UNIVERSITAT POLITÈCNICA DE VALÈNCIA

The Distributed and Assembly Scheduling Problem

Ph.D Thesis

Presented by:
Sara Hatami
Thesis advisors:
Carlos Andrés Romano
Rubén Ruiz García

Valencia, April 18, 2016

To my Mahla,
Dad, Mom
and Sima

Acknowledgments

Above all, I whole-heartedly thank my mighty God for giving me the vision, power, spirit and endurance to complete this interesting research. I thank Him for the countless bounties He has granted me and for giving me the ability to deal with my challenges during the years of my study. This thesis appears in its current form due to the assistance and guidance of several people. I would therefore like to offer my sincere thanks to all of them.

Carlos Andrés-Romano, my co-supervisor, my cordial thanks for your warm encouragement, thoughtful guidance, valuable advise, sharing your valuable suggestion and experience with me and for your support during the whole period of the study. Without you maybe I never would have come to Spain and start the Ph.D.

I would like to express my deepest gratitude to my dear co-supervisor, **Rubén Ruiz** for accepting me as a Ph.D student. Thank you Rubén, for generously supporting the thesis with your expertise and for broadening my horizon, invaluable advices and suggestions, critical comments, correction of the thesis and especially for your patience and guidance during the writing process. You showed me much kindness and provided all help in completing my research thesis. Without you would be impossible to finish this thesis.

I would like to thank my beloved **father** and **mother** for their constant encouragement, continuing support, their ubiquitous love, care and their constantly standing by my side ensuring success even during the most trying times in my life. Special thanks go to my lovely sister **Sima**, my grandmother and

aunts who are always there for me.

Thank you my **Mahla** I hope you will be happy wherever you are to see me accomplish one of your dreams. Special thanks to my Spanish family, **Paco**, **Fina**, **Victor** and **Santiago** to make me always happy and bring me wonderful moments that made me forget the absence of my family. They only give me love.

My very special thanks to my best friend **Mehdi**, who has listened to my moaning and complaining and supported me every step of the way. Thanks for helping me finish this thesis.

Sahar, **Dori**, **Reyhan**, **Amal** and **Nidhal**, I greatly appreciate your excellent assistance and your spiritual supports.

Thanks also to all the members of SOA, **Javier**, **Gabo**, **Gerardo**, **Eugenio**, **Eva** and **Pedro** for providing a good atmosphere in our workplace and for the useful discussions in lunch time.

I would also like to thank **Alfonso García Ninet**, because he accepted me as a member of Colegio Mayor “La Coma” in my first day in Spain and provided me with financial support.

Last but not least, I would like to thank all of my friends who helped me during the writing of this thesis but I cannot mention their names here one by one. Hopefully this thesis can give quite a contribution to the scheduling research.

Abstract

Nowadays, manufacturing systems meet different new global challenges and the existence of a collaborative manufacturing environment is essential to face with. Distributed manufacturing and assembly systems are two manufacturing systems which allow industries to deal with some of these challenges. This thesis studies a production problem in which both distributed manufacturing and assembly systems are considered. Although distributed manufacturing systems and assembly systems are well-known problems and have been extensively studied in the literature, to the best of our knowledge, considering these two systems together as in this thesis is the first effort in the literature. Due to the importance of scheduling optimization on production performance, some different ways to optimize the scheduling of the considered problem are discussed in this thesis.

The studied scheduling setting consists of two stages: A production and an assembly stage. Various production centers make the first stage. Each of these centers consists of several machines which are dedicated to manufacture jobs. A single assembly machine is considered for the second stage. The produced jobs are assembled on the assembly machine to form final products through a defined assembly program.

In this thesis, two different problems regarding two different production configurations for the production centers of the first stage are considered. The first configuration is a flowshop that results in what we refer to as the Distributed Assembly Permutation Flowshop Scheduling Problem (DAPFSP).

The second problem is referred to as the Distributed Parallel Machine and Assembly Scheduling Problem (DPMASP), where unrelated parallel machines configure the production centers. Makespan minimization of the product on the assembly machine located in the assembly stage is considered as the objective function for all considered problems.

In this thesis some extensions are considered for the studied problems so as to bring them as close as possible to the reality of production shops. In the DAPFSP, sequence dependent setup times are added for machines in both production and assembly stages. Similarly, in the DPMASP, due to technological constraints, some defined jobs can be processed only in certain factories.

Mathematical models are presented as an exact solution for some of the presented problems and two state-of-art solvers, CPLEX and GUROBI are used to solve them. Since these solvers are not able to solve large sized problems, we design and develop heuristic methods to solve the problems. In addition to heuristics, some metaheuristics are also designed and proposed to improve the solutions obtained by heuristics. Finally, for each proposed problem, the performance of the proposed solution methods is compared through extensive computational and comprehensive ANOVA statistical analysis.

