the available assembly alternatives that characterize a given ASALB Problem.

The following notation is considered*:*

AP_{ir} is the summation of the processing times of all predecessors of task i, when such tasks are processed according to the best assembly alternative (i.e., the combination of subgraphs with the minimum total time) and task i is processed thought route r.

AF_{ir} is the summation of the processing times of all successors of task i, when such tasks are processed according to the best assembly alternative (i.e., the combination of subgraphs with the minimum total time) and task i is processed thought route r.

For example, if the S-Graph of Figure 4.4 is considered:

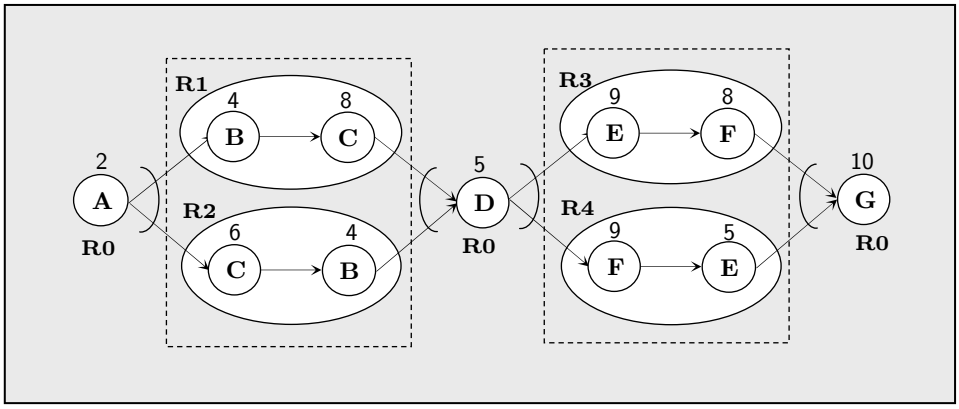

Figure 4.4: S-Graph for a small ASALBP example involving seven tasks

The following values are obtained:

$AP_{A0} = 0$
$AF_{A0} = min(4 + 8 ; 6 + 4) + 5 + min(9 + 8 ; 9 + 5) + 10 = 39$
$AP_{C1} = 2 + 4 = 6$
$AF_{C1} = 5 + min(9 + 8 ; 9 + 5) + 10 = 29$
$AP_{C2} = 2$
$AF_{C2} = 4 + 5 + min(9 + 8 ; 9 + 5) + 10 = 33$

Thus, the earliest and latest workstation values to which task i can be assigned when processed through route r are computed according to the equations [4.24] and [4.25], respectively.

$$E_{ir} = \lceil (t_{ir} + AP_{ir}) / tc \rceil \qquad [4.24]$$

$$L_{ir} = m_{max} + 1 - \lceil (t_{ir} + AF_{ir}) / tc \rceil \qquad [4.25]$$

4.5.2 Lower bound on the number of workstations

A simple theoretical minimum number of workstations, m_{min}, is defined in the literature (e.g. Baybars (1986), Johnson (1988), Scholl and Klein (1997), Scholl (1999) and Becker and Scholl (2006)), according to which the total time available in the assembly line must not be smaller that the total load required to process all tasks. Subsequently, this value is computed according to [4.26] which is the integer equal or greater than the quotient between the total processing time and the cycle time.

$$m_{min} = \left\lceil \sum_{i=1}^{n} t_i \; / \; tc \right\rceil \qquad [4.26]$$

In the ASALBP a new parameter, called Bt_{sum}, required to compute m_{min}, is defined as the summation of all the processing times when tasks are processed according to the best assembly alternative as defined previously. When assembly alternatives affect mutually exclusive sets of tasks, only the time of the alternative that lasts less is considered. Therefore, a lower bound on the number of workstations is given by [4.27].

$$m_{min} = \lceil Bt_{sum} / tc \rceil \qquad [4.27]$$

Considering the example of Figure 4.4 and a cycle time $ct = 10$:

$Bt_{sum} = 2 + min\ (4 + 8\ ;\ 6 + 4) + 5 + min\ (9 + 8\ ;\ 9 + 5) + 10 = 41$

$m_{min} = \lceil 41/10 \rceil = 5$

There exist other procedures to calculate lower bounds based on the analysis of the processing times of the successors of a task i; for example, Johnson (1988), Scholl and Klein (1997) and Scholl (1999) consider that m_{min} can be the number of tasks that have a processing time greater than half of the cycle time, since all of such tasks have to be assigned to different workstations. This bound can be further improved by adding to it half of the number of tasks with processing time equal to half of the cycle time, given that two of such tasks can be processed in the same workstation.

By following the same argument, a third value of m_{min} can be defined by considering the fact that three tasks with processing time greater than $tc/3$ cannot be processed in the same workstation; furthermore, none of these can share workstation with other tasks with time greater that $2tc/3$. This bound can be also further adjusted by considering that tasks with processing times exactly equal to $tc/3$ or equal to $2tc/3$ can share the same workstation (i.e., one task with $t_i = 2tc/3$ and other with $tc/3$ or three tasks with $t_i = tc/3$).

For the ASALB Problem these values are computed by adapting these concepts and by considering the Bt_{sum} parameter for the sum of processing times, as discussed previously.

Other methods used to compute lower bounds on the number of workstations are discussed, for example, in Scholl and Klein (1997) and in Fleszar and Hindi (2003).

4.5.3 Upper bound on the number of workstations

The simplest upper bound on the number of workstations is the number of tasks, since assigning only one task to each workstation is a feasible solution. In the case of the ASALB Problem this upper bound [4.28] is obtained by considering the combination of subgraphs that involves the maximum number of required assembly tasks.

$$m_{max} = n \qquad [4.28]$$

More adjusted upper bounds on the number of workstations can be computed following, for example, the procedures discussed in Scholl (1999) for the simple case.

Another approach that can be considered to obtain an upper bound on the number of workstations is to generate a feasible solution by applying an heuristic procedure.

4.6 Computational Experiment

To evaluate and compare the performance of the proposed mathematical models, M1 and M2, previously described in sections 4.3 and 4.4, respectively; a computational experiment was carried out. Both models were implemented and several problem instances were solved by using the ILOG CPLEX© optimization software, version 9.0. All computations were performed on a PC Pentium 4, CPU 2.88 GHz with 512 Mb of RAM.

4.6.1 Benchmark Selection

Since the ASALBP is a new generalized assembly line balancing problem, the data sets used in the computational experiment were designed by incorporating various alternative assembly subgraphs to benchmark SALB Problems available at www.assembly-line-balancing.de (the homepage focused on assembly line balancing research).

The following 9 problems were considered: Bowman, Mansor, Mitchell, Buxey, Gunther, Kilbrid, Hahn, Warnecke and Tonge; involving 8, 11, 21, 29, 35, 45, 53, 58 and 70 tasks, respectively. Two, three and four subassemblies were incorporated to each original problem, for each of which several assembly alternatives were generated (see Table 4.2): from 2 to 60 global routes were considered for model M1, and from 3 to 11 partial routes for model M2. Furthermore, up to three different cycle time values, also based on the available benchmark data sets, were considered for each test problem. Furthermore, new sets of tasks were added to the problems in order to account for problems instances involving mutually exclusive assembly processes. Then, a total of 44 problem instances were solved with both models.

All data for the problem instances solved are shown in Table 4.2, which includes the name of the problem, the number of tasks n, the cycle time ct, the number of global routes for model M1 and partial routes for model M2, and the number of constraints and binary variables involved in each model.

Table 4.2: Data of the ASALBP instances

Problem	n	ct	No. of routes		Contraints		Variables	
			Global	Partial	M1	M2	M1	M2
Bowman-1	10	20	6	5	134	56	434	615
Bowman-2	12	20	18	8	152	76	1744	880
Mansor-1	11	48	6	5	164	62	544	541
Mansor-2	11	62	6	5	158	58	408	407
Mansor-3	11	94	6	5	152	54	272	273
Mansor-4	11	62	12	7	288	74	804	547
Mansor-5	11	62	15	8	352	78	1002	614
Mansor-6	11	48	15	8	358	75	1336	708
Mansor-7	11	94	15	8	346	68	668	408
Mitchel-1	21	14	6	5	347	111	1792	1783
Mitchel-2	21	21	6	5	333	101	1280	1275
Mitchel-3	21	35	6	5	319	92	640	640
Mitchel-4	21	14	15	8	770	130	4438	2668
Mitchel-5	21	21	15	8	756	120	3170	1908
Mitchel-6	21	35	15	8	742	111	1585	958
Buxey-1	29	54	6	5	444	134	1941	1936
Buxey-2	29	36	6	5	464	161	3168	3155
Buxey-3	29	54	12	7	861	147	3850	2581
Buxey-4	29	30	18	8	1308	159	11004	5510
Buxey-5	29	36	18	8	1298	152	9432	4724
Buxey-6	29	54	18	8	1278	139	5764	2890
Gunther-1	35	41	32	11	2633	205	25806	8911
Gunther-2	40	81	60	11	4287	189	28824	6276
Kilbrid-1	45	56	12	7	1383	204	10840	7247
Kilbrid-2	45	79	12	7	1365	191	7588	5173
Kilbrid-3	45	92	12	7	1359	185	6504	4435
Kilbrid-4	45	79	18	8	2001	191	11368	5173
Kilbrid-5	45	92	18	8	1995	185	9744	4435
Kilbrid-6	45	69	24	10	2505	217	17312	7241
Kilbrid-7	45	79	24	10	2499	220	15148	7416
Kilbrid-8	45	92	24	10	2493	214	12948	6358
Hahn-1	53	2004	6	5	851	236	4480	4471
Hahn-2	53	3507	6	5	833	218	2560	2557
Hahn-3	53	4676	6	5	829	210	1920	1600
Hahn-4	53	4676	12	7	1635	228	3190	2403
Hahn-5	53	3507	12	7	1646	236	5104	3407
Hahn-6	53	4676	18	8	2424	238	4780	2403
Hahn-7	53	3507	18	8	2432	235	7648	3976
Hahn-8	55	2004	18	8	2450	253	13384	6952
Hahn-9	58	2004	24	10	3400	263	19516	8157
Hahn-10	62	2806	36	11	5210	280	22340	7471
Warnecke-1	58	111	2	3	368	235	4754	3186
Warnecke-2	58	111	4	5	648	253	7888	6318
Tonge	70	185	8	7	1428	342	21356	18702

Table 4.3 shows the results obtained by optimally solving the problems characterized in table 4.2 with both proposed mathematical models. It includes, for both models, the solution time (in seconds) and the percentage of improvement of M2 over M1, concerning the solution time.

Table 4.3: Results of optimally solving ASALBP instances

Problem	Solution Time		% of improprement
	M1	M2	
Bowman-1	0.56	0.04	92.9
Bowman-2	0.17	0.03	82.4
Mansor-1	0.20	0.02	90.0
Mansor-2	0.04	0.02	50.0
Mansor-3	0.03	0.01	66.7
Mansor-4	0.09	0.06	33.3
Mansor-5	0.11	0.03	72.7
Mansor-6	0.80	0.40	50.0
Mansor-7	1.12	0.03	97.3
Mitchel-1	1.84	0.15	91.8
Mitchel-2	0.25	0.04	84.0
Mitchel-3	0.12	0.04	66.7
Mitchel-4	7.59	0.33	95.7
Mitchel-5	4.93	0.07	98.6
Mitchel-6	1.04	0.13	87.5
Buxey-1	61547	92.03	100
Buxey-2	18485	0.86	100
Buxey-3	806	10.23	100
Buxey-4	>>200000	862	100
Buxey-5	>>200000	6.99	100
Buxey-6	>>200000	2.82	100
Gunther-1	89558	14805	83.5
Gunther-2	467	0.31	100
Kilbrid-1	213	1.41	100
Kilbrid-2	20.85	1.56	92.5
Kilbrid-3	49.75	7.10	85.7
Kilbrid-4	830	1.06	100
Kilbrid-5	930	1.56	100
Kilbrid-6	110	0.56	100
Kilbrid-7	112	2.02	98.2
Kilbrid-8	114	1.81	98.4
Hahn-1	2.63	0.18	93.2
Hahn-2	11.80	0.09	99.3
Hahn-3	15.94	0.34	97.9
Hahn-4	35.35	0.14	100
Hahn-5	29.13	0.09	100
Hahn-6	114	0.13	100
Hahn-7	92.53	1.20	98.7
Hahn-8	1373	33.52	97.6
Hahn-9	8356	3.48	100
Hahn-10	19785	249	98.7
Warnecke-1	7200	638	91.1
Warnecke-2	17709	1410	92.0
Tonge	>>200000	80122	100

4.6.2 Analysis of the results obtained with M1 and M2

The computational experiment showed that optimal solutions can be obtained and guaranteed in a reasonable amount of time, only for some of the small- and medium-scaled problem instances considered in the experiment, e.g. test problems involving from 10 to around 30 tasks and from 5 to 11 assembly subgraphs (i.e. partial routes). Such results could be expected taking into account the *NP-hard* nature of the ASALBP.

As can be observed in Table 4.2, the number of variables and constraints was significantly reduced in model M2 (as it was intended). As a result (see Table 4.3), the computation time required by M2 to solve a problem instance was considerably smaller comparing with the time required by the preliminary model M1 to solve the same problem instance. Table 4.3 also revealed that in all cases model M2 outperformed model M1: M2 achieved around 90.6% of average improvement over M1; reaching a 100% of improvement in more than a third of the problems solved.

Notwithstanding, most problems are optimally solved in a computing time significantly small, as can be observed in Table 4.3, the time required by the mathematical model to solve ASALB Problems increases exponentially with the number of tasks and the number of processing alternatives that are available. Furthermore, for some test problems the required computing time was significantly large for both mathematical models, such as, for example, Gunther-1 and Tonge. Bigger scale problems involving more than 70 tasks (e.g., Lutz and Arcus2, involving 89 and 111 tasks, respectively) were also intended to be optimally solved; however, neither M1 nor M2 were able to obtain the optimal solution within one week of computing time. Therefore, it is necessary to consider other methods, i.e. approximate procedures, in order to efficiently solve real-scale ASALBP.

Chapter 5

Approximate Methods to Solve the ASALBP

5.1 Introduction

Exact methods have a problem size limitation and can only be applied to solve small and medium scale problems. Although, in some cases mathematical programming models can provide the optimal solution to more realistic problems, the required computation time may be too large to be of practical use. As previously discussed, the ASALBP is more difficult to solve optimally, comparing with the simple case which by nature is *NP-hard*, since the inherent decision problem to selects the assembly subgraphs implies an even bigger computational effort. Therefore, in this thesis a group of heuristics methods are proposed to solve the Alternative Subgraphs Assembly Line Balancing Problem, aiming at yielding reasonable solutions in a significantly small computing time.

As previously mentioned, most heuristic techniques considered in the literature (e.g. Scholl and Voss (1996), Amen (2001), Dolgui *et al.* (2005), Fernades and Ribeiro (2005), Becker and Scholl (2006)) are constructive methods based on single priority rules, which have been successfully applied to assembly line balancing problems. Therefore, a significant number of constructive methods to solve the ASALBP have been designed, implemented and evaluated in this doctoral thesis (section 5.2).

In order to improve the solution of the approximate methods, two local optimization procedures, based on an adaptation of two classical neighbourhood search strategies, have been also implemented here (section 5.3). All these procedures are evaluated and compared via computational experiment. The analysis of the results is reported at the end of this chapter (section 5.4).

5.2 Heuristic Methods

The heuristic methods proposed here systematically build the solution to the ASALBP by selecting the assembly subgraphs and incrementally assigning the tasks to the workstations. Such methods use priority-rule-based and random strategies to select both the assembly subgraphs and the next task to be assigned. In the former case, the selection is done considering a decreasing (or increasing) value of a predetermined priority rule; in the latter case, tasks and/or subgraphs are selected following either a uniform distribution or a probability function based on weighted values of the priority rules.

A solution provided by these methods consists of a set of subgraphs (one for each subassembly that allows assembly variants), which determines the assembly tasks, the processing times, a number of required workstations and the assignment of the corresponding tasks to the workstations. In order to facilitate the evaluation of constructive methods involving most well-known priority rules, the proposed procedures aim at minimizing the number of workstations; therefore, they focus on resolving ASALBP-1.

To describe the proposed heuristic methods the following notation is considered:

n Number of tasks

ct Cycle time

m_{max} Upper bound on the number of workstations

R_i Set of all subgraphs through which task i can be processed ($i = 1,\ldots,n$)

t_{ir} Duration of task i when processed through subgraph r ($i = 1,\ldots,n$; $r \in R_i$)

P_{ir} Set of immediate predecessors of task i if task i is processed through subgraph r ($i = 1,\ldots,n$; $r \in R_i$)

S_{ir} Set of all successors of task i if it is processed through subgraph r ($i = 1,\ldots,n$; $r \in R_i$)

SR Set of selected subgraphs. SR is generated once the priority rules to select the subgraphs have been applied.

Once set SR is known (i.e. the assembly subgraphs have been selected), the following values can be defined:

AVT Set of available tasks, which is formed with the tasks that belong to the selected subgraphs and those tasks that do not allow assembly variants.

AST Set of assignable tasks. A task is assignable if all its predecessors have already been assigned and its time plus the time of the tasks assigned to the current workstation does not exceed the cycle time.

$sub(i)$ Subgraph chosen for task i ($\forall i \in AVT$); in this way it is possible to know $t_{i,sub(i)}$, which is the duration of task i. Since task $i \in AVT$, it is verified that subgraph $sub(i) \in SR$.

E_i Earliest workstation to which task i can be assigned ($\forall i \in AVT$).

L_i Latest workstation to which task i can be assigned ($\forall i \in AVT$).

SI_i Set of immediate successors of task i ($\forall i \in AVT$).

S_i Set of total successors of task i ($\forall i \in AVT$).

The general scheme for the proposed heuristic procedures is given in Algorithm 1.

Algorithm 1

Step 1. Set the stopping condition.

Step 2. Select one subgraph for each available subassembly and build the set of selected subgraphs SR.

Step 3. Form the set of available tasks, AVT.

Step 4. Set as current workstation the first workstation.

Step 5. If AVT is not empty, determine the set of assignable tasks, AST.

Step 6. Select the next task to be assigned to the current workstation from AST.

Step 7. If there are no assignable tasks (i.e. AST is empty) but there are remaining available tasks (i.e. AVT is not empty), then open a new workstation.

Step 8. Remove the assigned task from AVT and update AST.

Step 9. Repeat from 6 to 8 until all assembly tasks have been assigned (i.e. AVT is empty).

Step 10. If the solution obtained at the current iteration improves the best stored solution, then store current solution.

Step 11. Repeat from 2 onwards as long as the stopping condition holds.

Regarding the selection criterion used to select the assembly subgraphs (at step 2) and the tasks (at step 6), Algorithm 1 represents a *single-pass* method that generates a single solution, or a *multi-pass* method, in which multiple solutions are generated and compared keeping the best of all obtained solutions. The stopping condition of the Algorithm 1 is defined by a single iteration in the case of single-pass methods, and determined by a maximum computing time in the case of a multi-pass methods.

The following are all selection criterions considered in the proposed procedures.

Decision criteria for subgraphs

Four criteria are used to select the assembly subgraphs: three priority rules, and random choice (RS).

As previously mentioned, in the case of random choice subgraphs can be selected considering either a uniform distribution (i.e. all subgraphs of the same subassembly have the same probability of being selected) or a probability distribution based on weighted values of the priority rules.

The three priority rules considered are the following:

a. Minimum NP: this rule ranks the subgraphs of the same subassembly according to ascending number of precedence relations involved in each subgraph, which is the total number of arcs entering into and within the subgraph.
b. Minimum TT: subgraphs are ranked according to ascending total processing time.
c. Minimum NT: subgraphs are ranked according to ascending number of tasks.

Decision criteria for tasks

The decision criteria used to select the next task to be assigned are presented in Table 5.1, which shows an adaptation to the ASALBP of 13 well-known priority rules (e.g. Talbot *et al.* (1986)) and random choice assignment. Similarly to subgraphs, the random strategy can follow either a uniform distribution or a function based on weighted values of the priority rules: $f(pr)$.

Priority rules values are basically determined by measuring task processing times and precedence relations, and by considering the cycle time. For instance, RPW (*Rank Positional Weight*) can be computed by adding to the task time the sum of the times of all its successors. It is valid to mention at this point that, according to Algorithm 1, set SR is defined before the assembly tasks are selected.

Table 5.1: Decision criteria for tasks

No.	Name	Decision criteria	Procedure
1	RPW	Maximum Rank Positional Weight	$RPW_i = t_{i,sub(i)} + \sum_{j \in S_{i,sub(i)}} t_{j,sub(j)}$
2	T	Maximum Task Time	$t_{i,sub(i)}$
3	EW	Minimum Earliest Workstation	$EW_i = \left\lceil \left(t_{i,sub(i)} + \sum_{j \in P_{i,sub(i)}} t_{j,sub(j)} \right) \Big/ ct \right\rceil$
4	LW	Minimum Latest Workstation	$LW_i = m_{\max} + 1 - \left\lceil \left(t_{i,sub(i)} + \sum_{j \in S_{i,sub(i)}} t_{j,sub(j)} \right) \Big/ ct \right\rceil$
5	N	Minimum task Number	i
6	Sk	Minimum Slack	$Sk_i = LW_i - EW_i$
7	TLW	Minimum task time divided by Latest Workstation	$TL_i = t_{i,sub(i)} / LW_i$
8	IS	Maximum Number of Immediate Successors	$IS_i = \lvert SI_i \rvert$
9	TS	Maximum Number of total successors	$TS_i = \lvert S_i \rvert$
10	TTS	Maximum Task Time plus Total Number of Successors	$TTS_i = t_{i,sub(i)} + TS_i$
11	STS	Maximum Average Time of Successors	$STS_i = \left(\sum_{j \in S_{i,sub(i)}} t_{j,sub(j)} \right) \Big/ TS_i$
12	$TSSk$	Maximum Number of Total Successors divided by Sk	$TSSk_i = TS_i / (Sk_i + 1)$
13	$LWTS$	Minimum Average Latest Workstation	$LWTS_i = LW_i / (TS_i + 1)$
14	RT	Random task assignment	$i \sim U[0..nt] \vee i \sim f(pr)$

5.2.1 Single-pass methods

Single-pass methods generate a single solution by exploring the solution space only via single priority rules, whereby the subgraphs and tasks are selected according to the descendant or ascendant values of the predetermined priority rules. The stopping condition of Algorithm 1 is thus defined by a single iteration that is completed once all tasks, which belong to the subgraphs selected in Step 2, have been assigned to the workstations.

Table 5.2 lists the names and numbers of all 39 single-pass heuristic methods that are obtained by combining the priority rules for tasks (defined in Table 5.1) with the decision rules considered for the assembly subgraphs.

Table 5.2: Single-pass methods

Rules for tasks	**Rules for subgraphs**					
	NP		***TT***		***NT***	
	No.	**Label**	**No.**	**Label**	**No.**	**Label**
RPW	1	NP_RPW	14	TT_RPW	27	NT_RPW
T	2	NP_T	15	TT_T	28	NT_T
EW	3	NP_EW	16	TT_EW	29	NT_EW
LW	4	NP_LW	17	TT_LW	30	NT_LW
N	5	NP_N	18	TT_N	31	NT_N
Sk	6	NP_Sk	19	TT_Sk	32	NT_Sk
TLW	7	NP_TLW	20	TT_TLW	33	NT_TLW
IS	8	NP_IS	21	TT_IS	34	NT_IS
TS	9	NP_TS	22	TT_TS	35	NT_TS
TTS	10	NP_TTS	23	TT_TTS	36	NT_TTS
STS	11	NP_STS	24	TT_STS	37	NT_STS
TSSk	12	NP_TSSk	25	TT_TSSk	38	NT_TSSk
LWTS	13	NP_LWTS	26	TT_LWTS	39	NT_LWTS

As seen in Table 5.2, each method is labelled according to the following notation: [*SubgraphRule_TaskRule*]. For example, *TT_RPW* selects a combination of subgraphs that requires the minimum total processing time (*TT*) and ranks the tasks to be assigned considering descending *Rank Positional Weight* (*RPW*) values. Descending values are considered when the

rule refers to a maximization criterion, and ascending values, when it refers to a minimization criterion. Additionally, all methods use *task index* (N) as a tie-breaker rule for tasks. In the case of subgraphs, TT is used as a tie-breaker rule for the [$NT\_TaskRule$] and [$NP\_TaskRule$] methods, and NT for the [$TT\_TaskRule$] methods.

The following example illustrates how subgraphs and tasks are selected in the single-pass procedures proposed here.

Example 5.1: Single-pass procedures

The S-Graph of Figure 5.1 depicts an ASALBP that involves 17 tasks and 7 subgraphs, which represent the assembly variants for three parts of the system that allow alternatives: S1 and S2 for the first subassembly; S3 and S4 for the second; and S5, S6 and S7 for the third.

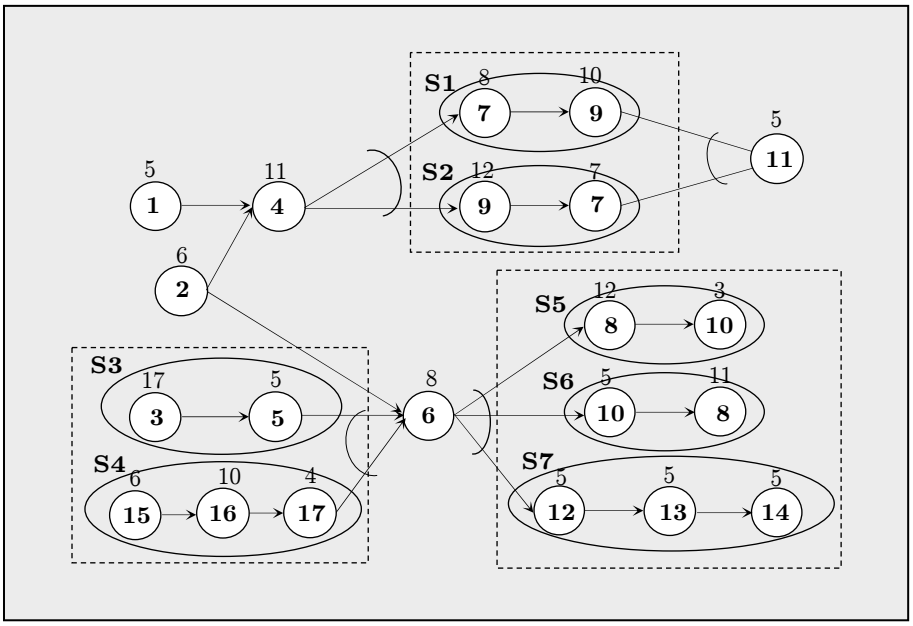

Figure 5.1: Precedence S-Graph for an ASALBP involving 17 tasks.

Table 5.3 shows the computed priority rule values for each available subgraph of Figure 5.1.

Table 5.3: Priority rule values for the assembly subgraphs

Priority rule	Subgraph						
	S1	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**
TT	18	19	22	20	15	16	15
NP	2	2	1	2	2	2	3
NT	2	2	2	3	2	2	3

Considering that priority rule TT is used at step 2 of Algorithm 1, then S1 will be selected for the first subassembly, since it involves the minimum total processing time of 18 time units, whereas S2 requires 19 time units. Similarly, for the second subassembly, the selected subgraph will be S4. For the third subassembly, however, more than one subgraph matches the selection criteria, meaning that a tie-breaker rule must be applied; application of the tie-breaker rule NT thus yields S5 as the selected subgraph.

Therefore, by using the $[TT\_TaskRule]$ family of methods, regardless of the rule used for tasks, the selected subgraphs are S1, S4 and S5, and the corresponding available tasks (i.e. the set AVT formed at step 3 of Algorithm 1) are 1, 2, 4, 6, 7, 8, 9, 10, 11, 15, 16, and 17. Similarly, when the $[NT\_TaskRule]$ or $[NP\_TaskRule]$ methods are applied, the selected subgraphs are S1, S3 and S5, and the corresponding available tasks are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 and 11.

Table 5.4 summarizes the results of using different methods to select the subgraphs at step 2 of Algorithm 1. It includes the method notation, the selected subgraphs, the resulting available assembly tasks (i.e., set AVT) and the first set of assignable tasks (i.e., AST) generated at step 5 of Algorithm 1.

Table 5.4: Selected subgraphs, available and assignable tasks

Methods	***SR***	***AVT***	***AST***
$[TT\_TaskRule]$	S1, S4, S5	1, 2, 4, 6, 7, 8, 9, 10, 11, 15, 16, 17	1, 2, 15
$[NT\_TaskRule]$	S1, S3, S5	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	1, 2, 3
$[NP\_TaskRule]$	S1, S3, S5	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11	1, 2, 3

If method TT_T is applied, then the set of assignable tasks comprises tasks 1, 2 and 15, which should be ranked according to descending values of task times -in this case 2, 15 and 1. It can be observed that tasks 2 and 15 have the same processing time. However, according to the tie-breaker rule, task 2 will be the first task to be assigned because it meets the tie-break condition (i.e. it has the smallest task index). If method TT_N is being used (i.e. N is considered as a primary rule), then task 1 will be the first to be assigned instead of task 2.

Table 5.5 includes the results obtained by applying four of the decision rules for tasks (RPW, T, EW and LW), taking, for example, NT as the decision rule for subgraphs, and assuming that ct=20 and m_{max}=9. It lists the computed priority rules values for each assignable task of the first set AST (i.e. 1, 2, 3), the first task to be assigned (1^{st} at), the number of required workstations (m), the resulting task assignment and the corresponding workstation time (shown in parentheses).

Table 5.5: Results of applying single-pass methods

Rule for tasks	1	2	3	1^{st} at	m	Task assignment (workstation time)					
						I	II	III	IV	V	VI
RPW	39	63	45	2	6	2, 1 (11)	3 (17)	4, 5 (16)	6, 7 (16)	8, 10 (15)	9, 11 (15)
T	5	6	17	3	5	3 (17)	2, 1, 5 (16)	4, 6 (19)	8, 7 (20)	9, 11, 10 (18)	-
EW	1	1	1	1	6	1, 2 (11)	3 (17)	4, 5 (16)	6, 7 (16)	9, 11 (15)	8,10 (15)
LW	8	6	7	2	6	2, 1 (11)	3 (17)	4, 5 (16)	6, 7 (16)	8, 10 (15)	9, 11 (15)

Table 5.5 reveals that different results can be obtained by using different decision rules. In this example, method NT_T requires five workstations, whereas the other methods require six workstations.

5.2.2 Multi-pass methods

Multi-pass methods solve several times the same problem instance by using a stochastic mechanism to select either the subgraphs or the tasks, or both. Therefore, multiple solutions are generated by repeating the general scheme given by Algorithm 1 and returning the best of all solutions obtained during the available computing time, which is the stopping condition.

Four classes of multi-pass heuristic procedures are distinguished:

a. [$Random\_TaskRule$]: at step 2 of Algorithm 1, a set of subgraphs (i.e. one for each subassembly) is selected randomly, having all subgraphs for the same subassembly the same probability of being selected. Then, at step 6, tasks are assigned by applying one of the thirteen single-pass priority rules. The whole procedure is then repeated by randomly selecting at any iteration a new set of assembly subgraphs and generating a line balance.

b. [*SubgraphRule_Random*]: at step 2 of Algorithm 1, subgraphs are selected by using one of the three priority rules for subgraphs; therefore, the selected subgraphs remain fixed during the given length of computing time. At each iteration a new balance is generated by randomly selecting (following a uniform probability distribution) the next task to be assigned, considering only the tasks belonging to the selected subgraphs.

c. [*Random_Random*]: both subgraphs and the next task to be assigned to the current workstation are selected randomly both following a uniform probability distribution.

d. *W*-[*SubgraphRule_TaskRule*]: both subgraphs and tasks are randomly selected. The probability distributions are built using weighted values that are proportional or inversely proportional, when using a maximizing or minimizing criterion, respectively, to the values obtained considering a given priority rule.

A total of 56 multi-pass heuristic methods have been proposed. Methods using single rule values, hereafter referred to as non-weighted multi-pass methods, are summarized in Table 5.6: methods 40 to 52 are of class *a*, 53 is of class *c*, and methods 54 to 56 are of class *b*.

Table 5.6: Non-weighted multi-pass methods

Rule for subgraph: Random (*RS*)					
Rule for tasks	**No.**	**Label**	**No.**	**Rule for tasks**	**Label**
RPW	40	*RS_RPW*	47	***IS***	*RS_IS*
T	41	*RS_T*	48	***TS***	*RS_TS*
EW	42	*RS_EW*	49	***TTS***	*RS_TTS*
LW	43	*RS_LW*	50	***STS***	*RS_STS*
N	44	*RS_N*	51	***TSSk***	*RS_TSSk*
Sk	45	*RS_Sk*	52	***LWTS***	*RS_LWTS*
TLW	46	*RS_TLW*	53	***RT***	*RS_RT*

Rule for tasks: Random (*RT*)								
Rule for subgraph	**No.**	**Label**	**Rule for subgraph**	**No.**	**Label**	**Rule for subgraph**	**No.**	**Label**
NP	54	*NP_RT*	***TT***	55	*TT_RT*	***NT***	56	*NT_RT*

The combination of the resulting probability distributions based on the various priority rules for subgraphs and tasks produces 39 class-d multi-pass methods, hereafter also referred to as weighted multi-pass methods, which are listed in Table 5.7 (i.e. methods 57 to 95).

Table 5.7: Weighted multi-pass methods

Rule for tasks	Rule for subgraphs					
	NP		***TT***		***ENT***	
	No.	**Label**	**No.**	**Label**	**No.**	**Label**
RPW	57	W-NP_RPW	70	W-TT_RPW	83	W-NT_RPW
T	58	W-NP_T	71	W-TT_T	84	W-NT_T
EW	59	W-NP_EW	72	W-TT_EW	85	W-NT_EW
LW	60	W-NP_LW	73	W-TT_LW	86	W-NT_LW
N	61	W-NP_N	74	W-TT_N	87	W-NT_N
Sk	62	W-NP_Sk	75	W-TT_Sk	88	W-NT_Sk
TLW	63	W-NP_TLW	76	W-TT_TLW	89	W-NT_TLW
IS	64	W-NP_IS	77	W-TT_IS	90	W-NT_IS
TS	65	W-NP_TS	78	W-TT_TS	91	W-NT_TS
TTS	66	W-NP_TTS	79	W-TT_TTS	92	W-NT_TTS
STS	67	W-NP_STS	80	W-TT_STS	93	W-NT_STS
TSSk	68	W-NP_TSSk	81	W-TT_TSSk	94	W-NT_TSSk
LWTS	69	W-NP_LWTS	82	W-TT_LWTS	95	W-NT_LWTS

Example 5.2: Multi-pass procedures

Considering the *S-Graph* of Figure 5.1 and that method *W-TT_T* is applied, then, the cumulative probability distribution for selecting a subgraph ss_1, ss_2 and ss_3 for subassembly 1, 2 and 3, respectively, are as follows ($r \epsilon [0,1)$ is a random value):

$$ss_1 = \begin{cases} S1 \text{ if } 0 \le r < 0.514 \\ S2 \text{ if } 0.514 \le r < 1 \end{cases}; \quad ss_2 = \begin{cases} S3 \text{ if } 0 \le r < 0.476 \\ S4 \text{ if } 0.476 \le r < 1 \end{cases}; \quad ss_3 = \begin{cases} S5 \text{ if } 0 \le r < 0.34 \\ S6 \text{ if } 0.34 \le r < 0.66 \\ S7 \text{ if } 0.66 \le r < 1 \end{cases}$$

Similarly, probability functions are built for the resulting available tasks obtained at step 6 of Algorithm 1. Therefore, supposing that the selected subgraphs are S1, S4 and S5, then the available tasks are 1, 2 and 15. The cumulative probability distribution for selecting the next task st to be assigned is the following ($r \epsilon [0,1)$ is a random value):

$$st = \begin{cases} 1 \text{ if } 0 \le r < 0.29 \\ 2 \text{ if } 0.29 \le r < 0.65 \\ 15 \text{ if } 0.65 \le r < 1 \end{cases}$$

5.3 Local Optimization Procedures

Two local optimization procedures based on two neighbourhood search strategies have been developed, aiming at improving the solution generated by the proposed approximate methods. At this point, it is valid to comment that a solution to the problem is represented by a sequence of tasks, which results from orderly assigning the tasks to the workstations.