Resumen

Los sistemas de producción se enfrentan a retos globales en los que el concepto de fabricación colaborativa es crucial para poder tener éxito en el entorno cambiante y complejo en el que nos encontramos. Una característica de los sistemas productivos que puede ayudar a lograr este objetivo consiste en disponer de una red de fabricación distribuida en la que los productos se fabriquen en localizaciones diferentes y se vayan ensamblando para obtener el producto final. En estos casos, disponer de modelos y herramientas para mejorar el rendimiento de sistemas de producción distribuidos con ensamblajes es una manera de asegurar la eficiencia de los mismos.

En esta tesis doctoral se estudian los sistemas de fabricación distribuidos con operaciones de ensamblaje. Los sistemas distribuidos y los sistemas con operaciones de ensamblaje han sido estudiados por separado en la literatura. De hecho, no se han encontrado estudios de sistemas con ambas características consideradas de forma conjunta.

Dada la complejidad de considerar conjuntamente ambos tipos de sistemas a la hora de realizar la programación de la producción en los mismos, se ha abordado su estudio considerando un modelo bietápico en la que en la primera etapa se consideran las operaciones de producción y en la segunda se plantean las operaciones de ensamblaje.

Dependiendo de la configuración de la primera etapa se han estudiado dos variantes. En la primera variante se asume que la etapa de producción está compuesta por sendos sistemas tipo flowshop en los que se fabrican los componentes que se ensamblan en la segunda etapa (Distributed Assembly Permutation Flowshop Scheduling Problem o DAPFSP). En la segunda variante se considera un sistema de máquinas en paralelo no relacionadas (Distributed Parallel Machine and Assembly Scheduling Problem o DPMASP). En ambas variantes se optimiza la fecha de finalización del último trabajo secuenciado ($C_{\max}$) y se contempla la posibilidad que existan tiempos de cambio (setup) dependientes de la secuencia de trabajos fabricada. También, en el caso DPMASP se estudia la posibilidad de prohibir o no el uso de determinadas máquinas de la etapa de producción.

Se han desarrollado modelos matemáticos para resolver algunas de las variantes anteriores. Estos modelos se han resuelto mediante los programas CPLEX y GUROBI en aquellos casos que ha sido posible. Para las instancias en los que el modelo matemático no ofrecía una solución al problema se han desarrollado heurísticas y metaheurísticas para ello.

Todos los procedimientos anteriores han sido estudiados para determinar el rendimiento de los diferentes algoritmos planteados. Para ello se ha realizado un exhaustivo estudio computacional en el que se han aplicado técnicas ANOVA.

Los resultados obtenidos en la tesis permiten avanzar en la comprensión del comportamiento de los sistemas productivos distribuidos con ensamblajes, definiendo algoritmos que permiten obtener buenas soluciones a este tipo de problemas tan complejos que aparecen tantas veces en la realidad industrial.

Resum

Els sistemes de producció s'enfronten a reptes globals en què el concepte de fabricació col·laborativa és crucial per a poder tindre èxit en l'entorn canviant i complex en què ens trobem. Una característica dels sistemes productius que pot ajudar a aconseguir este objectiu consistix a disposar d'una xarxa de fabricació distribuïda en la que els productes es fabriquen en localitzacions diferents i es vagen acoblant per a obtindre el producte final. En estos casos, disposar de models i ferramentes per a millorar el rendiment de sistemes de producció distribuïts amb acoblaments és una manera d'assegurar l'eficiència dels mateixos.

En esta tesi doctoral s'estudien els sistemes de fabricació distribuïts amb operacions d'acoblament. Els sistemes distribuïts i els sistemes amb operacions d'acoblament han sigut estudiats per separat en la literatura però, en allò que es coneix, no s'han trobat estudis de sistemes amb ambdós característiques conjuntament. Donada la complexitat de considerar conjuntament ambdós tipus de sistemes a l'hora de realitzar la programació de la producció en els mateixos, s'ha abordat el seu estudi considerant un model bietàpic en la que en la primera etapa es consideren les operacions de producció i en la segona es plantegen les operacions d'acoblament.

Depenent de la configuració de la primera etapa s'han estudiat dos variants. En la primera variant s'assumix que l'etapa de producció està composta per sengles sistemes tipus flowshop en els que es fabriquen els components que s'acoblen en la segona etapa (Distributed Assembly Permutation Flowshop

Scheduling Problem o DAPFSP). En la segona variant es considera un sistema de màquines en paral·lel no relacionades (Distributed Parallel Machine and Assembly Scheduling Problem o DPMASP). En ambdós variants s'optimitza la data de finalització de l'últim treball seqüenciat ($C_{\max}$) i es contempla la possibilitat que existisquen temps de canvi (setup) dependents de la seqüència de treballs fabricada. També, en el cas DPMASP s'estudia la possibilitat de prohibir o no l'ús de determinades màquines de l'etapa de producció.

S'han desenvolupat models matemàtics per a resoldre algunes de les variants anteriors. Estos models s'han resolt per mitjà dels programes CPLEX i GUROBI en aquells casos que ha sigut possible. Per a les instàncies en què el model matemàtic no oferia una solució al problema s'han desenrotllat heurístiques i metaheurísticas per a això. Tots els procediments anteriors han sigut estudiats per a determinar el rendiment dels diferents algoritmes plantejats. Per a això s'ha realitzat un exhaustiu estudi computacional en què s'han aplicat tècniques ANOVA.