The following notation is used to describe such search strategies:

m_k Number of workstations required for a given sequence (solution) k
ISq Initial task sequence generated by a given heuristic method
WS Working sequence (the first WS is ISq)
SS Stored sequence (the first SS is ISq)
NS Neighbour sequence
Slk_j Slack (cycle time minus workstation time) of workstation j
α Weight parameter

The local optimization procedures generate the neighbourhood of the working sequence WS by using a transformation or exchange movement. Each exchange k generates a neighbour sequence NS. Then, task are orderly assigned to the workstations resulting in a number of required workstation m_k. If NS improves SS (i.e., it requires fewer workstations), the neighbour sequence becomes the stored sequence SS.

When a neighbour sequence requires the same number of workstations as the store sequence, a secondary objective function [5.1] is used as a tie-breaker. This function gives more importance to solutions that load the first workstations at maximum capacity and the last ones at minimum capacity. To achieve this objective, the weight parameter α of f is set to 10 (it was confirmed that equivalent results can be obtained using $\alpha = 10^e$, where e is an integer greater than 1).

$$max\ f = \sum_{j=1}^{m_k} \alpha^j \cdot Slk_j \qquad [5.1]$$

The local search ends when all feasible exchanges have been made for each task in WS, i.e., when all neighbours have been generated and evaluated. For the next iteration, the stored sequence SS is assigned to the working sequence WS. The whole procedure is repeated until a predetermined computing time has been completed. The final solution is the best of all solutions generated.

Exchange movements

An adaptation of two classical transformations (see, for example, Armentano and Bassi (2006)) has been considered to generate the neighbourhood of a given solution:

a. The exchange of the positions in WS of a pair of tasks.

In this case, the exchange movement tries to exchange the position in the sequence WS of two tasks i and k, provided it is feasible; i.e., the precedence relations among the tasks are maintained. Furthermore, task i and task k should have been assigned to different workstations. When task i and task k belong to the same subgraph s, new neighbour sequences are searched by interchanging s with each one of the remaining subgraphs available for such tasks (which can affect the order of all tasks belonging to such subgraphs).

b. The movement of task i to another position of the working sequence WS (i.e., a task is yielded to a different workstation).

A task i can be moved to the position of task k when the tasks precedence relations are maintained and when task k and task i have been assigned to different workstations. In this case, all tasks between the positions of task i and k including task k are moved in the sequence one position backwards. For each movement, neighbour sequences are generated by interchanging the alternative subgraphs available for the moved task.

When a movement exchange type a is applied the local optimization procedure is regarded as LOP-1; otherwise, it is referred to as LOP-2.

Example 5.3. Exchange movements

The following initial sequence is obtained by applying the heuristic method NT_RPW to the example of Figure 5.1 with a ct=20 (see Table 5.5):

ISq = 2, 1, 3, 4, 5, 6, 7, 8, 10, 9, 11

Let consider transformation a: then a neighbour sequence is generated by interchanging, for example, tasks 2 and 3 since neither task 2 nor task 1 are predecessors of task 3, neither task 1 nor task 3 are successors of task 2 (i.e. precedence constraints are kept), and (as can be seen in Table 5.5) both tasks are assigned to different workstations (task 2 is assigned to workstation I and task 3 to workstation II). Therefore, one of the resulting neighbour sequences is illustrated in Figure 5.2.

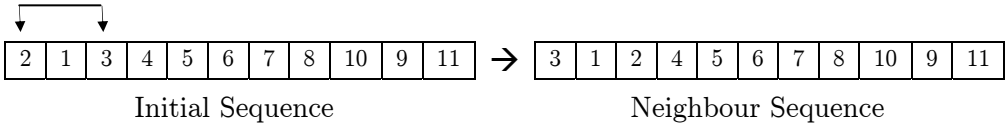

Figure 5.2: Generation of a neighbour sequence using transformation a.

If transformation b is considered, then a neighbour sequence is generated by moving, for example, task 2 to the position of task 3 (see Figure 5.3), which is a feasible movement since neither task 1 nor task 2 are predecessors of task 3. In this case, the neighbour sequence is as follows:

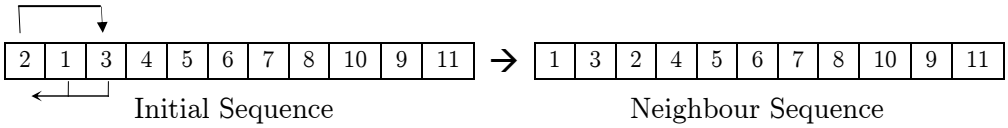

Figure 5.3: Generation of a neighbour sequence using transformation b.

At this point, it is valid to comment on a class of metaheuristic method, called GRASP (*Greedy Randomized Adaptive Search Procedure*), which consists in two phases: a first phase that generates an initial solution by applying a constructive method (as previously described) and a second phase which improves such a solution by applying local optimization procedures. A stochastic mechanism is introduced to generated multiples initial solutions during a given computing time or for a predetermined number of iterations. An adaptation of this approach is distinguished in this work when any of the proposed local optimization procedures are iteratively applied using, in the constructive phase, a multi-pass method. In particular, methods which use probability distributions based on weighted values of various priority rules to select the assembly subgraphs and the assembly tasks can be considered.

5.4 Computational Experiment

To evaluate and compare the performance of the heuristic procedures described in the previous sections, a computational experiment was carried out, for which small-, medium- and large-scale ASALBP instances were considered. Even though small-scaled problems can be solved optimally in significantly low computing times with exact methods (i.e., mathematical programming models) their solutions are considered as a mean to measure the quality of the solutions provided by the proposed heuristic methods.

5.4.1 Experimental conditions

The data sets used in this computational experiment are also based on an adaptation of benchmark SALB Problems that are available at www.assembly-line-balancing.de. The experiment involved the following small-, medium- and large-scale problems: Bowman, Mansor, Mitchell, Buxey, Gunther, Kilbrid,

Hahn, Warnecke, Tonge, Wee-Mag, Lutz3, Arcus2, Bartholdi and Scholl; with 8, 11, 21, 29, 35, 45, 53, 58, 70, 75, 89, 111, 149 and 297 tasks, respectively. Benchmark problems were subdivided into two, three and four subassemblies (involving five, eight and eleven subgraphs, respectively) and from 1 to 5 different cycle time values were considered. Furthermore, to consider alternative assembly processes involving different sets of tasks, new assembly tasks were also added to the problems.

Table 5.8 shows the data sets considered in the computational experiment. It includes the name of the benchmark problem, the cycle time values used, and the number of tasks involved for each group of mutually exclusive assembly subgraphs. It is noteworthy that the first four data sets in Table 5.8 are considered as small-scale problems, the following seven data sets are medium-scale, whereas the remaining sets are considered as large-scale problems. As can be also observed in Table 5.8, small-scale problems involve from one to three cycle time values and from five to eight subgraphs (the dashes in Table 5.8 indicate that those values do not apply for the corresponding problems). A total of 166 (i.e. $1+3+3\cdot2+3\cdot2+5\cdot3\cdot10$) problem instances, involving from 10 to 305 tasks, were solved with each of the 95 heuristics procedures.

All heuristic methods were implemented using C++ programming language, and the experiments were carried out on a Pentium IV, 3 GHz CPU with 512 Mb of RAM.

Table 5.8: Data sets

Problem	**Cycle time values**					**Number of subgraphs**		
						5	**8**	**11**
	$\mathbf{ct_1}$	$\mathbf{ct_2}$	$\mathbf{ct_3}$	$\mathbf{ct_4}$	$\mathbf{ct_5}$	**Number of tasks**		
Bowman	20	-	-	-	-	10	-	-
Mansor	48	62	94	-	-	11	-	-
Mitchell	14	21	35	-	-	21	21	-
Buxey	30	36	54	-	-	29	29	-
Gunther	41	44	49	61	81	37	37	37
Kilbrid	57	79	92	138	184	45	46	48
Hahn	2004	2338	2806	3507	4676	56	56	63
Warnecke	54	62	74	92	111	63	63	67
Tonge	160	176	207	251	320	73	75	75
Wee-Mag	28	33	39	46	56	77	81	83
Lutz3	75	83	97	118	150	93	98	101
Arcus2	5785	6540	7916	9400	11570	115	121	125
Bartholdi	403	470	564	705	805	151	157	160
Scholl	75	83	97	118	150	299	302	305

5.4.2 Analysis of the results

To present the results obtained in the computational experiment, the following notation is used: *NI*: number of the tested instances; *CT*: computing time; *NBS*: number of best solutions obtained; *PBS*: percentage of best solutions obtained; Δ_{max}, Δ_{av}, Δ_{min}, maximal, average and minimal deviation from the best solution *BS* respectively; $T_{\max}$, T_{av}, $T_{\min}$, maximal, average and minimal solution time, respectively. For each problem instance, the relative deviation from the best value Δ is computed, for each heuristic solution *HS*, as follows:

$$\Delta = 100 \cdot \frac{HS - BS}{BS}.$$

The evaluation was based on the number of best solutions provided by all the methods. The best solution for each problem instance, the basis for the comparative analysis, is the best of all solutions found by the compared heuristic methods. For instance, the best solution found by any single-pass method is used to evaluate the efficiency of single-pass methods. Similarly, the best solution found by any multi-pass method is used to evaluate the efficiency of all multi-pass methods. The overall performance of all methods is evaluated by considering the best solution found by the best single-pass heuristic or by the best multi-pass heuristic method.

The application of the 39 single-pass methods (see Table 5.2) implied 6474 computational experiments. On the other hand, the proposed 56 multi-pass procedures (see Tables 5.6 and 5.7) conducted 9296 experiments. Furthermore, two local optimization procedures were applied to each of the proposed heuristic methods, which considering a single *CT* value (60 seconds), entailed 31540 additional computational tests. Finally, to evaluate the effect of different computing times, *CT,* in the percentage of best solutions, 1162 further experiments were realized.

Applying simple-pass methods to solve small-scale problems

Table 5.9 presents the results obtained by using all single-pass methods defined in Table 5.2 to solve small-scale problems (15 test instances). As observed in Table 5.9, the methods *NP–TTS, TT–TTS* and *NT–TTS* significantly outperformed all other methods, achieving the best solutions in 93.3% of the cases, and having the lowest Δ_{max} (12.5%) and Δ_{av} (0.8%). Other methods that had a relatively good performance include *NP–T, NP–TLW, NP–TS, NP–STS, NP–LWTS, TT–T, TT–TLW, TT–TS, TT–STS, TT–LWTS, NT–T, NT–TLW, NT–TS, NT–STS* and *TT–LWTS*, which

provided the best solutions in 66.7% of the cases, having a Δ_{max} of 33.3% and a Δ_{av} between 7.2 and 9.1%. On the other hand, the methods [*SubgraphRule_EW*] performed the worst, generating the best solutions in only 26.7% of the small-scale instances tested, furthermore, Δ_{max} is considerable high (50%). As it could be expected, single-pass-methods require a very low solution time, on average, only 0.001 seconds (maximum 0.002 sec) to solve small-scale problems.

Table 5.9: Results for solving small-scale problems using single-pass methods (NI=16)

	Method	NBS	PBS	Δ_{max}	Δ_{av}	T_{max}	T_{av}
1	NP_RPW	8	53.3	50.0	15.8	0.01	0.001
2	NP_T	10	66.7	33.3	7.4	0.00	0.000
3	NP_EW	4	26.7	50.0	19.5	0.00	0.000
4	NP_LW	8	53.3	50.0	15.8	0.01	0.001
5	NP_N	7	46.7	50.0	16.6	0.00	0.000
6	NP_Sk	8	53.3	50.0	15.8	0.01	0.001
7	NP_TLW	10	66.7	33.3	7.2	0.01	0.001
8	NP_IS	7	46.7	50.0	16.6	0.00	0.000
9	NP_TS	10	66.7	33.3	9.1	0.00	0.000
10	NP_TTS	14	93.3	12.5	0.8	0.00	0.000
11	NP_STS	10	66.7	33.3	8.3	0.00	0.000
12	NP_TSSk	7	46.7	50.0	16.6	0.01	0.001
13	NP_LWTS	10	66.7	33.3	9.1	0.00	0.000
14	TT_RPW	8	53.3	50.0	15.8	0.00	0.000
15	TT_T	10	66.7	33.3	7.4	0.00	0.000
16	TT_EW	4	26.7	50.0	19.5	0.00	0.000
17	TT_LW	8	53.3	50.0	15.8	0.00	0.000
18	TT_N	7	46.7	50.0	16.6	0.00	0.000
19	TT_Sk	8	53.3	50.0	15.8	0.01	0.001
20	TT_TLW	10	66.7	33.3	7.2	0.01	0.001
21	TT_IS	7	46.7	50.0	16.6	0.00	0.000
22	TT_TS	10	66.7	33.3	9.1	0.01	0.001
23	TT_TTS	14	93.3	12.5	0.8	0.01	0.001
24	TT_STS	10	66.7	33.3	8.3	0.01	0.001
25	TT_TSSk	7	46.7	50.0	16.6	0.01	0.002
26	TT_LWTS	10	66.7	33.3	9.1	0.01	0.001
27	NT_RPW	8	53.3	50.0	15.8	0.00	0.000
28	NT_T	10	66.7	33.3	7.4	0.01	0.001
29	NT_EW	4	26.7	50.0	19.5	0.01	0.001
30	NT_LW	8	53.3	50.0	15.8	0.00	0.000
31	NT_N	7	46.7	50.0	16.6	0.01	0.001
32	NT_Sk	8	53.3	50.0	15.8	0.01	0.002
33	NT_TLW	10	66.7	33.3	7.2	0.01	0.001
34	NT_IS	7	46.7	50.0	16.6	0.01	0.001
35	NT_TS	10	66.7	33.3	9.1	0.01	0.001
36	NT_TTS	14	93.3	12.5	0.8	0.01	0.001
37	NT_STS	10	66.7	33.3	8.3	0.01	0.001
38	NT_TSSk	7	46.7	50.0	16.6	0.01	0.001
39	NT_LWTS	10	66.7	33.3	9.1	0.00	0.000
Δ_{min} and $T_{min} = 0$ in all cases							

Applying single-pass methods to solve medium-scale problems

Table 5.10 presents the results obtained by using all single-pass methods to solve medium-scale problems (105 test instances).

Table 5.10: Results for solving medium-scale problems using single-pass methods (NI=105)

	Method	NBS	PBS	Δ_{max}	Δ_{av}	T_{max}	T_{av}
1	NP_RPW	75	71.4	33.3	2.8	0.02	0.01
2	NP_T	83	79.0	33.3	2.2	0.02	0.01
3	NP_EW	35	33.3	33.3	7.3	0.02	0.01
4	NP_LW	69	65.7	33.3	3.1	0.02	0.01
5	NP_N	44	41.9	33.3	6.5	0.02	0.01
6	NP_Sk	59	56.2	33.3	4.2	0.02	0.01
7	NP_TLW	90	85.7	33.3	1.8	0.02	0.01
8	NP_IS	46	43.8	33.3	6.2	0.02	0.01
9	NP_TS	64	61.0	33.3	3.5	0.02	0.01
10	NP_TTS	89	84.8	33.3	2.1	0.02	0.01
11	NP_STS	55	52.4	33.3	4.6	0.02	0.01
12	NP_TSSk	45	42.9	33.3	6.4	0.02	0.01
13	NP_LWTS	70	66.7	33.3	3.2	0.02	0.01
14	TT_RPW	80	76.2	33.3	1.8	0.02	0.01
15	TT_T	88	83.8	25.0	1.5	0.02	0.01
16	TT_EW	36	34.3	33.3	7.1	0.02	0.01
17	TT_LW	74	70.5	33.3	2.6	0.02	0.01
18	TT_N	46	43.8	33.3	6.1	0.02	0.01
19	TT_Sk	60	57.1	33.3	3.9	0.02	0.01
20	TT_TLW	94	89.5	25.0	1.1	0.02	0.01
21	TT_IS	49	46.7	33.3	5.9	0.02	0.01
22	TT_TS	66	62.9	33.3	3.3	0.02	0.01
23	TT_TTS	94	89.5	33.3	1.3	0.02	0.01
24	TT_STS	57	54.3	33.3	4.3	0.02	0.01
25	TT_TSSk	47	44.8	33.3	6.0	0.02	0.01
26	TT_LWTS	72	68.6	33.3	3.0	0.02	0.01
27	NT_RPW	76	72.4	33.3	2.5	0.02	0.01
28	NT_T	83	79.0	33.3	2.2	0.02	0.01
29	NT_EW	35	33.3	33.3	7.3	0.02	0.01
30	NT_LW	69	65.7	33.3	3.1	0.02	0.01
31	NT_N	44	41.9	33.3	6.5	0.02	0.01
32	NT_Sk	59	56.2	33.3	4.2	0.02	0.01
33	NT_TLW	90	85.7	33.3	1.8	0.02	0.01
34	NT_IS	46	43.8	33.3	6.2	0.02	0.01
35	NT_TS	64	61.0	33.3	3.5	0.02	0.01
36	NT_TTS	91	86.7	33.3	1.7	0.02	0.01
37	NT_STS	55	52.4	33.3	4.6	0.02	0.01
38	NT_TSSk	45	42.9	33.3	6.4	0.02	0.01
39	NT_LWTS	70	66.7	33.3	3.2	0.02	0.01
Δ_{min} and $T_{min} = 0$ in all cases							

As observed in Table 5.10, the methods *TT_TLW* and *TT_TTS* performed the best, achieving the best solutions in 89.5% of the cases. Similar results were obtained with the methods *NP_TLW*, *NP_TTS*, *TT_T,* *NT_TLW* and *NT_TTS*, which provided the best solutions in 83.8 to 86.7% of the cases. On the other hand, even they behave slightly better, methods [*SubgraphRule_EW*] again performed the worst, generating best solutions in a maximum of only 34.3% (36 out of 105) of the medium-scale instances tested. Other methods with low performance include *NP_N*, *NP_IS*, *NP_TSSk*, *TT_N*, *NT_N*, *NT_IS* and *NT_TSSk*, which all provided best solutions at most in 43.8% of the cases. Table 5.10 also shows that although Δ_{av} is small, for most problems Δ_{max} is rather high (i.e. 33.3%).