Els resultats obtinguts en la tesi permeten avançar en la comprensió del comportament dels sistemes productius distribuïts amb acoblaments, definint algoritmes que permeten obtindre bones solucions a este tipus de problemes tan complexos que apareixen tantes vegades en la realitat industrial.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CHAPTER 1

INTRODUCTION

Nowadays, manufacturing enterprises struggle to achieve competitive manufacturing systems to be able to meet new global challenges like market globalization, increasing product varieties, higher product customization, shorter lead-times, reduced product life cycles, etc. Therefore, the necessity of a collaborative manufacturing environment has become more sensible. Constructing a new manufacturing environment through a wisely composition of appropriate manufacturing systems is an efficient way to face the challenges. On the other hand, optimizing these manufacturing systems is a complex task that can significantly affect production performance. As a result, optimizing these production systems has received considerable attention and has become an active research topic in the recent years. Therefore, in this thesis we propose new methods to manage a manufacturing system able to tackle these recent global challenges and try to present different ways to find optimum or near optimum solutions.

In the remainder of this chapter, we briefly mention the motivation of the thesis, present the addressed problem and highlight the main contributions of our work.

1.1 Motivation for the thesis

Today, more than ever the manufacturing industry is faced with challenges like a tremendous competition and a rapidly changing environment. Market globalization, aggressive competition at a global scale, product life cycle reduction, increasing market demands for more innovation and demand variability, faster delivery, higher quality products, customers' push for higher product varieties and increased productivity through highly optimized production processes are some of the challenges faced by the manufacturing industry today.

Traditional manufacturing systems do not adequately face these challenges. For example, they are not flexible enough and cannot cope with market requirements due to being too slow and in most cases too costly in time response to predominantly random excitations of the markets (Butala and Sluga, 2002). Most of the information in traditional manufacturing systems is incomplete, incorrect and unreliable. Therefore, decision makers act more or less based on guessing and usually using the rule of thumb (Peklenik, 1997). The organizational structures of these systems are predetermined and rigid, because they are based on labor division and the optimization of performance is based on central planing and control (Westkaemper, 1997). Therefore, in order to meet these challenges, the manufacturing system has to be able to respond to the dynamic changes in the environment, capable of producing large product variants, increased levels of flexibility, responsiveness, reconfigurability, robustness and intelligence into manufacturing systems.

In order to face these new challenges, a shift of the manufacturing paradigm from traditional into new manufacturing prospects considering natural understanding is needed. Novel manufacturing control systems that are able to manage production changes and disturbances, both effectively and efficiently (Brussel et al., 1998) are needed to meet these challenges. Several effective approaches arise to incorporate increasing levels of flexibility, reconfigurability, and intelligence into manufacturing systems in order to meet challenges such as “flexible manufacturing” (Jha, 1991), “holonic manufacturing” (Brussel et al., 1999; Brussel, 2014), “agile manufacturing” (Goldman et al., 1994), “reconfigurable manufacturing” (Koren et al., 1999; Mehrabi et al., 2000), “fractal factory” (Warnecke, 1993), “bionic manufacturing systems” (Ueda, 1992), “distributed manufacturing systems” (Peklenik, 1992), etc.

A Flexible Manufacturing System (FMS)(Jha, 1991) is a manufacturing system in which there is some amount of flexibility that allows the system to react in case of changes, whether predicted or unpredicted. A FMS consists of several machine tools along with part and tool handling devices such as robots, arranged so that it can handle any family of parts for which it has been designed and developed. It can be changed or adapted rapidly to manufacture different products or components at different volumes of production. Flexible manufacturing systems are usually seen at their most when manufacturing components rather than finished products.

The Holonic Manufacturing System (HMS)(Brussel et al., 1999; Brussel, 2014) presents a new paradigm for next-generation manufacturing systems. HMS brings a different perspective, as it introduces a method for meeting the challenges of manufacturing environments for mass customization or low-volume and high-variety products. It also satisfies customers’ requirements according to the concept of Holon. Holons in HMS refer to key elements like machines, work centres, plants, parts, products, persons, departments. Furthermore, a Holon is autonomous, co-operative and sometimes intelligent. The interaction of holons determines the activity of each Holon in the system and there is no need to have a centralized mechanism. The theory of HMS focuses particularly on manufacturing control and manufacturing information technology. This system aims to provide a dynamic and decentralized manufacturing process and also effectively integrated with the human, therefore the changes can be made dynamically and continuously.

Agile manufacturing (Goldman et al., 1994) is a term applied to an organization. The aim of this system is to create the needed processes, tools, and to train the system so to have a strong focus on rapid response to the customer needs and market changes while maintaining standards of quality and controlling the overall costs involved in the production. In fact, this is a union of the organization, people and technology all together, which are integrated and coordinated. Agile production processes are designed flexible so they can change rapidly to create new or custom products using existing equipment, tools, labor and raw materials. Agile manufacturers can brace themselves for dramatic performance improvements in competitive market of these days.

A Reconfigurable Manufacturing System (RMS) (Koren et al., 1999) is a new manufacturing system model which gives the ability to adjust the production capacity and practically satisfy new market conditions and requirements. This system helps being cost-effective and very reactive to all the changes in market, product and system failures. RMS allows to be flexible in both producing varied parts and changing the system itself. The system can continuously improve itself and this improvement comes from adapting to modern technology, changing rapidly to host innovations and changes in product demand, not being throw away or replaced.

The concept of fractal factories (Warnecke, 1993) comes out of fractal features such as self-organization, dynamics and self-similarity. These factories are manufacturing companies made-up of small parts or fractal entities. In this system, consistency of system goals is ensured by participation and coordination. This process is applied through an inheritance mechanism to the system. A flexible and efficient information and navigation system constantly check target areas, reassess their position and progress and correct them if necessary, in order to support the system. Anyway, there still is lack of much study and effort to coordinate the each fractal's activity and run mechanisms that permit self-organization and dynamic restructuring.

The bionic manufacturing system (Ueda, 1992) is inspired of biological metaphors that focus on the self-organizing nature of element in the manufacturing system. Noticeable characteristic of these systems are autonomous and spontaneous behavior, harmony within hierarchically ordered relationships, self-recognition, self-growth and self-recovery. These features are integrated intelligently together to create a new manufacturing system to respond future manufacturing needs. The basic unit of these structure, is a cell which comprises all other parts. These similar cells have different functions and do multiple operations. Production units on the shop floor resemble biological cells.

In a Distributed Manufacturing System (DMS) (Peklenik, 1992) factories of various levels of manufacturing capabilities are equipped with diverse types of machines and, tools run in parallel and are scattered around many geographically different locations. This system also can be viewed as a multi agent system in which each manufacturing center is an agent to produce the whole or part of the ordered products by the customers. Studies and investigations (Rosenau, 1996; Wang, 1997) has proven that distributed manufacturing enables enterprises to achieve better product quality, lower production costs and reduced management risks. Studies show that the manufacturing costs (single location) are more costly than outsourced (multiple off-shore location) production (Mahdavi et al., 2008). For example, during the late 1980s, many Japanese manufacturers aggressively distributed shops into several Asian countries due to the sharp increase in domestic labor costs. Distributed workloads across multiple suppliers in comparison of only one supplier, significantly reduce management risks.

As mentioned before, nowadays one of the main challenges that each manufacturing system faces with is diversification of consumer needs (Hu et al., 2011). The manufacturing system should be able to adapt itself to unstable market situations quickly (Manzini et al., 2004). Increasing variety of products is one key method for the systems to deal with this type of challenges and to remain in the competitive market. Product variety has been recognized as an important source for manufacturers to increase their profits as well as the competitiveness in the market (Wang et al., 2013a). Product variety can satisfy customer requirements in two ways, either collectively through a family of product variants, or individually through a configurable product design (ElMaraghy et al., 2013). Products can be configured to satisfy individual customer requirements. Product variety is achieved through combinational assembly of different variants of modules (Wang et al., 2013a).

Assembly systems have been employed in recent years to provide flexible and quickly deployable solutions in order to deal with unpredictable changes following market trends (Ferreira et al., 2014). These systems are mostly used in mass production. Assembly systems are one of the key elements of effective mass production system (Chutima and Chimklai, 2012), and as such it has acquired great importance and is of considerable interest for many industries.

As mentioned before, the ability of distributed manufacturing and assembly systems to overcome recent challenges is clear. The idea is to use both of these systems together in a same manufacturing system. Each of the distributed manufacturing and assembly systems has their own advantages to face challenges and to help industries remain competitive. The combined advantages can make industries stronger, more flexible and bring more efficient and effective solution ways to deal with the challenges.

On the other hand, optimizing these complex systems consisting of more than one manufacturing system that can significantly affect production performance is more complex than considering just one of them. Scheduling the system can lead to an improved production performance. Scheduling is the process of arranging, controlling and optimizing work and workloads in a production or manufacturing process. It is an important tool for manufacturing, where it can have a major impact on the productivity of a process. Production scheduling aims to maximize the efficiency of the operation and to reduce costs. Given such circumstance, optimizing the manufacturing problems through scheduling has received considerable attention and has become an active research topic in recent years (T'Kindt and Billaut (2006); Pinedo (2009, 2012); Framinan et al. (2014), among others). Therefore, this thesis is focused on studying the scheduling problem through modeling and on proposing solution procedures to schedule these kind of systems.

1.2 Statement of the problem

In this thesis, a new model composed of distributed manufacturing and assembly system as two basic keys facing new challenges is studied. The studied model consists of two stages: the first is composed of various identical manufacturing factories which produce different components. The second stage is an assembly stage where these previously manufactured components are assembled into complete products via given assembly programs. To the best of our knowledge, this thesis is the first attempt to combine the distributed manufacturing and assembly systems.