Regarding solution time, single-pass methods required an average of only 0.01 seconds (maximum 0.02 sec) to solve medium-scale problems.

Applying single-pass methods to solve large-scale problems

Table 5.11 shows the results obtained using all single-pass methods to solve large-scale problems (45 test instances). In this case, method *TT_RPW* performed the best, which generated best solutions in 88.9% of the problems solved. In general, methods [*SubgraphRule_RPW*] had the best performance: both *NP_RPW* and *NT_RPW* found the best solutions in 77.8% of the cases. These results indicate much higher performance of these methods for large-scale problems than for small- and medium-scale problems (e.g., the *PBS* of *TT_RPW* for large-scale problems is 88.9%, whereas for medium-scale it is 76.2 and only 53.3% for small-scale problems). Methods [*SubgraphRule_LW*] presented a similar high performance: *TT_LW*, *NP_LW* and *NT_LW* generated best solutions in 71.1, 75.6%, 71.1%, respectively. Good solutions were also obtained with *TT_TS,* *TT_TTS* and *TT_LWTS*, which each provided the best solutions for more than 71% of the cases. For large-scale problems, methods [*SubgraphRule_EW*] again performed poorly (i.e. maximum *PBS*=26.7%). Furthermore, as can be observed in Table 5.11, Δ_{av}, and particularly Δ_{max}, were much smaller for large-scale problems; i.e., Δ_{max} was 12.5% for most methods, and the maximum was 22.2% (which for medium- and small-scale problems was 33.3% and 50%, respectively). On the other hand, solution time is also relatively small for large-scale problems: averaged T_{av}=0.07 seconds.

Table 5.11: Results for solving large-scale problems using single-pass methods (NI=45)

	Method	NBS	PBS	Δ_{max}	Δ_{av}	$\mathbf{T}_{max}$	$\mathbf{T}_{av}$
1	NP_RPW	35	77.8	12.5	1.9	0.6	0.07
2	NP_T	26	57.8	12.5	2.6	0.6	0.07
3	NP_EW	12	26.7	22.2	6.7	0.6	0.07
4	NP_LW	32	71.1	12.5	2.1	0.6	0.07
5	NP_N	12	26.7	22.2	6.6	0.7	0.07
6	NP_Sk	28	62.2	12.5	2.4	0.6	0.07
7	NP_TLW	29	64.4	12.5	1.7	0.6	0.07
8	NP_IS	19	42.2	22.2	5.2	0.6	0.07
9	NP_TS	30	66.7	12.5	2.4	0.6	0.07
10	NP_TTS	30	66.7	12.5	2.2	0.6	0.07
11	NP_STS	23	51.1	12.5	2.7	0.6	0.07
12	NP_TSSk	12	26.7	22.2	6.6	0.6	0.07
13	NP_LWTS	30	66.7	12.5	2.3	0.6	0.07
14	TT_RPW	40	88.9	12.5	1.0	0.7	0.09
15	TT_T	29	64.4	12.5	2.1	0.8	0.09
16	TT_EW	12	26.7	15.0	6.0	0.8	0.09
17	TT_LW	34	75.6	12.5	1.5	0.8	0.09
18	TT_N	13	28.9	20.0	5.6	0.8	0.09
19	TT_Sk	31	68.9	12.5	1.8	0.8	0.09
20	TT_TLW	28	62.2	12.5	2.2	0.8	0.09
21	TT_IS	20	44.4	12.5	4.1	0.8	0.09
22	TT_TS	32	71.1	12.5	1.5	0.8	0.09
23	TT_TTS	33	73.3	12.5	1.4	0.8	0.09
24	TT_STS	22	48.9	12.5	2.9	0.8	0.09
25	TT_TSSk	13	28.9	20.0	5.6	0.8	0.09
26	TT_LWTS	33	73.3	12.5	1.7	0.8	0.09
27	NT_RPW	35	77.8	12.5	1.9	0.6	0.07
28	NT_T	26	57.8	12.5	2.6	0.6	0.07
29	NT_EW	12	26.7	22.2	6.7	0.6	0.07
30	NT_LW	32	71.1	12.5	2.1	0.7	0.07
31	NT_N	12	26.7	22.2	6.6	0.7	0.07
32	NT_Sk	28	62.2	12.5	2.4	0.6	0.07
33	NT_TLW	29	64.4	12.5	1.7	0.6	0.07
34	NT_IS	19	42.2	22.2	5.2	0.6	0.07
35	NT_TS	30	66.7	12.5	2.4	0.6	0.07
36	NT_TTS	30	66.7	12.5	2.2	0.6	0.07
37	NT_STS	23	51.1	12.5	2.7	0.6	0.07
38	NT_TSSk	12	26.7	22.2	6.6	0.6	0.07
39	NT_LWTS	30	66.7	12.5	2.3	0.6	0.07

Δ_{min} and $T_{min} = 0$ in all cases

Single-pass methods – Overall results

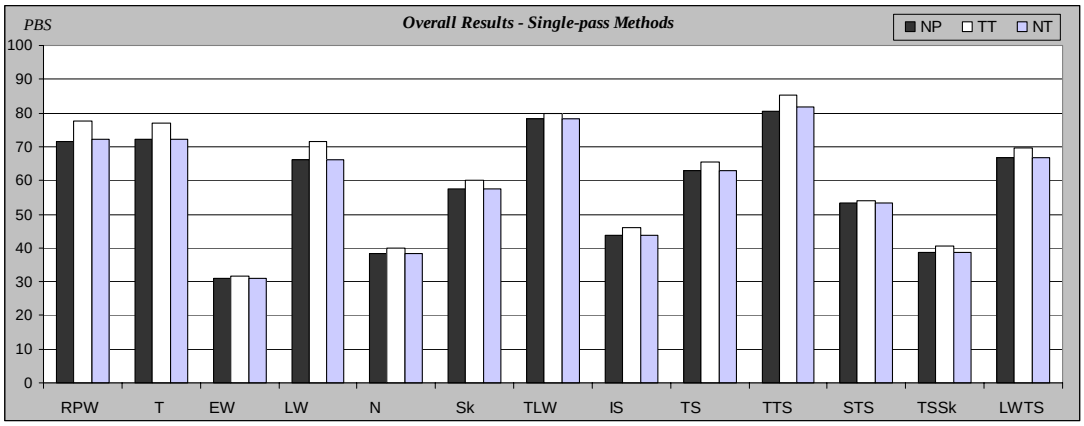

Figure 5.4: Overall results of *PBS* for single-pass methods.

Figure 5.4 summarizes the *PBS* for all single-pass methods based on a total of 166 small-, medium- and large-scale problems. The results are grouped by task priority rule. This Figure reveals that the methods perform similarly for the same rule using different criteria for subgraphs. Additionally, it can be observed in Figure 5.4 that methods [*SubgraphRule_TSS*] provided the highest percentage of best solutions. Therefore, *TT* is, on average, the best rule for selecting subgraphs; being *TT_TTS* the best performing of all single-pass methods. Other methods with similar good performance are [*SubgraphRule_RPW*], [*SubgraphRule_T*], [*SubgraphRule_LW*], [*SubgraphRule_TLW*] and [*SubgraphRule_LWTS*]. Furthermore, as seen in the partial results, [*SubgraphRule_EW*] are the worst of all proposed methods, which were able to find best solutions in less than 32% of the problems solved. Other families of methods with similarly poor results include [*SubgraphRule_N*], [*SubgraphRule_IS*] and [*SubgraphRule_TSSk*].

Improving the solution provided by single-pass methods

Table 5.12 shows the results obtained by applying the proposed local optimization procedures to improve the solutions provided by single-pass methods, considering all 166 data sets and a computing time of 60 seconds (the base of the comparison). It includes the number and percentage of solutions improved in k workstations (*NSkstat* and *PSkstat*, respectively) with both LOP-1 and LOP-2 (in this case, a maximum of only one workstation improvement was obtained; therefore, *NSkstat* and *PSkstat*=0 for $k \geq 2$). As can be observed in Table 5.12, the highest improvements were obtained, as expected, for methods with low performance, namely [*SubgraphRule_EW*], [*SubgraphRule_N*], [*SubgraphRule_IS*] and [*SubgraphRule_TSSk*].

Table 5.12: Improving the solutions provided by single-pass methods (NI=166)

Method	LOP-1		LOP-2	
	NS1stat	PS1stat	NS1stat	PS1stat
NP_RPW	2	1.2	2	1.2
NP_T	0	0.0	6	3.6
NP_EW	34	20.6	59	35.8
NP_LW	7	4.2	10	6.1
NP_N	32	19.4	44	26.7
NP_Sk	14	8.5	17	10.3
NP_TLW	0	0.0	4	2.4
NP_IS	25	15.2	32	19.4
NP_TS	9	5.5	12	7.3
NP_TTS	1	0.6	2	1.2
NP_STS	11	6.7	18	10.9
NP_TSSk	35	21.2	45	27.3
NP_LWTS	5	3.0	7	4.2
TT_RPW	1	0.6	2	1.2
TT_T	0	0.0	5	3.0
TT_EW	41	24.8	58	35.2
TT_LW	7	4.2	10	6.1
TT_N	38	23.0	45	27.3
TT_Sk	14	8.5	18	10.9
TT_TLW	0	0.0	2	1.2
TT_IS	23	13.9	32	19.4
TT_TS	13	7.9	8	4.8
TT_TTS	0	0.0	2	1.2
TT_STS	10	6.1	16	9.7
TT_TSSk	41	24.8	46	27.9
TT_LWTS	5	3.0	9	5.5
NT_RPW	2	1.2	2	1.2
NT_T	0	0.0	6	3.6
NT_EW	34	20.6	59	35.8
NT_LW	7	4.1	10	6.1
NT_N	32	19.4	44	26.7
NT_Sk	14	8.5	17	10.3
NT_TLW	0	0.0	4	2.4
NT_IS	25	15.2	32	19.4
NT_TS	9	5.5	12	7.3
NT_TTS	1	0.6	2	1.2
NT_STS	11	6.7	18	10.9
NT_TSSk	35	21.2	45	27.3
NT_LWTS	5	3.0	7	4.2

Table 5.12 reveals that solutions could be improved up to 24.8 and 35.8% (on average, 8.4 and 12%) with LOP-1 and LOP-2, respectively. However, better results were obtained when LOP-2 (i.e. an exchange movement of one task that implies its assignment to a different workstation) was used; which outperformed most methods applying LOP-1 (see Figure 5.5).

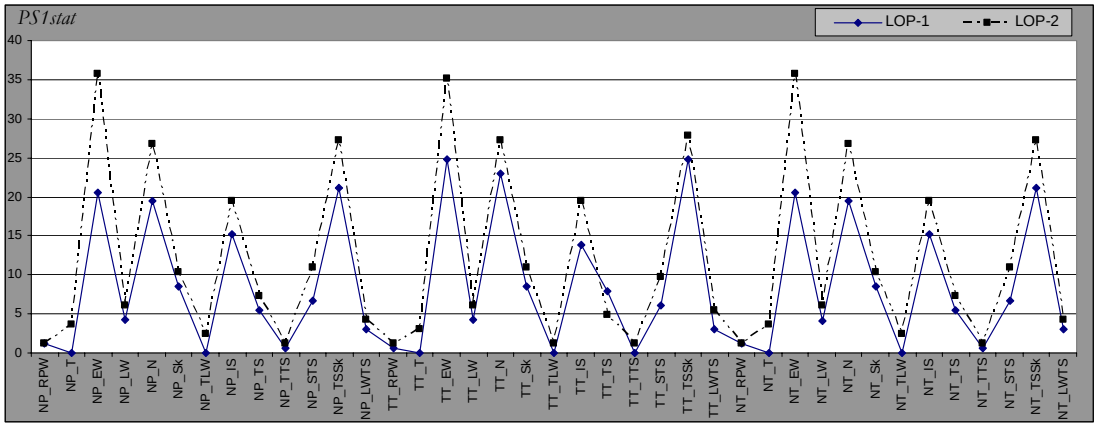

Figure 5.5: Applying local optimization procedures and single-pass methods.

On the other hand, Figure 5.5 also shows that methods employing different local search procedures behave very similarly when the same heuristic method is used to build the initial solution.

Applying non-weighted multi-pass methods – class a, b and c

Table 5.13 presents the results obtained by using all multi-pass methods defined in Table 5.6 to solve small-, medium- and large-scale problems. Since multi-pass methods generate multiple solutions within a given computing time, the average solution time required by single-pass methods to solve the tested instances was employed as stopping criteria: 0.1 seconds (which, for simplicity, is the average value, 0.069, rounded up to a single decimal number). In this way, all single-pass and multi-pass methods can be compared evenly. The impact of considering longer computing times on the solution quality is discussed later in this section.

Regarding ***small-scale problems***, the best results were obtained with methods *RS_TTS* and *RS_TS*, which found best solutions in 100% of the cases. Furthermore, methods [*SubgraphRule_RT*] were able to provide best solutions in 80% of the cases. Method *RS_EW* performed the worst (i.e. *PBS*=26.7%). For most methods Δ_{max} and Δ_{av} are considerably large (i.e. maximum Δ_{max} and Δ_{av} is 50 and 21.8%, respectively).

Considering ***medium-scale problems***, method *RS_TTS* performed the best, providing best solutions in 87.6%. Good results were also obtained with methods *RS_TLW*, *RS_T* and RS_RPW that were able to find the best solutions in 80, 76.2 and 67.6% of the cases, respectively. Although Δ_{av} is much smaller than for small-scale ones, for the majority of problems Δ_{max} is considerably large: 33.3%. On the other hand, the worst performing method was *RS_EW* (i.e. *PBS*=31.4%).

Table 5.13: Performance of non-weighted multi-pass methods, CT=0.1

Method	Small (NI=16)				Medium (NI=105)				Large (NI=45)			
	NBS	PBS	Δ_{max}	Δ_{av}	NBS	PBS	Δ_{max}	Δ_{av}	NBS	PBS	Δ_{max}	Δ_{av}
RS_RPW	6	40.0	50.0	19.3	71	67.6	33.3	2.1	42	93.3	14.3	0.4
RS_T	13	86.7	33.3	4.4	80	76.2	25.0	2.1	34	75.6	14.3	1.5
RS_EW	4	26.7	50.0	21.8	33	31.4	33.3	7.3	15	33.3	15.0	5.1
RS_LW	6	40.0	50.0	19.3	66	62.9	33.3	2.9	38	84.4	14.3	1.1
RS_N	5	33.3	50.0	20.2	52	49.5	33.3	5.5	24	53.3	14.3	3.3
RS_Sk	6	40.0	50.0	19.3	56	53.3	33.3	4.0	34	75.6	14.3	1.4
RS_TLW	13	86.7	33.3	4.4	84	80.0	25.0	1.8	31	68.9	14.3	1.6
RS_IS	5	33.3	50.0	20.2	50	47.6	33.3	5.7	28	62.2	14.3	2.9
RS_TS	8	53.3	33.3	12.7	62	59.0	33.3	3.7	35	77.8	12.5	1.1
RS_TTS	15	100	0.0	0.0	92	87.6	33.3	1.6	37	82.2	14.3	1.0
RS_STS	8	53.3	33.3	11.8	52	49.5	33.3	4.5	25	55.6	14.3	2.1
RS_TSSk	5	33.3	50.0	20.2	52	49.5	33.3	5.4	24	53.3	14.3	3.3
RS_LWTS	8	53.3	33.3	12.7	63	60.0	33.3	3.7	38	84.4	12.5	0.9
NP_RT	12	80.0	33.3	4.4	61	58.1	33.3	4.4	14	31.1	18.5	5.0
TT_RT	12	80.0	33.3	4.4	61	58.1	33.3	4.2	17	37.8	14.3	4.3
NT_RT	12	80.0	33.3	4.4	61	58.1	33.3	4.4	14	31.1	18.5	5.0
RS_RT	15	100	0.0	0.0	59	56.2	33.3	3.7	17	37.8	14.3	4.1
Δ_{min} = 0 in all cases												

For ***large-scale problems***, methods *RS_RPW* performed the best, providing the best solutions in 93.3%. Good results were also obtained with methods *RS_LW*, *RS_LWTS*, *RS_TTS*, *RS_TS*, *RS_T* and *RS_Sk*, which found the best solutions in 84.4, 84.4, 82.2, 77.8, 75.6 and 75.6% of the cases, respectively. In contrast, class *b* methods (i.e. [*RuleSubgraphs_RT*]) performed worse for large-scale problems than for small- and medium-scale problems, which only provided the best solutions in 37.8% or less of the cases. Such results could be expected since a larger number of tasks need to be assigned; therefore, only few iterations of the heuristic procedure can be performed. Furthermore, by fixing the subgraphs at the beginning of the procedure, the best combination of subgraphs may remain unexplored. Bad results were also obtained with methods *RS_EW* and *RS_RT*, which provided the best solutions for only 33.3 and 37.8%, respectively. Table 5.13 also reveals that the results imply a much smaller Δ_{max} (i.e. 18.5% or less).