Minimizing the makespan of the products in the assembly stage is considered as the objective function. Makespan is one of the most important criteria in every production system. The practical implication is obvious: minimizing the makespan leads to the minimization of the total production run (Framinan et al., 2002). Therefore, how to schedule to obtain the minimum makespan and enhance the overall performance is an important issue for the industries (Low et al., 2010). In the literature, special attention has been paid to makespan minimization and this criterion has been studied extensively in the scheduling literature. Some reviews are Lee et al. (1993); Framinan et al. (2002); Ruiz and Maroto (2005); Zobolas et al. (2009) and Low et al. (2010).

1.3 Thesis objectives

The main objective of this thesis is to schedule the presented production models by considering makespan minimization of the products that are assembled in the assembly stage. This criterion focuses on completing the last job/product as soon as possible which leads to the minimization of the total production run. Therefore production scheduling, with the objective of minimizing the makespan, is an important task in manufacturing systems. Two different problems are presented in this thesis. All problems consist of two stages: production and assembly. In the first problem, the shop configuration of all production factories in the first stage is modeled as a flowshop. Jobs are produced on the first stage and assembled into final products through a defined assembly program in a second assembly stage. This problem is referred to as the Distributed Assembly Permutation Flowshop Scheduling Problem or DAPFSP. A MILP is provided to solve the problem. Since, this is a $\mathcal{NP}$-Hard problem, some simple algorithms and heuristics in order to minimize the makespan of the products on the assembly stage are presented. The criteria of minimizing the makespan is considered only for the assembled products on the assembly stage. A schematic diagram of this problem is shown in Figure 1.1.

The first problem is modified by adding sequence dependent setup time (SDST) on the production and assembly machines. Logically, the complexity of the new problem is greater than the previous one after adding SDST into both stages. Therefore, some efficient heuristics and metaheuristics are introduced to solve it. The objective is the same as in the previous model. A schematic diagram of this problem considering SDST on both stages (DAPFSP-SDST) is shown in Figure 1.2.

In the second problem, all factories on the first stage consist of a set of unrelated parallel machines and a single assembly machine in the assembly stage. Jobs are processed on the first stage and assembled into final products through a defined assembly program in the assembly stage. This problem is referred to as the Distributed Parallel Machine Assembly Scheduling Problem or DPMASP. This third problem is studied in two different conditions: allowing and not allowing the production machines being left empty. The aim is to present MILP models for both problem conditions and to propose efficient constructive heuristics to report good results in order to minimize

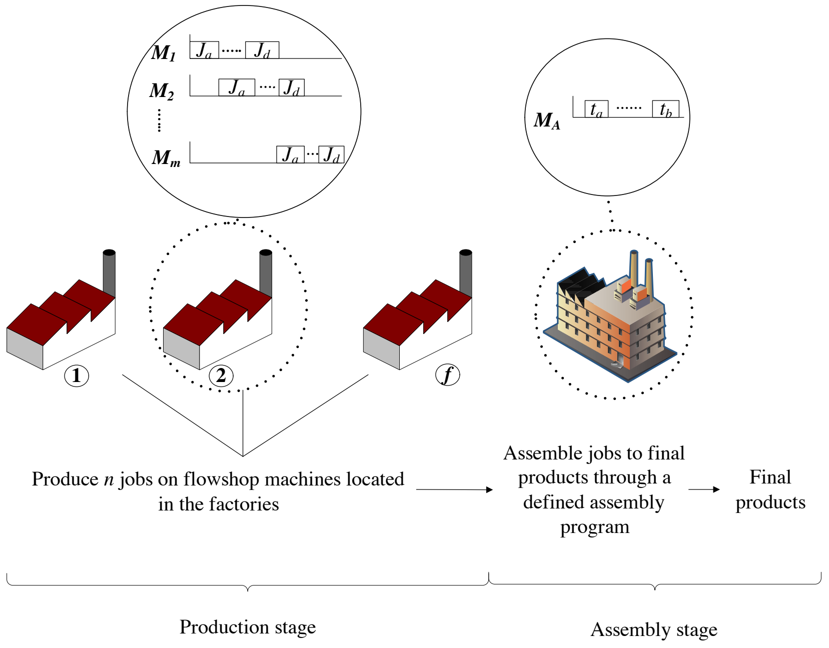


Figure 1.1: A schematic diagram of DAPFSP.

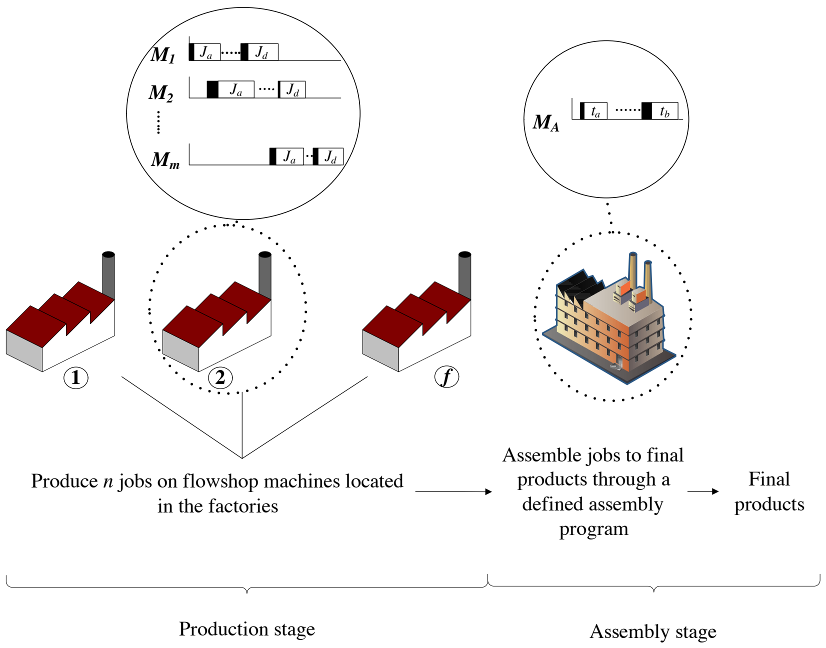
the makespan of products on the second stage. Figure 1.3 shows a schematic diagram of the problem.

1.4 Summary of contributions

Our contributions can be described as follows:

1. Introducing the Distributed Assembly Permutation Flowshop Scheduling Problem (DAPFSP). In the DAPFSP, the first stage consists of f identical production factories with a flowshop configuration each and the second stage is a single assembly machine.

    - A Mixed Integer Linear Programming model (MILP) is presented.
    - Three constructive algorithms are proposed.



M_1
J_a
J_d
M_2
M_m
M_A
t_a
t_b
1
2
f
Produce n jobs on flowshop machines located in the factories
Assemble jobs to final products through a defined assembly program
Final products
Production stage
Assembly stage

Figure 1.2: A schematic diagram of DAPFSP-SDST.

- A Variable Neighborhood Descent (VND) algorithm also has been designed.
- A large variety of instances in two different sizes (small and large) by considering several instance factors are generated to test the MILP and proposed algorithms.
- Two state-of-the-art commercial solvers, CPLEX and GUROBI are used to solve the MILP model for the small set of instances. Various test factors are considered to study the solution procedures.
- An exhaustive Chi-squared Automatic Interaction Detection tool (CHAID) is used to draw a decision tree to analyze the effect and interactions of the tested factors for CPLEX and GUROBI results.
- Both sets of instances, small and large, are used to test the constructive and VND algorithms. The results are analyzed through a

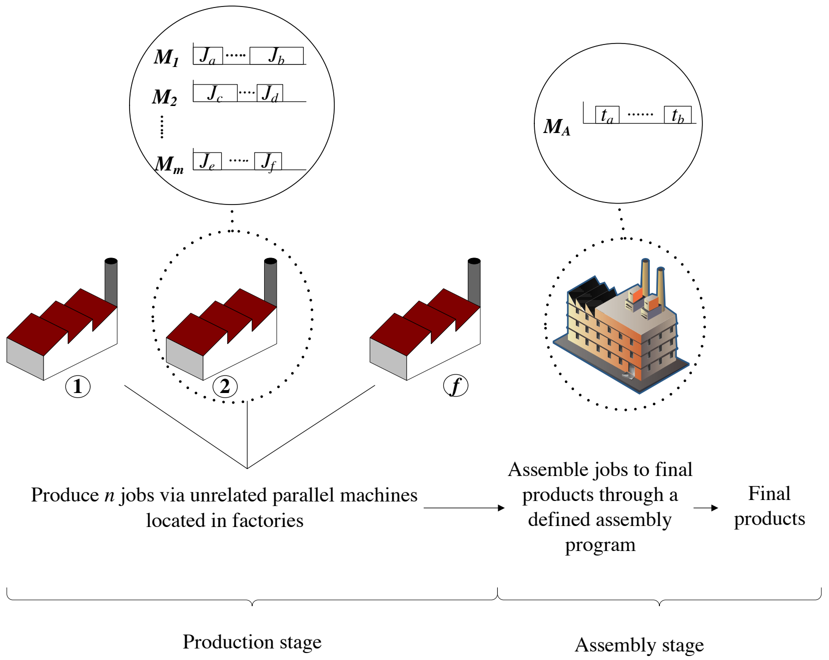

M_1
J_a
J_b
M_2
J_c
J_d
M_m
J_e
J_f
M_A
t_a
t_b
1
2
f
Produce n jobs via unrelated parallel machines located in factories
Assemble jobs to final products through a defined assembly program
Final products
Production stage
Assembly stage

Figure 1.3: A schematic diagram of DPMASP.

comprehensive ANOVA statistical analysis.

2. We also add Sequence Dependent Setup Times (SDST) for all production and assembly machines in both stages of the DAPFSP.

   - Two simple constructive heuristics and two different metaheuristics VND and Iterated Greedy (IG) are proposed.
   - Two different solution representations are used in both simple constructive heuristics and metaheuristics.
   - A complete calibration and analydsis through a Design of Experiments (DOE) approach is carried out to select the best levels of the factors for the IG.
   - In the process, important knowledge of the studied problem is ob-

tained as well as some simplifications for the powerful IG methodology which results in a simpler approach with less parameters.