Non-weighted multi-pass methods – Overall results

Figure 5.6 summarizes the overall results obtained for multi-pass methods class-*a*: [*Random_TaskRule*], class-*b*: [*SubgraphRule_Random*] and class-*c*: [*Random_Random*] used to solve all data sets (i.e. 166 ASALBP instances). As can be observed in Figure 5.6, the highest-performing procedure was a class-*a* method: *RS_TTS* which obtained the best solutions in 87.3% of the cases. Similar results were obtained with methods *RS_TLW*, *RS_T, RS_RPW*, which were able to find the best solutions in more that 72.1% of the cases. In average, *RS_EW* is the worst performing method (i.e. *PBS*=31.5%), which confirms the condition of *EW* as a very inefficient rule for selecting tasks in an ASALBP. Other methods with poor results include *RS_N*, *RS_IS* and *RS_TSSk*, which provided the best solutions in less than 50% of the cases.

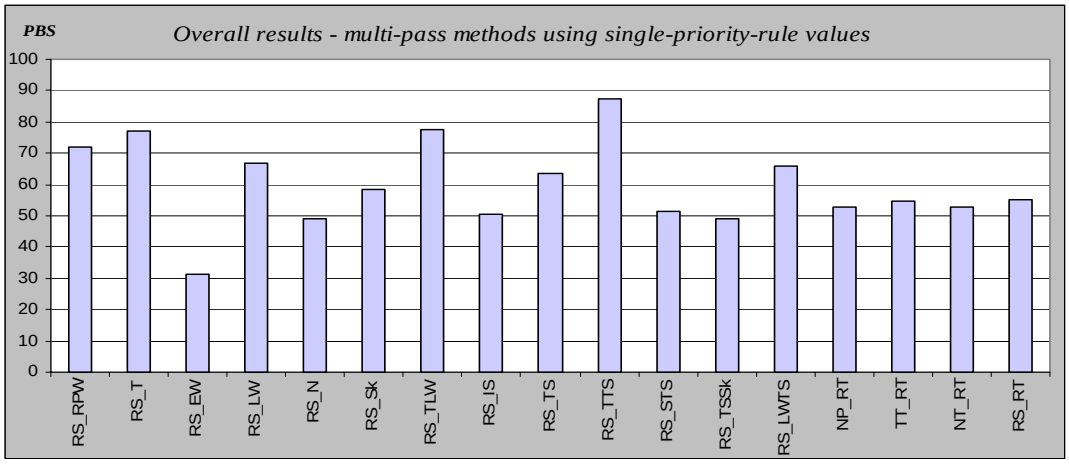

Figure 5.6: Overall results for non-weighted multi-pass methods.

Improving the solution provided by non-weighted multi-pass methods

Table 5.14 shows the results obtained by applying the proposed local optimization procedures to improve the solutions provided by non-weighted multi-pass methods (class *a*, *b* and *c*), taking into account all available data sets. It includes the number and percentage of solutions improved with both LOP-1 and LOP-2. As can be observed in Table 5.14, solutions were improved in up to 2 workstations with both methods. Highest improvements were obtained for methods with low performance (see Figure 5.6): *RS_EW, RS_N, RS_IS*, *RS_STS*, and *RS_TSSk*. Furthermore, in all cases LOP-2 outperformed LOP-1: for the former averaged *PS1st1at* and *PS2stat* were 8 and 0.3%, respectively; whereas for LOP-1 these values were 3.8 and 0.1%, respectively.

Table 5.14: Improving the solutions provided by non-weighted multi-pass methods (NI=166)

Method	LOP-1				LOP-2			
	NS1stat	PS1stat	NS2stat	PS2stat	NS1stat	PS1stat	NS2stat	PS2stat
RS_RPW	0	0	0	0	1	0.6	1	0.6
RS_T	0	0	0	0	7	4.2	0	0
RS_EW	26	15.8	2	1.2	43	26.1	4	2.4
RS_LW	4	2.4	0	0	5	3.0	0	0
RS_N	11	6.7	0	0	22	13.3	1	0.6
RS_Sk	9	5.5	0	0	11	6.7	0	0
RS_TLW	0	0	0	0	5	3.0	0	0
RS_IS	11	6.7	0	0	22	13.3	0	0
RS_TS	5	3.0	0	0	7	4.2	0	0
RS_TTS	0	0	0	0	0	0.0	1	0.6
RS_STS	1	0.6	0	0	14	8.5	0	0
RS_TSSk	11	6.7	0	0	22	13.3	1	0.6
RS_LWTS	4	2.4	0	0	6	3.6	0	0
NP_RT	5	3.0	0	0	15	9.1	0	0
TT_RT	10	6.1	0	0	14	8.5	0	0
NT_RT	5	3.0	0	0	15	9.1	0	0
RS_RT	4	2.4	0	0	16	9.7	1	0.6

Applying weighted multi-pass methods – class d

Table 5.15 presents the results obtained by all weighted multi-pass methods (defined in Table 5.7) to solve small-, medium- and large-scale problems.

Small-scale problems: 23% of the weighted multi-pass methods were able to find the best solutions in 100% of the problems solved: W-[$SubgraphRule\_RPW$], W-[$SubgraphRule\_TLW$], W-[$SubgraphRule\_TTS$], W-[$SubgraphRule\_LWTS$], W-NP_Sk, W-TT_STS and W-NT_STS. In general, most methods had a very high performance, producing, on average, 91.1% of best solutions. The worth method of all was W-[NP_N] which found best solutions in 53.3% of the cases. For most methods Δ_{max} and Δ_{av} are considerably low, which are much smaller values comparing with other methods solving small-scale problems.

Medium-scale problems: best performing methods for medium-scale problem are W-[$SubgraphRule\_T$], which provided the best solutions in 81.9 to 84.8% of the cases. Good results were also obtained with methods W-[$SubgraphRule\_RPW$], W-[$SubgraphRule\_TS$], W-[$SubgraphRule\_TTS$], which were able to find the best solutions for more that 73.3% of the cases. Methods W-[$SubgraphRule\_TSSk$] and W-[$SubgraphRule\_LWTS$] performed the worst: maximum PBS of 47.6%. For medium-scale problems Δ_{ave} is larger than for small-scale ones; furthermore, for most problems Δ_{max} is considerably large: 33.3%.

Large-scale problems: best performing methods for large-scale problems are W-[*SubgraphRule_TS*] which provided the best solutions from 88.9 to 93.3% of the cases. Similar good results were obtained by applying W-[*SubgraphRule_RPW*] methods, which generated the best solutions for more than 75.6% of the cases. On average weighted-multi-pass methods worked on large-scale problems similarly to on medium-scale ones; however, for the former the methods implied a much smaller Δ_{max} (maximum=11.8% and averaged=8%). The worst performing methods for large-scale problems were W-[*SubgraphRule_LWTS*], with *PBS* equal to 44.4%.

Table 5.15: Results of weighted multi-pass methods, CT=0.1

Method	SMALL (NI=16)				MEDIUM (NI=105)				LARGE (NI=45)			
	NBS	PBS	Δ_{max}	Δ_{av}	NBS	PBS	Δ_{max}	Δ_{av}	NBS	PBS	Δ_{max}	Δ_{av}
W-NP_RPW	15	100	0.0	0.0	80	76.2	25.0	1.6	34	75.6	6.7	0.8
W-NP_T	14	93.3	12.5	0.8	86	81.9	33.3	1.8	31	68.9	8.0	1.2
W-NP_EW	13	86.7	12.5	1.7	58	55.2	33.3	4.1	30	66.7	8.0	1.6
W-NP_LW	14	93.3	12.5	0.8	61	58.1	33.3	3.8	25	55.6	8.0	2.0
W-NP_N	8	53.3	33.3	11.7	64	61.0	33.3	3.7	24	53.3	8.0	2.0
W-NP_Sk	15	100	0.0	0.0	61	58.1	33.3	3.9	25	55.6	8.0	2.0
W-NP_TLW	15	100	0.0	0.0	58	55.2	33.3	4.1	28	62.2	8.0	1.6
W-NP_IS	12	80.0	33.3	4.9	70	66.7	33.3	2.2	26	57.8	8.0	1.7
W-NP_TS	13	86.7	20.0	2.7	80	76.2	33.3	1.7	42	93.3	4.8	0.2
W-NP_TTS	15	100	0.0	0.0	77	73.3	33.3	2.6	27	60.0	8.0	1.6
W-NP_STS	14	93.3	12.5	0.8	64	61.0	33.3	3.7	24	53.3	8.0	1.9
W-NP_TSSk	13	86.7	20.0	2.7	50	47.6	33.3	4.7	24	53.3	8.0	2.1
W-NP_LWTS	15	100	0.0	0.0	49	46.7	33.3	4.8	20	44.4	11.8	2.4
W-TT_RPW	15	100	0.0	0.0	79	75.2	33.3	2.0	35	77.8	4.8	0.7
W-TT_T	14	93.3	20.0	1.3	88	83.8	33.3	1.7	30	66.7	8.0	1.4
W-TT_EW	12	80.0	20.0	3.0	58	55.2	33.3	4.1	30	66.7	8.0	1.6
W-TT_LW	14	93.3	20.0	1.3	61	58.1	33.3	3.8	25	55.6	8.0	2.1
W-TT_N	11	73.3	33.3	6.4	65	61.9	33.3	3.6	23	51.1	8.0	2.0
W-TT_Sk	13	86.7	20.0	2.2	61	58.1	33.3	3.9	25	55.6	8.0	2.1
W-TT_TLW	15	100	0.0	0.0	59	56.2	33.3	4.0	28	62.2	8.0	1.6
W-TT_IS	13	86.7	20.0	2.7	71	67.6	11.1	2.0	27	60.0	8.0	1.6
W-TT_TS	13	86.7	20.0	2.7	78	74.3	14.3	1.5	40	88.9	7.7	0.4
W-TT_TTS	15	100	0.0	0.0	78	74.3	33.3	2.5	27	60.0	8.0	1.6
W-TT_STS	15	100	0.0	0.0	66	62.9	33.3	3.6	24	53.3	8.0	2.0
W-TT_TSSk	13	86.7	20.0	2.7	50	47.6	33.3	4.7	24	53.3	8.0	2.3
W-TT_LWTS	15	100	0.0	0.0	50	47.6	33.3	4.7	20	44.4	11.8	2.6
W-NT_RPW	15	100	0.0	0.0	80	76.2	14.3	1.4	35	77.8	7.7	0.7
W-NT_T	14	93.3	20.0	1.3	89	84.8	33.3	1.9	30	66.7	8.0	1.4
W-NT_EW	12	80.0	20.0	3.0	59	56.2	33.3	4.0	29	64.4	8.0	1.7
W-NT_LW	14	93.3	20.0	1.3	61	58.1	33.3	3.9	24	53.3	8.0	2.3
W-NT_N	11	73.3	33.3	6.4	64	61.0	33.3	3.6	23	51.1	8.0	2.0
W-NT_Sk	14	93.3	20.0	1.3	61	58.1	33.3	3.9	24	53.3	8.0	2.3
W-NT_TLW	15	100	0.0	0.0	59	56.2	33.3	4.0	28	62.2	8.0	1.6
W-NT_IS	13	86.7	20.0	2.7	71	67.6	11.1	2.0	27	60.0	8.0	1.6
W-NT_TS	13	86.7	20.0	2.7	79	75.2	14.3	1.4	40	88.9	7.7	0.4
W-NT_TTS	15	100	0.0	0.0	77	73.3	33.3	2.5	27	60.0	8.0	1.6
W-NT_STS	15	100	0.0	0.0	66	62.9	33.3	3.7	24	53.3	8.0	2.0
W-NT_TSSk	13	86.7	20.0	2.7	50	47.6	33.3	4.7	25	55.6	8.0	2.1
W-NT_LWTS	15	100	0.0	0.0	49	46.7	33.3	4.7	20	44.4	11.8	2.6

Weighted Multi-pass methods – Overall results

Figure 5.7 summarizes the overall results (averaged *PBS*) obtained for multi-pass methods class-*d*, considering all data sets (166 tested problems). As can be observed in Figure 5.7, best performing methods were W-[*SubgraphRule_TS*], in particular *W-NP_TS*, which provided best solutions in more than 80% of the problems solved. Methods W-[*SubgraphRule_RPW*], W-[*SubgraphRule_T*] and W-[*SubgraphRule_TTS*] also performed well, all of which were able to find the best solutions in more than 70% of the cases.

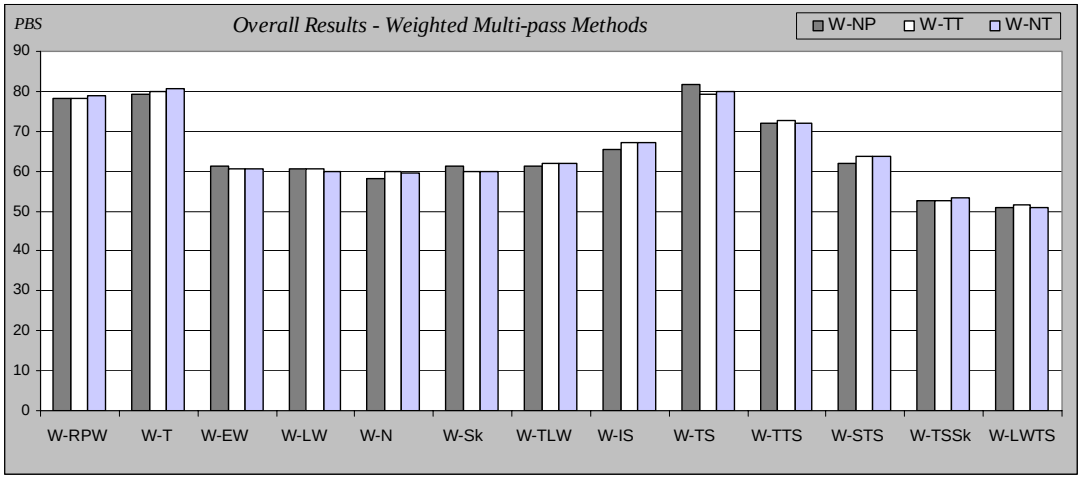

Figure 5.7: Overall results for weighted multi-pass methods

Improving the solution provided by weighted multi-pass methods

Table 5.16 presents the results obtained by applying the proposed local optimization procedures to solve all data sets, being the solutions generated by applying multi-pass methods that used weighted values of the priority rules to select both subgraphs and tasks. As can be observed in Table 5.16, solutions provided by all methods were improved in one workstation by both local optimization procedures: minimum improvement obtained with LOP-1 and LOP-2 is 2.4 and 3.6%, respectively. Figure 5.8 shows that LOP-2 outperformed in all cases LP0-1: average and maximum *PS1stat* are 10.4 and 20%, respectively; whereas for LOP-1 these values are 6.1 and 10.9%, respectively.

Table 5.16: Improving the solutions provided by weighted multi-pass methods (NI=166)

Method	LOP-1		LOP-2	
	NS1stat	PS1stat	NS1stat	PS1stat
W-NP_RPW	6	3.6	7	4.2
W-NP_T	4	2.4	11	6.7
W-NP_EW	14	8.5	21	12.7
W-NP_LW	12	7.3	19	11.5
W-NP_N	10	6.1	17	10.3
W-NP_Sk	12	7.3	17	10.3
W-NP_TLW	10	6.1	18	10.9
W-NP_IS	7	4.2	15	9.1
W-NP_TS	5	3.0	7	4.2
W-NP_TTS	6	3.6	11	6.7
W-NP_STS	8	4.8	15	9.1
W-NP_TSSk	14	8.5	25	15.2
W-NP_LWTS	18	10.9	30	18.2
W-TT_RPW	11	6.7	13	7.9
W-TT_T	5	3.0	10	6.1
W-TT_EW	15	9.1	21	12.7
W-TT_LW	11	6.7	18	10.9
W-TT_N	13	7.9	21	12.7
W-TT_Sk	11	6.7	18	10.9
W-TT_TLW	9	5.5	18	10.9
W-TT_IS	6	3.6	13	7.9
W-TT_TS	4	2.4	9	5.5
W-TT_TTS	8	4.8	13	7.9
W-TT_STS	8	4.8	17	10.3
W-TT_TSSk	17	10.3	27	16.4
W-TT_LWTS	16	9.7	32	19.4
W-NT_RPW	10	6.1	10	6.1
W-NT_T	4	2.4	10	6.1
W-NT_EW	16	9.7	21	12.7
W-NT_LW	12	7.3	18	10.9
W-NT_N	12	7.3	20	12.1
W-NT_Sk	11	6.7	17	10.3
W-NT_TLW	9	5.5	18	10.9
W-NT_IS	6	3.6	13	7.9
W-NT_TS	4	2.4	9	5.5
W-NT_TTS	8	4.8	13	7.9
W-NT_STS	8	4.8	17	10.3
W-NT_TSSk	16	9.7	27	16.4
W-NT_LWTS	17	10.3	33	20.0

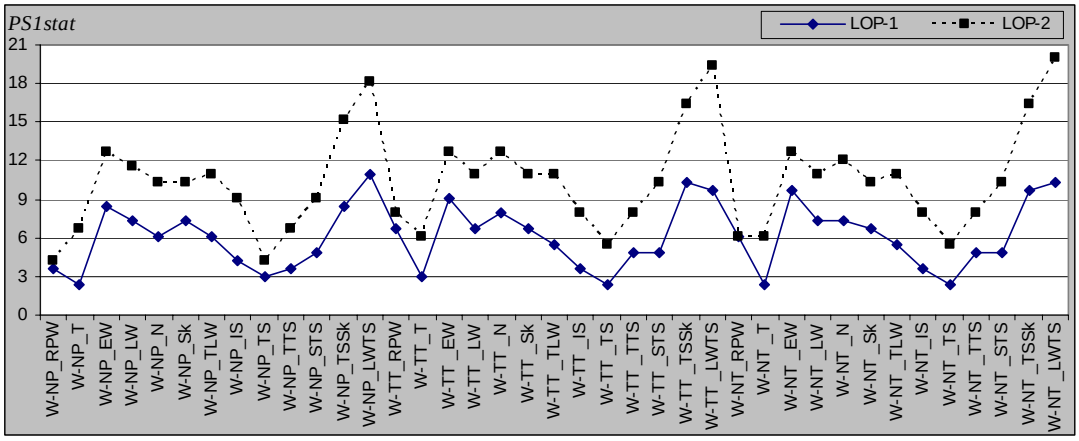

Figure 5.8: Applying local optimization procedures and weighted multi-pass methods

Figure 5.8 that presents the behaviour of applying both optimization procedures and weighted multi-pass methods, shows that methods employing different local search procedures behave very similarly when the same heuristic method is used to build the initial solution (similar results are obtained when considering single-pass methods – as shown in Figure 5.5.).

Joint evaluation of the quality of the obtained solutions

To evaluate the quality of the obtained solutions, and therefore, the efficiency of all heuristic methods, a set of 44 test-ASALBP instances, for which the optimal solution is known, has been considered. This set includes an adaptation of the problems of Bowman, Mansor, Mitchell, Buxey, Gunther, Kilbrid, Hahn and Tonge, with 10, 11, 21, 29, 41, 45, 56 and 70 tasks, respectively; from 1 to 5 different cycle time values and 5, 8 and 11 subgraphs were considered. The optimal solution of each tested problem instance is compared with the solution obtained with each of the heuristic methods. In this way, the percentage of obtained solutions equal to the optimal solution (i.e. *POS*) and the average deviation from the optimum are computed.

Single-pass methods

Table 5.17 presents the analysis of the results for single-pass methods. It includes the *POS*, the percentage of solutions (%*S*) with 1 and 2 workstations deviation from the optimum; and $Opt\Delta_{min}$, $Opt\Delta_{ave}$ and $Opt\Delta_{max}$, which are minimum, average and maximum deviation from the optimum, respectively.

Table 5.17: Solution quality evaluation for single-pass methods (NI=44)

Method	POS	%S with difference from optimum		Deviation from optimum	
		1 station	2 stations	$Opt\Delta_{max}$	$Opt\Delta_{ave}$
NP_RPW	47.7	52.3	0	50.0	11.6
NP_T	50.0	45.5	4.5	33.3	9.9
NP_EW	29.5	63.6	6.8	50.0	14.6
NP_LW	47.7	52.3	0	50.0	11.6
NP_N	43.2	54.5	2.3	50.0	12.2
NP_Sk	45.5	54.5	0	50.0	12.2
NP_TLW	50.0	45.5	4.5	33.3	9.8
NP_IS	40.9	56.8	2.3	50.0	13.0
NP_TS	50.0	50.0	0	33.3	9.9
NP_TTS	59.1	40.9	0	33.3	7.1
NP_STS	47.7	50.0	2.3	33.3	9.9
NP_TSSk	43.2	54.5	2.3	50.0	12.2
NP_LWTS	50.0	50.0	0	33.3	9.9
TT_RPW	52.3	47.7	0	50.0	10.8
TT_T	52.3	47.7	0	33.3	9.1
TT_EW	29.5	68.2	2.3	50.0	14.1
TT_LW	52.3	47.7	0	50.0	10.8
TT_N	47.7	50.0	2.3	50.0	11.4
TT_Sk	47.7	52.3	0	50.0	12.0
TT_TLW	52.3	47.7	0	33.3	9.0
TT_IS	43.2	56.8	0	50.0	12.4
TT_TS	52.3	47.7	0	33.3	9.7
TT_TTS	68.2	31.8	0	33.3	5.2
TT_STS	50.0	47.7	2.3	33.3	9.6
TT_TSSk	47.7	50.0	2.3	50.0	11.4
TT_LWTS	52.3	47.7	0	33.3	9.7
NT_RPW	47.7	52.3	0	50.0	11.6
NT_T	50.0	45.5	4.5	33.3	9.9
NT_EW	29.5	63.6	6.8	50.0	14.6
NT_LW	47.7	52.3	0	50.0	11.6
NT_N	43.2	54.5	2.3	50.0	12.2
NT_Sk	45.5	54.5	0	50.0	12.2
NT_TLW	50.0	45.5	4.5	33.3	9.8
NT_IS	40.9	56.8	2.3	50.0	13.0
NT_TS	50.0	50.0	0	33.3	9.9
NT_TTS	63.6	36.4	0	33.3	6.0
NT_STS	47.7	50.0	2.3	33.3	9.9
NT_TSSk	43.2	54.5	2.3	50.0	12.2
NT_LWTS	50.0	50.0	0	33.3	9.9
$Opt\Delta_{min} = 0$ in all cases					

As can be seen in Table 5.17, single-pass methods with larger POS were [$SubgraphRule\_TT$], in particular TT_TTS which was able to find the optimal solution for 68.2% of the instances solved (this method was the one that performed the best in the computing experiment involving the 166 test instances –see Figure 5.4). Furthermore, method TT_TTS implied a minimum, average, and maximum deviation from the optimum of 0, 5.2 and 33%, respectively. These results indicate that good solutions can be expected by applying method TT_TTS. On the other hand, single-pass methods

generated solutions with one and, for 46% of the methods, two workstations deviation from the optimal solution. Although, average deviation from the optimum is relatively low, for most methods, maximum deviation is high. The analysis of the results also revealed that methods with the lowest performance are [*SubgraphRule_EW*], which were able to obtain the optimal solution for only 29.5% of the problems solved (similar result was obtained when working with all data sets, i.e., 166 problem instances).

Non-weighted multi-pass methods

Table 5.18 presents the analysis of the results for non-weighted multi-pass methods (i.e. methods based on single priority rule values and random choice). The best performance was recorded for method *RS_TTS* (as previously seen in the analysis involving the 166 problem instances), which was able to find the optimal solution for 75% of the instances solved, yielding comparatively small $Opt\Delta_{ave}$ and $Opt\Delta_{max}$: 3.7 and 16.7%, respectively. On average, multi-pass methods were able to find 56.3% of the tested problems, most of which implied a relatively high $Opt\Delta_{max}$. Multi-pass methods generated solutions with a maximum of two workstations deviation from the optimal solution; however, most methods implied a deviation of only one workstation. The worst performance was recorded for method *RS_EW* (*POS*=34.1%).

Table 5.18: Solution quality evaluation for non-weighted multi-pass methods (*NI*=44, *CT*=0.1)

Method	*POS*	%S with difference from optimum		Deviation from optimum	
		1 station	2 stations	$Opt\Delta_{max}$	$Opt\Delta_{ave}$
RS_RPW	59.1	1.0	0	50.0	9.9
RS_T	63.6	36.4	0	33.3	6.9
RS_EW	34.1	63.6	2.3	50.0	12.4
RS_LW	59.1	40.9	0	50.0	9.9
RS_N	47.7	50.0	2.3	50.0	11.4
RS_Sk	54.5	45.5	0	50.0	11.0
RS_TLW	63.6	36.4	0	33.3	6.9
RS_IS	43.2	56.8	0	50.0	12.4
RS_TS	52.3	47.7	0	33.3	9.7
RS_TTS	75.0	25.0	0	16.7	3.7
RS_STS	50.0	47.7	2.3	33.3	9.6
RS_TSSk	47.7	50.0	2.3	50.0	11.4
RS_LWTS	52.3	47.7	0	33.3	9.7
NP_RT	61.4	38.6	0	33.3	6.5
TT_RT	63.6	36.4	0	33.3	6.5
NT_RT	61.4	38.6	0	33.3	6.8
RS_RT	68.2	31.8	0	25.0	5.4
$Opt\Delta_{min} = 0$ in all cases					

Weighted multi-pass methods

Table 5.19 presents the analysis of the results for methods using probability distributions based on weighted values of the priority rules (methods class *d*). As can be observed in Table 5.19, the best results were obtained with methods *W-TT_LWTS*, *W-TT_RPW* and *W-NT_LWTS*, which were able to find the optimal solution for 84.1, 81.8 and 81.8% of the problems solved, respectively. Furthermore, these methods yielded an average and maximum deviation from the optimum of up to 2.6 and 17%, respectively. On average, all methods were able to find the optimal solution in 70.7% of the problems solved. It is unexpected that methods [*SubgraphRule_LWTS*] be that effective since when considering the 166 test instances they performed the worst.

Table 5.19: Solution quality evaluation for weighted multi-pass methods (*NI*=44, *CT*=0.1)

Method	*POS*	%S 1 station diff. from opt.	Deviation from optimum	
			$Opt\Delta_{max}$	$Opt\Delta_{ave}$
W-NP_RPW	77.3	22.7	17	3.3
W-NP_T	68.2	31.8	25	5.3
W-NP_EW	61.4	38.6	25	6.1
W-NP_LW	65.9	34.1	25	5.7
W-NP_N	52.3	47.7	33	8.9
W-NP_Sk	68.2	31.8	25	5.4
W-NP_TLW	72.7	27.3	25	4.2
W-NP_IS	65.9	34.1	33	5.5
W-NP_TS	79.5	20.5	20	3.1
W-NP_TTS	75.0	25.0	17	3.7
W-NP_STS	65.9	34.1	25	5.3
W-NP_TSSk	79.5	20.5	20	3.2
W-NP_LWTS	77.3	22.7	17	3.3
W-TT_RPW	81.8	18.2	17	2.6
W-TT_˙T	72.7	27.3	25	4.6
W-TT_EW	63.6	36.4	25	5.8
W-TT_LW	68.2	31.8	25	5.3
W-TT_N	59.1	40.9	33	7.6
W-TT_Sk	63.6	36.4	25	6.1
W-TT_TLW	75.0	25.0	25	3.9
W-TT_IS	68.2	31.8	25	5.3
W-TT_TS	70.5	29.5	25	4.8
W-TT_TTS	75.0	25.0	17	3.7
W-TT_STS	72.7	27.3	25	4.2
W-TT_TSSk	72.7	27.3	20	4.2
W-TT_LWTS	84.1	15.9	17	2.3
W-NT_RPW	79.5	20.5	17	3.0
W-NT_T	70.5	29.5	25	4.9
W-NT_EW	61.4	38.6	25	6.4
W-NT_LW	65.9	34.1	25	5.8
W-NT_N	63.6	36.4	33	6.4
W-NT_Sk	65.9	34.1	25	5.8
W-NT_TLW	75.0	25.0	25	3.9
W-NT_IS	65.9	34.1	25	5.9
W-NT_TS	72.7	27.3	25	4.4
W-NT_TTS	75.0	25.0	17	3.7
W-NT_STS	70.5	29.5	25	4.8
W-NT_TSSk	75.0	25.0	20	3.9
W-NT_LWTS	81.8	18.2	17	2.6
$Opt\Delta_{min} = 0$ in all cases				

The worth of all results was obtained with method *W-NP_N* which was able to find the optimal solution for 52.3% of the problems solved. A noteworthy result is that all methods generated solutions with at maximum one workstation deviation from the optimum. These results indicate that good solutions can be expected by applying this type of heuristic method.

Nevertheless, if a single computing time is used, all heuristic procedures can be evenly compared, since they can be evaluated based on the same reference value: the optimal solution. In this way, the best method can be identified.

Effects of longer computing times on solution quality

To study the effects of longer computing times on solution quality, all data sets were solved by using multi-pass method *RS_TTS* (the non-weighted multi-pass method with highest percentage of best solutions: *PBS*=87.3%) and computing times of 0.1, 1, 5, 30 60, 180, 300 and 600 seconds. The results obtained are presented in Table 5.20.

Table 5.20: Results for *RS_TTS* considering different *CT* values (*NI*=166)

Measure	Computing time in seconds							
	0.1	**1**	**5**	**30**	**60**	**180**	**300**	**600**
NBS	144	145	145	146	148	150	151	152
PBS	87.3	87.3	87.9	88.5	89.7	90.9	91.5	92.1

As shown in Table 5.20, for 0.1 seconds, the heuristic was able to find the best solutions for 87.3% of the cases solved; the same result was obtained for 1 second. For 5 seconds (a 5000% bigger computation time) the percentage increased to 87.9%, which represents only 0.6% of improvement. Furthermore, for 30, 60, 180 and 600 seconds an improvement of 1.2, 2.4, 3.6, 4.2, and 4.8%, respectively, was achieved over the solution provided with 0.1 second. It is noteworthy, that the percentage of the improvement is not proportional to the computing time increments; since, for example, only a 4.8% of improvement is achieved over the solution obtained with a 600000% smaller computing time. However, as illustrated in Figure 5.9, the results provided by multi-pass methods can be expected to improve with much longer computing times.

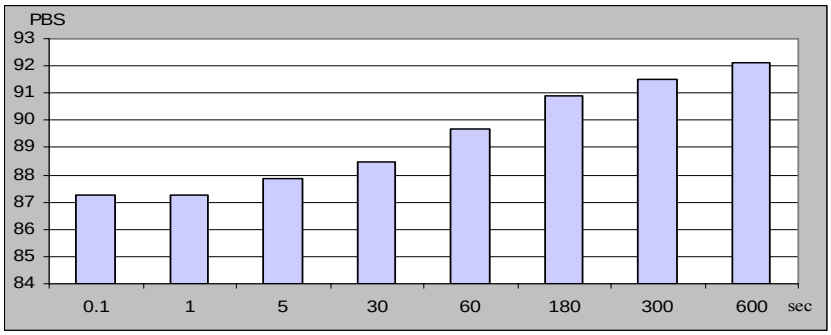

Figure 5.9: Multi-pass methods: percentage of best solutions for different *CT*

Comparison of the performance of simple- and multi-pass methods

Figure 5.10 shows the percentage of best solutions generated by all proposed methods. The comparison is carried out considering again three categories: 1) single-pass, 2) non-weighted multi-pass and 3) weighted multi-pass methods. As can be seen in Figure 5.10, the best result was obtained with a class-d multi-pass method: $W\text{-}TT\_RPW$, which found the best solutions in 84.8% of the cases. In general, weighted multi-pass methods performed better than all other methods, the majority of which provided the best solutions for at minimum 60% of the tested problems. On the other hand, single-pass methods performed the worst, for most of which the PBS only yielded less than 50%.

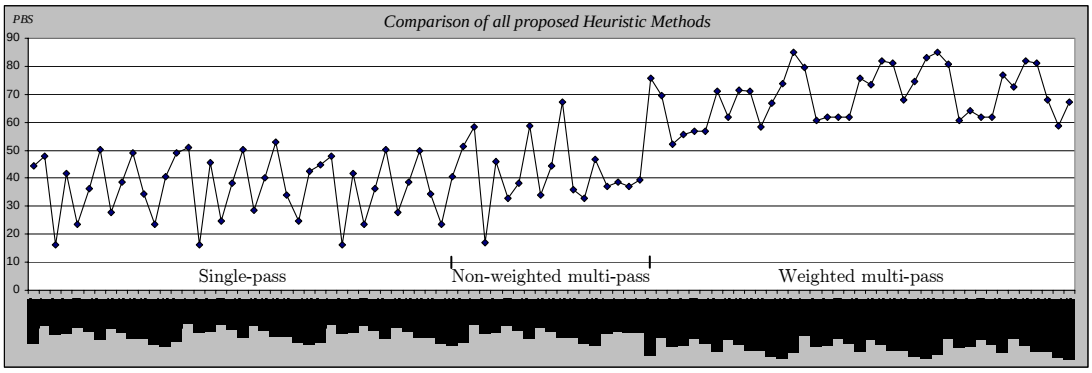

Figure 5.10: Overall performance of single-pass and multi-pass methods

Figure 5.11 shows, per category, the percentage of the best solutions (PBS), the average deviation (Ave_dev), the percentage of solutions equal to the optimal solution (POS) and maximum average deviation from the optimal solution (Dev_opt). Figure 5.11 corroborates the low performance of single-pass methods, which obtained the lowest PBS (52.7%) and POS (68.2%) and the highest deviations (Ave_dev=8.5%, Dev_opt=14.6%). Furthermore, weighted multi-pass methods obtained the highest PBS (84.8%) and the highest POS (84.1%). Furthermore, the solutions provided by weighted multi-pass methods implied the lowest deviations (Ave_dev=2.4%, Dev_opt=8.9%). Therefore, it can be stated that weighted multi-pass methods performed the best.