- The performance of the proposed methods is compared through extensive computational and statistical experiments.

3. We also study the Distributed Parallel Machine and Assembly Scheduling Problem (DPMASP) which consists of two stages, production and assembly. There is a set of distributed identical factories, each one with a set of unrelated parallel machines at the production stage and a single assembly machine in the assembly stage.

    - Two assumptions are considered for the problem: a) due to technical constraints, machines can not be left empty and b) empty machines at factories are permitted.
    - Two MILP models by considering each mentioned assumptions are presented for the problem and two state-of-the-art solvers CPLEX and GUROBI are used to solve them.
    - Different fast and high-performing heuristics are proposed for the problems.
    - Different sized instances are designed to test the MILP and proposed heuristics and the results are analyzed through a comprehensive ANOVA statistical analysis.

1.5 Outline of the thesis

The remainder of the thesis is organized as follows:

Chapter 2: Presents the necessary background for the thesis. Notions of scheduling are introduced, followed by an overview of scheduling problems. Some concepts of the distributed manufacturing and assembly systems are presented. A brief introduction of optimization and finally some solution methods for the scheduling problems are enumerated and explained.

Chapter 3: A detailed explanation of Distributed Assembly Permutation Flowshop Scheduling Problem (DAPFSP) is presented. A literature review on problems close to the DAPFSP is provided. A Mixed Integer Linear Programming model is presented along with three constructive heuristics and a Variable Neighborhood Descent (VND) algorithm. The MILP, simple constructive heuristics and the VND are tested.

Chapter 4: An extension of the problem studied at Chapter 3 is presented by adding sequence dependent setup times to production machines on the first stage and to assembly machine on the second stage. The problem is discussed in detail and a literature review is provided. Two simple constructive heuristics and two different metaheuristics, VND and Iterated Greedy (IG) are designed for the problem. Two different solution representations are applied to all solution methods. A Design Of Experiments (DOE) approach is carried out for calibration. The algorithms are tested through a set of computational experiments.

Chapter 5: A new problem is presented. In this case, in the first stage instead of a flowshop configuration, we have a set of unrelated parallel machines in all production factories. Due to technological constraints, machines cannot be idle and some jobs can be processed only in certain factories. A literature review of related problems is presented. A mathematical model is developed and also two different constructive heuristics are proposed to solve this setting.

Chapter 6: studies the presented model in Chapter 5 by relaxing the special constraint that no machine at any factory might be empty due to technological or economical constraints. A mathematical model is presented and four simple, fast and high performing heuristics are proposed.

Chapter 7: This chapter presents a general discussion on the all obtained results in this thesis.

Chapter 8: A general conclusion that includes the description of the search findings, a discussion of the results and possible future work are presented in this final chapter.

CHAPTER 2

BACKGROUND

In today's complex manufacturing reality, there are multiple product lines and complex products consisting of various components or sub-assembled parts. Each product requires many different steps and machines for its manufacturing and sometimes there is an economic justification to operate multiple plants and to collaborate with partners across the country and around the globe. Therefore, there is a need to find a way to successfully manage resources in order to produce products in the most efficient possible way. For this purpose, to design a production schedule is strongly recommended. Most of these problems are hard to solve and many different techniques have been developed for tackling them.

In this chapter we review the scheduling methods and manufacturing systems used in the thesis and some of the scheduling problem solution techniques. It is not possible to cover, in this chapter, the huge research effort carried out so far in the scheduling area, therefore, a brief introduction overview is considered instead. The chapter starts by presenting a short history of scheduling and a brief survey on scheduling problems classification. Next, the two manufacturing systems in which our studied problems are based: distributed and assembly are described in detail. The chapter is followed by a short overview of some optimization techniques and methods for scheduling problems.

2.1 Scheduling

Scheduling deals with the problem of allocating one or several limited resources to activities over a certain time period, subject to specific constraints (Pinedo, 2012). The idea of resources and activities takes many forms. Typically, resources can be machines or raw materials for the manufacturing or stretches of a track in the train scheduling or the date and the venue in a sport context. Activities may be operations or processing to be carried out in a production process, the train running from a point to another on a track, or a game to be played in a football competition.

The goal of scheduling is to optimize some objective function depending on the application domain at hand. For example in manufacturing environments the function to optimize is usually the maximum completion time, i.e., the time elapsed since the beginning of the first task till the end of the last one. This is commonly referred to as makespan. Other possibilities are to minimize the tardiness of jobs, total flowtime, etc.

Scheduling problems appear in several different domains such as production planning, timetabling, product configuration, transportation, distribution, information processing and communications. Scheduling and sequencing problems have been studied for many decades. The pioneering works of Henry Gantt (1861-1919) were the starting point of scheduling in manufacturing, after the theory of scientific management of Frederick Winslow Taylor. Since the early 1950s (Johnson, 1954; Jackson, 1956; Smith, 1956), the theory and application of scheduling has grown into an important field of research. *Naval Research Logistic Quarterly* published the first scientific paper on scheduling in the early 50s (Johnson, 1954). Later, in the 1960s and 1970s a sharp growth occurred in the area and has retained the momentum ever since. With computing development automated scheduling tools are used for scheduling systems from the 80s (Pinedo, 2012).