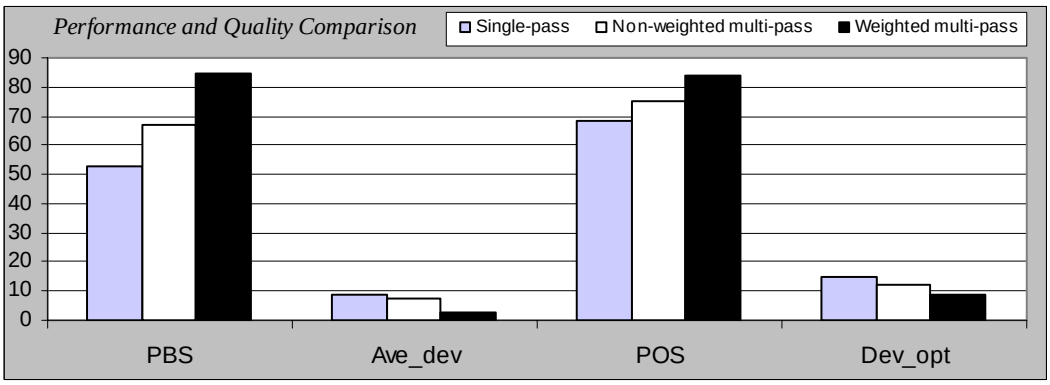

Figure 5.11: Method performance and solution quality comparison

Chapter 6

Conclusions, Contributions and Future Research Proposals

This doctoral thesis addressed a new generalized assembly line balancing problem with practical relevance that has been defined and entitled here ASALBP: the *Alternative Subgraphs Assembly Line Balancing Problem*. The core feature of such a problem is that it considers alternative variants for different parts of an assembly or manufacturing process. Each variant is represented by a precedence subgraph that defines the tasks to be performed, their precedence relations and their corresponding processing times. Furthermore, mutually exclusive assembly processes, involving different sets of assembly tasks, are also considered. To solve the ASALBP efficiently, two problems have to be solved simultaneously: (1) the decision problem to select the assembly alternative and (2) the balancing problem that assigns the tasks to the workstations. This problem implies a high level of difficulty since for the simple case it is verified the *NP-hard* condition.

6.1 Main Results

Many real-life assembly line balancing problems involve assembly variants. Therefore, there is an increasing interest of addressing problems that consider assembly alternatives. The comprehensive analysis of the state-of-the-art on assembly line balancing problems showed that most studies deal with the simple case (SALBP) and problems involving assembly variants are not often

considered in the literature. When processing alternatives exist they are mainly related to the problem of equipment selection. On the other hand, strategies have been proposed aiming at integrating the sequence planning into the balancing process. However, due to its complexity, a two-stage approach is usually considered to select, according to a given criterion, one of the available alternatives; and then the line is balanced considering that choice. In this work it was illustrated how, by following such an approach, a problem involving assembly alternatives can be sub-optimized since the effect of the unselected variants remains unexplored. Furthermore, the best solution can be discarded due to it does not match the decision criterion considered.

The literature review also revealed that the *Alternative Subgraphs Assembly Line Balancing Problem*, which considers the variants that different parts of an assembly process may admit, has not been addressed before. Only the works of Pinto *et al.* (1983) and, much more recently, Scholl *et al.* (2007) considered a similar problem involving processing alternatives. In the former case, the alternatives are defined by the assignment of a given equipment type to the workstations; furthermore, they mentioned the possibility of having variable precedence requirements but they did not considered such a case. The latter work considered a special case of the ASALBP in which the alternatives are represented by time increments that are added to the processing times, which are dependent on the task processing sequence. However, none of the cases treated the problem in which alternative sets of precedence constraints are allowed but instead they considered them fixed. Thus, a new GALB Problem has been defined.

Due to the impossibility to depict all available assembly variants in a standard precedence graph, in this work the *S-Graph* has been proposed as a diagramming tool to represent in a unique graph all available alternatives.

In order to formalize the new ASALBP, two mathematical programming models were proposed in this work. In a preliminary model (M1) assembly alternatives were regarded as global routes, which were represented by a complete precedence graph and determined by the combination of the available subgraphs. By analysing this model, it was considered that its dimension could be reduced by considering each individual subgraph as a partial route, involving only a reduce set of the assembly tasks. Therefore, an enhanced model (M2) was developed considering partial routes.

Different test problems were generated considering small-, medium- and large-scale benchmark SALBP. ASALBP test instances, which can themselves be considered benchmarks, were generated by adapting the original problems in such a way that assembly alternatives were involved. The computational experiment carried out revealed that the number of variables and constraints were significantly reduced with M2, which resulted in a considerable reduction on the computation time comparing with M1. Furthermore, in all cases M2 outperformed M1, yielding in 33% of the cases a 100% of improvement. The analysis of the results also indicates that mathematical programming models can be applied to optimally solve only small- and medium-scale ASALBP instances; i.e., from 10 to around 30 assembly tasks and from 5 to 11 subgraphs.

This new combinatorial optimization problem thus required of the design and development of approximate methods to solve industrial-scale problems. Several heuristic methods to solve the ASALBP were proposed in this thesis. As discussed earlier, constructive methods based on priority rules have been successfully applied to assembly line balancing problems; therefore, this type of methods were considered here. Due to it has been proven that workstation-oriented methods perform better than task-oriented ones, all proposed procedures followed such an approach; therefore, a new workstation is open only once the current workstation is fully loaded.

Several criteria were considered to select the assembly subgraphs. In order to be able to evaluate the impact of a priori selection of a given assembly alternative on the solution of the problem, three single-priority rules were considered. Random search mechanisms were also used to allow a more flexible exploration of the solution space. On the other hand, decision criteria for selecting the tasks were based on an adaptation to the ASALBP of 13 of the most well-known priority rules used to solve SALBP, and on random choice. Furthermore, both subgraphs and tasks were selected by using probability distributions based on weighted, instead of nominal, values of the priority rules. The combination of all decision criteria gave rise to a total of 95 heuristic procedures, divided into single-pass and multi-pass methods (the latter further divided into non-weighted and weighted multi-pass methods), being able to provide a single solution and multiples solutions, respectively.

The performance of all methods was evaluated via a computational experiment based on the number of best solutions generated, involving 48472 experiments. Furthermore, the optimal solutions found with the mathematical models were used to evaluate the quality of the provided solutions; i.e., the deviation from the optimal solution.

The analysis of the results showed that single-pass and multi-pass methods using *EW* (Minimum Earliest Workstation), *N* (Minimum Task Number) and *IS* (Maximum Number of Immediate Successors) as decision rules for tasks proved to be inefficient at solving ASALB problems (i.e. maximum *PBS* is less than 50.5%); being *TTS* (Maximum Task Time plus Total number of Successors) one of the most efficient priority rules. Furthermore, the results obtained revealed that multi-pass methods outperformed single-pass ones; particularly, weighted multi-pass methods were able to find best solutions in 84.8% of the cases. When subgraphs are selected randomly, the solution space is explored more exhaustively; therefore, there is a better chance of selecting the subgraphs that provide the best solution.

Additionally, the comparison of the obtained solutions with the found optimal solution corroborated the results obtained when considering the percentage of best solution (*PBS*) as the evaluation measure. Multi-pass methods using probability distributions based on weighted values of the priority rules performed the best having the highest *POS* (i.e. 84.1%) and the lowest deviations from both the best solution and the optimal solution (2.4 and 8.9%, respectively). Therefore, the application of weighted multi-pass methods can be recommended.

In order to improve the solution of the proposed heuristic methods, two local optimization procedures were also proposed here, which are based on an adaptation of two classical neighbourhood search strategies: LOP-1 that considers the exchange of the positions in the solution sequence of a pair of tasks, i and k; and LOP-2 that is based on the movement of one task i to the position of another task k, which also implies the movement of task k and all tasks in between tasks i and k. A computational experiment designed to evaluate the performance of both procedures revealed that improved solutions could be obtained in which up to two workstations less were required, which indicates that a financial benefit can be obtained by applying the proposed local optimization methods. On the other hand, the results also showed that in all cases LOP-2 outperformed LOP-1, yielding improved solutions in one and two workstations in 35.8 and 26.1%, respectively. Thus, all proposed methods that used LOP-2 could be applied to solve an ASALBP to select the best overall solution.

On the other hand, it was shown that the results provided by multi-pass methods can be expected to improve with much longer computing times. Therefore, if there are no time constraints, multi-pass heuristics could be applied with much greater available computing time, considering for example 3600 or 18000 seconds, which are realistic time-windows considering industrial-size problems.

6.2 Proposals for Future Research Work

The *Alternative Subgraphs Assembly Line Balancing Problem* introduced and defined in this thesis is a new GALBP with practical relevance. Therefore, future research work will mainly involve exploring other methods to solve efficiently this new problem. Furthermore, and aiming at closing the gap between research works and real applications, other relevant characteristics can be included to the ASALB Problem.

Exact approaches

Branch and bound procedures has been successfully applied to solve hard optimization problems. Therefore, this strategy can be considered to optimally solve the ASALBP. Another optimization approach that can be explored refers to disjunctive programming models, which have been used to solve problems involving alternative constraints.

Metaheuristic procedures

The growing interest on using Evolutionary Algorithms (e.g. Genetic Algorithms) to solve optimization problems in industry makes the use of such procedures an attractive approach, which, in addition, has been successfully applied to complex assembly line balancing problems.

Additional characteristics

In order to increase the practicality of the problem, its definition can be extended by including new features such as, for example, stochastic processing times.

6.3 Contributions

The following written contributions are part of the research work undertaken in this doctoral thesis.

1. Capacho, L. and Pastor, M. (2004). Generación de secuencias de montaje y equilibrado de líneas, Technical Report IOC-DT-P-2004-04, Technical University of Catalonia, Barcelona, Spain.
2. Capacho, L. and Pastor, R. (2005). ASALBP: the Alternative Subgraphs Assembly Line Balancing Problem. Technical Report: IOC-DT-P-2005-5. UPC. Barcelona, Spain. International Journal of Production Research (to appear).
3. Capacho, L. and Pastor, R. (2005). Modelo de Programación Matemática del Problema de Equilibrado de Líneas con Subgrafos de Montaje Alternativos. IX Congreso de Ingeniería de Organización Gijón, 8 y 9 de Septiembre de 2005.
4. Capacho, L., Guschinskaya, O., Dolgui, A., Pastor, R. (2006). Approximation Methods to Solve the Alternative Subgraphs Assembly

Line Balancing Problem, Research Report, G2I-EMSE 2006-500-003, Ecole des Mines, SE, France, April 2006.

5. Capacho, L., Guschinskaya, O., Dolgui, A., Pastor, R. (2006). A Comprehensive Comparative Analysis of Heuristic Methods for the Alternative Subgraphs Assembly Line Balancing Problem, Research Report, G2I-EMSE 2006-500-005, Ecole des Mines de Saint Etienne, France, 2006.
6. Capacho, L. and Pastor, R. (2006). The ASALB Problem with Processing Alternatives Involving Different Tasks: Definition, Formalization and Resolution, in The 2006 International Conference on Computational Science and its Applications, ICCSA 2006, Lecture Notes in Computer Science, Eds. M. Gavrilova *et al.*: Springer-Verlag, Berlin, May 2006, 3982, pp. 554–563.
7. Capacho, L. and Pastor, R. (2006). Equilibrado de Líneas con Alternativas de Montaje. SEIO 2006: Contribuciones a la Estadística y a la Investigación Operativa, Tenerife, 15-19 de Mayo de 2006.
8. Capacho, L. and Pastor, R. (2006). Formalización matemática del problema de equilibrado de líneas con procesos de montaje mutuamente excluyentes. X Congreso de Ingeniería de Organización Valencia, 7 y 8 de Septiembre de 2006.
9. Capacho, L., Guschinskaya, O., Dolgui, A., Pastor, R. (2006). An Evaluation Study of Approximate Methods for a Line Balancing Problem with Assembly Alternatives. 8th International Conference on The Modern Information Technology in the Innovation Processes of the Industrial Enterprises, 11-12 September, 27-30, Budapest, Hungary, 2006.
10. Capacho, L., Pastor, R., Guschinskaya, O. and Dolgui, A. (2006). Heuristic Methods to Solve the Alternative Subgraphs Assembly Line Balancing Problem. IEEE Conference on automation Science and Engineering CASE 2006, Shanghai-China, 8-11 October 2006.
11. Capacho, L., Guschinskaya, O., Dolgui, A., Pastor, R. (2006). A Comparative Analysis of Heuristic Methods for the Alternative Subgraphs Assembly Line Balancing, XIII Congreso Latino-Iberoamericano de Investigación Operativa CLAIO2006, Montevideo-Uruguay, 27-30 November, 2006.
12. Capacho, L., Pastor, R., Dolgui, A., Guschinskaya, O. (2007). An Evaluation of Constructive Heuristic Methods to Solve the Alternative Subgraphs Assembly Line Balancing Problem. Journal of Heuristics (to appear).
13. Capacho, L. and Pastor, R. (2007). A Metaheuristic Approach to Solve the ASALBP: an Assembly Line Balancing Problem Involving Assembly Alternatives (in preparation).

References

Agpak, K. and Gokcen, H. (2005). Assembly line balancing: Two resource constrained cases. International Journal of Production Economics, 96, 129–140.

Ahn, J. and Kusiak, A. (1990). Scheduling with Alternatives Process Plans. Modern Productions concepts. Theory and Applications in Proceedings of an Int. Conference, Fernunniversitat, Hagen, August 20–24, Springer–Verlag.

Ajenblit, D. and Wainwright, R. (1998). Applying genetic algorithms to the U–shaped assembly balancing problem, Proceedings of the 1998 IEEE Int. Conference on Evolutionary Computation, Anchorage, Alaska, 96–101.

Amen, M. (2000). Heuristic methods for cost–oriented assembly line balancing: A survey. International Journal of Production Economics, 68, 1–14.

Amen, M. (2001). Heuristic methods for cost–oriented assembly line balancing: A comparison on solution quality and computing time. International Journal of Production Economics, 69, 255–264.

Andres, C., Miralles, C. and Pastor, R. (2006). Balancing and sequencing tasks in assembly lines with sequence–dependent setup times. European Journal of Operational Research (to appear).

Arcus, A. (1966). COMSOAL: A computer method of sequencing operations for assembly lines. International Journal of Production Research, 4, 259–277.

Armentano, A. and Bassi, O. (2006). Graph with Memory-based Mechanisms for Minimizing Total Tardiness in Single Machine Scheduling with Setup Times. Journal of Heuristics, 12, 427–446.

Avram, F., Bertsimas, D. and Ricard, M. (1995). Fluid models of sequencing problems in open queuing networks: an optimal control approach. In Stochastic Networks, Mathematics and its Applications, 71, 199-234.

Bard, J. (1989). Assembly line balancing with parallel workstations and dead time. International Journal of Production Research, 27, 1005–1018.

Bard, J., Dar–El, E. and Shtub, A. (1992). An analytic framework for sequencing mixed model assembly lines. International Journal of Production Research, 30, 35–48.

Bartholdi, J. (1993). Balancing two–sided assembly lines: A case study. International Journal of Production Research, 31, 2447–2461.

Bautista, J. and Pereira, J. (2002). Ant algorithms for assembly line balancing. In: Dorigo, M., Di Caro, G., Sampels, M. (Eds.). Lecture Notes in Computer Science, Springer, Berlin, 2463, 65–75.

Bautista, J. and Pereira, J. (2003). Algoritmos de hormigas para un problema de equilibrado de líneas. V Congreso de Ingeniería de Organización, Valladolid–Burgos. 2003.

Baybars, I. (1986). A survey of exact algorithms for the simple assembly line balancing problem. Management Science, 32, 909–932.

Baykasoglu, A., Dereli, T., Erol, R. and Sabuncu, I. (2003). An ant colony based optimization algorithm for solving assembly line balancing problems. International XII Turkish Symposium on Artificial Intelligence and Neural Networks.TAINN 2003.

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.

Berger, I., Bourjolly, J. and Laporte, G. (1992). Branch–and–bound algorithms for the multi–product assembly line balancing problem. European Journal of Operations Research, 58, 215–222.

Bockmayr, A. and Pisaruk, N. (2001). Solving assembly line balancing problems by combining IP and CP. Proceedings of the 6th Annual Workshop of the ERCIM Working Group on Constraints, Prague, Czech Republic.

Boctor, F. (1995). A Multiple–Rule Heuristic for Assembly Line Balancing. Journal of the Operational Research Society, 46, 62–69.

Bowman, E. (1960). Assembly Line Balancing by Linear Programming. Operations Research, 8, 385–389.

Boysen, N., Fliedner, M. and Scholl, A. (2007a). Assembly line balancing: Which model to use when?. International Journal of Production Economics (to appear, doi: 10.1016/j.ijpe.2007.02.026).

Boysen, N., Fliedner, M. and Scholl, A. (2007b). A classification of assembly line balancing problems. European Journal of Operational Research, 183, 674–693.

Bukchin, J. (1998). A comparative study of performance measures for throughput of a mixed model assembly line in a JIT environment. International Journal of Production Research, 36, 2669–2685.

Bukchin, J. and Masin, M. (2004). Multi–objective design of team oriented assembly. European Journal of Operational Research, 156, 326–352.

Bukchin, J. and Rubinovitz, J. (2003). A weighted approach for assembly line design with station paralleling and equipment selection. IIE Transactions, 35, 73–85.

Bukchin, J. and Tzur, M. (2000). Design of flexible assembly line minimize equipment cost. IIE Transactions, 32, 585–598.

Bukchin, J., Dar–El, E. and Rubinovitz, J. (2002). Mixed–model assembly line design in a make–to–order environment. Computers & Industrial Engineering, 41, 405–421.

Bukchin, Y. and Rabinowitch, I. (2005). A branch–and–bound based solution approach for the mixed–model assembly line–balancing problem for minimizing stations and task duplication costs. European Journal of Operational Research (to appear).

Bukchin, Y., Meller, R. and Liu, Q. (2006). Assembly system facility design. IEE Transactions, 38, 53–65.

Buxey, G. (1974). Assembly line balancing with multiple Stations. Management Science, 20, 1010–1021.

Capacho, L. and Pastor, R. (2005). ASALBP: the Alternative Subgraphs Assembly Line Balancing Problem. Technical Report: IOC–DT–P–2005–5. UPC. Barcelona, Spain. To appear in: International Journal of Production Research.

Capacho, L. and Pastor, R. (2006). The ASALB Problem with Processing Alternatives Involving Different Tasks: Definition, Formalization and Resolution. Lecture Notes in Computer Science, Springer, 3982, 554–563.

Chiang, W–C. (1998). The application of a tabu search metaheuristic to the assembly line balancing problem. Annals of Operations Research, 77, 209–227.

Corominas, A., Pastor, R. and Plans, J. (2006). Balancing assembly line with skilled and unskilled workers. OMEGA (In Press, Corrected Proof).

Dai, J. G. and Weiss, G. (2002). A fluid heuristic for minimizing makespan in job-shops. Operations Research, 50, 692–707.

Das, S. and Nagendra, P. (1997). Selection of routes in a flexible manufacturing facility. International Journal of Production Economics, 48, 237–247.

DePuy, G. and Whitehouse, G. (2000). Applying the COMSOAL computer heuristic to the constrained resource allocation problem. Computers and Industrial Engineering, 38, 3, 413–422.

Dolgui, A. (2006). Balancing Assembly and Transfer Lines. European Journal of Operational Research, 168, 663–665.

Dolgui, A., Finel, B., Guschinsky, N. and Levin, G. (2005). A heuristic approach for transfer line balancing. Journal of Intelligence Manufacturing, 16, 159–171.

Dorigo, M., Maniezzo, V. and Colorni, A. (1996). The Ant System: Optimization by a Colony of Cooperating Agents. IEEE Transactions on Systems, Man and Cybernetics - Part B, 26, 1, 29–41.

Erel, E. and Gokcen H. (1999). Theory and Methodology: Shortest–route formulation of mixed–model assembly line balancing problem. European Journal of Operational Research, 116, 194–204.

Erel, E. and Sarin, S. (1998). A survey of the assembly line balancing procedures. Production Planning & Control, 9, 414–434.

Feo, T., Resende, M. and Smith, S. (1994). A greedy randomized adaptive search procedure for maximum independent set. Operations Research, 42, 860–878.

Fernades, E.R and Ribeiro, C.C (2005). A Multistart Constructive Heuristic for Sequencing by Hybridization Using Adaptive Memory. Electronic Notes in Discrete Mathematics, 19, 41–47.

Festa, P. and Resende, M. (2004). An Annotated Bibliography of GRASP. Technical Report, TD-5WYSEW, AT&T Labs, February 2004.

Feyzbakhsh, S.A. and Matsui, M. (1999). Adam–Eve–like genetic algorithm: A methodology for optimal design of a simple flexible assembly system. Computers and Industrial Engineering, 36, 233–258.

Flake, G.W. (1999). The computacional beuty of Nature. Computer Explorations of Fractals, Chaos, Complex Systems and Adaptation. The MIT Press.

Fleszar, K. and Hindi, K. (2003). An enumerative heuristic and reduction methods for the assembly line balancing problem. European Journal of Operational Research, 145, 606–620.

Gaalman, G., Slomp, J. and Suresh, N. (1999). Towards an Integration of process planning and Control for Flexible Manufacturing Systems. The International Journal of Flexible Manufacturing Systems, 11, 5–17.

Gamberini, R., Grassi, A. and Rimini, B. (2005). A new multi–objective heuristic algorithm for solving the stochastic assembly line re–balancing problem. International Journal of Production Economics, 102, 226–243.

Gen, M., Tsujimura, Y. and Li, Y. (1996). Fuzzy assembly line balancing using genetic algorithms. Computers and Industrial Engineering, 31, 3/4, 631–634.

Ghosh, S. and Gagnon, R. (1989). A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. International Journal of Production Research, 27, 4, 637–670.

Glover, F. (1990). Tabu search: a tutorial. Interfaces, 20, 74–94.

Glover, F. (1996). Tabu search and adaptive memory programming. Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington (eds.). Interfaces in Computer Science and Operations Research, pages 1-75. Kluwer.

Glover, F. and Laguna, M. (1993). Tabu Search. In Modern Heuristic Techniques for Combinatorial Problems, Reeves (ed.), Blackwell, Oxford. 70–150.

Glover, F. and Laguna, M. (1997). Tabu search. Kluwer Academia Publishers, Boston.

Goldberg, D.E. (1989). Genetic Algorithms in Search, Optimization Learning, Addison Wesley Publishing Company.

Gottlieb, J., Puchta, M. and Solnon, C. (2003). A study of greedy, local search, and ant colony optimization approaches for car sequencing problems. Lecture Notes in Computer Science, Springer, 2611, 246–257.

Gungor, A. and Gupta S. (1997). An Evaluation Methodology For Disassembly Processes. Computers in industrial Engineering, 33, 1–4.

Hackman, S., Magazine, M. and Wee, T. (1989). Fast, Effective Algorithms for Simple Assembly Line Balancing Problems. Operations Research, 37, 916–924.

Hao, N. (2005). Sequencing and Balancing Problem of Mixed–Model–Assembly–Line with Window Cycle Time. Doctoral Thesis. UPC. Barcelona. Spain.

Held, M., Karp, R. and Shareshian, R. (1963). Assembly line balancing–dynamic programming with precedence constraints. Operations Research, 11, 3, 442-459.

Helgeson, W. and Birnie, D. (1961). Assembly line balancing using the ranked positional weight technique. Journal of Industrial Engineering, 12, 394–398.

Hoffmann, T.R. (1992). EUREKA: A hybrid system for assembly line balancing, Management Science, 38, 39–47.

Hong, D. and Cho, H. (1999). A genetic–algorithm–based approach to the generation of robotic assembly sequences. Control Eng. Practice, 7, 151–159.

Jackson, J. (1956). A computing Procedure for a Line Balancing Problem. Management Science, 2, 261–271.

Ji, P., Sze, M.T. and Lee, W.B. (2001). A genetic algorithm of determining cycle time for printed circuit board assembly lines. European Journal of Operational Research, 128, 175–184.

Johnson, R.V. (1988). Optimally balancing large assembly lines with "FABLE". Management Science, 34, 240–253.

Kao, E. and Queyranne, M. (1982). On dynamic programming methods for assembly line balancing. Operations Research, 30, 375–390.

Karabati, S. and Sayin, S. (2003). Assembly line balancing in a mixed–model sequencing environment with synchronous transfers. European Journal of Operational Research, 149, 417–429.

Kilincci, O. and Bayhan, G. (2006). A Petri net approach for simple assembly line balancing problems. The International Journal of Advanced Manufacturing Technology, 30, 1165-1173.

Kilincci, O. and Bayhan, G. (2007). A *P*-invariant-based algorithm for simple assembly line balancing problem of type-1. The International Journal of Advanced Manufacturing Technology (to appear, doi: 10.1007/s2007.02.026 00170-007-0975-2).

Kim, Y.K., Kim Y.J. and Kim, Y. (1996). Genetic algorithms for assembly line balancing with various objectives. Computers & Industrial Engineering, 30, 3, 397–409.

Kim, Y.K., Kim, Y. and Kim, Y.J. (2000). Two–sided assembly line balancing: a genetic algorithm approach. Production Planning & Control, 11, 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 Operations Research, 91, 367–385.

Kubiak, W. and Suresh, S. (1991). A note on level schedules for mixed model assembly lines in Just in Time Production Systems. Management Science, 37, 121–122.

Lambert, A. (2006). Generation of assembly graphs by systematic analysis of assembly structures. European Journal of Operational Research, 168, 932–951.

Lapierre, S. and Ruiz, A. (2004). Balancing assembly lines: an industrial case study. Journal of the Operational Research Society, 55, 559–597.

Lawler, E.L. (1979). Efficient implementation of dynamic programming algorithms for sequencing problems. Report BW 106/79, Stichting Mathematisch Centrum, Amsterdam.

Lee, Q. (2000). How to balance manufacturing work cell. Institute of Industrial Engineers – IE Solutions Conference, May 21–23, Cleveland, Ohio.

Little, J., Murty, K., Sweeney, D. and Karel, C. (1963). An algorithm for the traveling salesman problem. Operations Research, 11, 972–989.

Malakooti, B. and Kumar, A. (1996). A knowledge–based system for solving multi–objective assembly line balancing problem. International Journal of Production Research, 34, 9, 2533–2552.

Martí, R. and Moreno, M. (2003). Multistart methods. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial, 19, 49–60.

Martin, G.E. (1994). Optimal design of production lines. International Journal of Production Research, 32, 989–1000.

Martinez, U. and Duff, W. (2004). Heuristic approaches to solve the U–shaped line balancing problem augmented by genetic algorithms. Proc. of the 2004 Systems and Information Eng. Design Symposium. Mattew H. Jones, Stephen D. Patek, and Barbara E. Tawney. Eds.

McMullen, P. and Frazier, G. (1998a). Using simulated annealing to solve a multi–objective assembly line balancing problem with parallel workstations. International Journal of Production Research, 36, 2717–2741.

McMullen, P. and Frazier, G. (1998b). Using Simulation and Data Envelopment Analysis to compare assembly line balancing solutions. Journal of Productivity Analysis, 11, 149–168.

McMullen, P. and Tarasewich, P. (2006). Multi-objective assembly line balancing via a modified ant colony optimization technique. International Journal of Production Research, 44, 27–42.

Mendes, A., Ramos, A., Simaria, A. and Vilarino, P. (2005). Combining heuristic procedures and simulation models for balancing a PC camera assembly line. Computers and Industrial Engineering, 49, 3, 413-431.

Merengo, C., Nava, F. and Pozetti, A. (1999). Balancing and sequencing manual mixed–model assembly lines. International Journal of Production Research, 37, 2835–2860.

Miltenburg, J. (1998). Balancing U–lines in a multiple U–line facility. European Journal of Operational Research, 109, 1–23.

Miltenburg, J. (2002). Balancing and scheduling mixed–model U–shaped production lines. International Journal of Flexible Manufacturing Systems, 14, 119–151.

Miltenburg, J. and Wijngaard, J. (1994). The U–line line balancing problem. Management Science, 40, 1378–1388.

Miralles, C. (2004). Modelos, métodos y algoritmos de resolución para el problema de asignación de puestos y equilibrado en líneas con tiempos dependientes del operario. Aplicación en centros especiales de empleo para personas con discapacidades. Tesis Doctoral; UPV.

Miralles, C., Capó, J., García, J.P. and Andrés C. (2003). Equilibrado de Líneas de Montaje considerando variables los tiempos de operación y las habilidades de los operarios. 27 Congreso Nacional de Estadística e Investigación Operativa. Lleida, 8–11 de Abril de 2003.

Moberly, L.E. and Wyman, F.P. (1973). An application of simulation to the comparison of assembly line configurations. Decision Sciences, 4, 505-516.

Moon, C., Lee, M., Seo, Y. and Lee, Y.H. (2002). Integrated machine tool selections and operation sequencing capacity and precedence constraints using genetic algorithm. Computers & Industrial Engineering, 43, 605–621.

Nicosia, G., Pacciarelli, D. and Pacifici, A. (2002). Optimally balancing assembly lines with different Workstations. Discrete Applied Mathematics, 118, 99–113.

Osman, I.H. and Laporte, G. (1996). Metaheuristics: a bibliography. Annals of Operations Research, 63, 513–623.

Park, K., Park, S. and Kim, W. (1997). A heuristic for an assembly line balancing problem with incompatibility, range, and partial precedence constraints. Computers & Industrial Engineering, 32, 2, 321–332.

Pastor R. (1999). Metalgoritmo de optimizacion combinatoria mediante la exploracion de grafos, Tesis doctoral, Universitat Politecnica de Catalunya.

Pastor, R., Andres, C., Duran, A. and Perez, M. (2002). Tabu search algoritms for an industrial multi–product, multi–objective assembly line balancing problem, with reduction of task dispertion. Journal of Operations Research Society, 53, 1317–1323.

Patterson, J. and Albracht, J. (1975). Assembly Line Balancing: 0–1 Programming with Fibonacci Search. Operations Research, 23, 166–174.

Peeters, A. (2006). Linear programming based lower bound for the simple assembly line balancing problem. European Journal of Operations Research, 168, 716–731.

Phonganant, S., Yang, Y.N., Leep, H.R. and Parsaei, H.R. (2001). Expert System for Mixed–Model Assembly Line Balancing. Dallas, TX: 10th Annual Industrial Engineering Research Conference.

Pierreval, H., Caux, C., Paris, J. and Viguier, F. (2003). Evolutionary approaches to the design and organization of manufacturing systems. Computers & Industrial Engineering, 44, 339–364.

Pinnoi, A. and Wilhelm, W.E. (1998). Assembly system design: A branch and cut approach. Management Science, 44, 103–118.

Pinto, P., Dannenbring, D. and Khumawala, B. (1975). A branch and bound algorithm for assembly line balancing with paralleling. International Journal of Production Research, 13, 183–196.

Pinto, P., Dannenbring, D. and Khumawala, B. (1981). Branch and bound and heuristic procedures for assembly line balancing with paralleling of stations. International Journal of Production Research, 19, 565–576.

Pinto, P., Dannenbring, D. and Khumawala, B. (1983). Assembly line balancing with processing alternatives: an application. Management Science, 29, 817–830.

Plans, J. (1999). Classificació, modelització i resolució dels problemes de disseny i assignació de tasques en línies de producció.Tesis Doctoral; UPC.

Ponnambalam, S.G., Aravindan, P. and Subba Rao, M. (2003). Genetic algorithms for sequencing problems in mixed model assembly lines. Computers & Industrial Engineering, 45, 4, 669–690.

Reeves, C. (1993). Modern Heuristics Techniques for Combinatorial Problems. McGraw Hill.

Reeves, C. (1997). Genetic algorithms for the operations researcher. INFORMS Journal of Computers, 9, 231–250.

Rekiek, B. (2001). Assembly Line Design, multiple objective grouping genetic algorithm and the balancing of mixed–model hybrid assembly line. Doctoral Thesis. Universite Libre de Bruxelles.

Rekiek, B., Dolgui, A., Delchambre, A. and Bratcu, A. (2002). State of art of optimization methods for assembly line design. Annual Reviews in Control, 26, 163–174.

Rubinovitz, J. and Bukchin, J. (1993). RALB – A heuristic algorithm for design and balancing of robotic assembly lines. Annals of the CIRP, 42, 497–500.

Rubinovitz, J. and Levitin, G. (1995). Genetic algorithm for assembly line balancing. International Journal of Production Economics, 41, 343–354.

Sarin, S., Erel, E. and Dar–El, E. (1999). A methodology for solving single–model, stochastic assembly line balancing problem. International Journal of Management Science, 27, 525–535.

Sawik, T. (2002). Monolithic vs. hierarchical balancing and scheduling of a flexible assembly line. European Journal of Operational Research, 109, 1–23.

Scholl, A. (1999). Balancing and sequencing assembly lines, 2nd. edition, Physica–Verlag, 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, 666–693.

Scholl, A. and Klein, R. (1997). SALOME: A bidirectional branch and bound procedure for assembly line balancing. INFORMS Journal on Computing, 9, 319–334.

Scholl, A. and Klein, R. (1999a). Balancing assembly lines effectively –A computational comparison. European Journal of Operational Research, 114, 50–58.

Scholl, A. and Klein, R. (1999b). ULINO: Optimally balancing U–shaped JIT assembly lines. International Journal of Production Research, 37, 721–736.

Scholl, A. and Voss, S. (1996). Simple assembly line balancing–Heuristic approaches. Journal of Heuristics, 2, 217–244.

Scholl, A., Boysen, N. and Fliedner, M. (2007). The sequence-dependent assembly line balancing problem. Operations Research Spectrum (to appear, doi: 10.1007/s00291-006-0070-3).

Schrage, L. and Baker, K.R. (1978). Dynamic programming solution of sequencing problems with precedence constraints. Operations Research, 26, 444–449.

Senin, N., Groppetti, R. and Wallace, D. (2000). Concurrent assembly planning with genetic algorithms. Robotics and Computer Integrated Manufacturing, 16, 65–72.

Silver E. (2002). An overview of heuristic solution methods. Working paper 2002–15, Haskayne School of Business, University of Calgary.

Spina, R., Galantucci, M. and Dassisti, M. (2003). A hybrid approach to the single line scheduling problem with multiple products and sequence–dependent time. Computers & Industrial Engineering, 45, 4, 573–583.

Suer, G. (1998). Designing Parallel assembly lines. Computers & Industrial Engineering, 35, 3–4, 467–470.

Suresh, G. and Sahu, S. (1994). Stochastic assembly line balancing using simulated annealing. International Journal of Production Research, 32, 1801–1810.

Talbot, F.B, Patterson, J.H. and Gehrlein, W.V. (1986). A comparative evaluation of heuristic line balancing techniques. Management Science, 32, 431–453.

Thangavelu, S. and Shetty, C. (1971). Assembly Line by Zero–One Integer Programming. AIIE Transactions, 3, 61–68.

Tsai, D. and Yao, M. (1993). A line–balanced base capacity planning procedure for series–type robotic assembly line. International Journal of Production Research, 31, 1901–1920.

Tseng, H.E. and Tang, C.E. (2006). A sequential consideration for assembly sequence planning and assembly line balancing using the connector concept. International Journal of Production Research, 44, 1, 97–166.

Urban, T. (1998). Note: Optimal balancing of U-shaped assembly lines. Management Science, 44, 738-741.

Voss, S. (1994). Tabu search in manufacturing. In H. Dyckhoff, U. Derigs, M. Salomon and H. C. Tijms (eds.). Operational Research Proceedings, 183–194.

Wee, T. and Magazine M. (1982). Assembly line balancing as generalized Bin Packing. Operations Research Letters, 1, 56–58.

Weiss, G. (1999). Scheduling and Control of Manufacturing Systems --- a Fluid Approach. Proceedings of the 37 Allerton Conference, 577-586.

White, W. (1961). Comments on a Paper by Bowman. Operations Research, 9, 274–276.

Wikipedia, Assembly line (2003). http://www.wikipedia.org/wiki/Assembly_line. Visited: September 2003.