Scheduling covers a very wide family of problems and has been a large research area for decades. A lot of work has been done to model, introduce, classify, and solve scheduling problems. In the next section an overview of scheduling classification is presented.

2.2 Classification of scheduling problems

Basically there are three class as of scheduling problems with respect to the structure or configuration of the machines in the manufacturing shop, which are: single-machine problems, single-stage multi-machine problems and multi-stage multi-machine problems.

2.2.1 Single-machine problems

In a single-machine scheduling problem, a group of tasks have to be scheduled into a single machine or resource. It has attained most attention in theoretical scheduling studies. Understanding existing rules and theories in this area paves the way for a better analysis of the multi-machine systems.

In a single machine scheduling problem the following assumptions generally have been applied in order to simplify, formulate and solve scheduling problems:

1. The machine is continuously available during the scheduling period.
2. The machine can process one job at the same time.
3. The processing time of the job on the machine is a positive integer value, known in advance, deterministic and independent.

4. In non-preemptive scheduling, jobs have to finish processing without interruption. Otherwise, in preemptive scheduling, jobs may be removed from the machine before completion to be finished at a later time.

2.2.2 Single-stage multi-machine problems

Parallel machine scheduling problems (PMSP) are a type of problems which consist of a single production stage with multiple machines disposed in parallel. It has wide applications on manufacturing systems in the real world. It has been broadly studied during the last decades, mainly because of the aforementioned real world applications (Lin et al., 2011a; Ying, 2012).

In the parallel machine scheduling problem multiple machines are available. In this problem the aim is to schedule a set N of n jobs on a set M of m machines. These machines are disposed in parallel and jobs have to be assigned and scheduled in these machines. Jobs have to be assigned and processed by exactly one out of the m parallel machines. A job cannot be processed on more than one machine at the same time. Therefore, the processing of the jobs has to start and finish on a given machine (non-preemption). A machine cannot process more than one job at the same time. The processing time, p_{ij}, is the time required for machine i to process job j. This time is a known, deterministic and positive. There are three different types of PMSP:

- Parallel identical machines (P): where jobs are scheduled on multiple identical machines. The processing time of each job is the same on any machine, so that for a set M of m machines, p_{1j}=p_{2j}= $\ldots$ = p_{mj}= p_j.

When the machines are not equal, then we face to the non-identical parallel machine scheduling problems, which can be divided into two cases:

- Uniform related machines (Q): Machines that work with different speeds and the processing time of a job j on machine i follows the relationship $p_{ij} = p_j/s_i$, where s_i represents a different speed for machine i

when processing the jobs. In other words, a job processing time depends on the machine speed factor.

- Unrelated machines (R): machines have different characteristics and are unrelated to each other. Therefore, the processing time of each job j on each machine i is different and denoted by p_{ij} (Allahverdi et al., 2008; Lin et al., 2011b).

2.2.3 Multi-stage multi-machine problems

This type of scheduling problem is a vast and applicable topic in the literature. These problems are divided into four groups of scheduling problems as described bellow:

- ***Flowshop problems***: A typical scheduling problem which contains m different machines arranged in series where a set of n jobs has to be processed. Each job needs m operations and each operation must be performed on a separate machine. Therefore, each of the jobs has to go through each machine on the shop floor. The flow of the work is unidirectional; every job has to be processed on each machine in a given prescribed order. All jobs are available at time zero. Each operation is to be performed on a specific machine. Each machine can process at most one job at the same time and each job can be processed by at most one machine at a time. Jobs have different processing times for each machine.

  If an arbitrary sequence of jobs on each machine is considered, then there are $(n!)^m$ possible schedules for the problem. Finding the best schedule in a problem with such a high amount of feasible schedules is difficult. Therefore, researchers have focused on reducing this number of feasible schedules as much as possible without compromising the final solutions. Therefore, it is assumed that the order in which a job passes through the machines is the same for all the machines in most of the flowshop scheduling literature. In the other words, each machine has the same sequence of jobs. Considering this assumption, the number of feasible schedules is now $n!$. This type of flowshop scheduling problem by considering the reduction of feasible schedules is known as *permutation flowshop problem*.

- ***Jobshop problems***: Each job consists of a chain of operations and has a specified machine visitation order, which determines the required machine for the tasks and the processing times. Each operation has a single predecessor and requires a certain machine. In the jobshop the visitation order for the machines depends on each job and it is not the same for all jobs like in a flowshop. Some constraints are considered on jobs and machines. For example, there are no precedence constraints among operations of different jobs; interruption of operations is not allowed (non-preemption); each machine can process only one job at a time and also each job can be performed only on one machine at a time.

- ***Openshop problems***: A given set of jobs must be processed by a given set of machines where there is no ordering constraints. There are m machines that perform different operations and n jobs which consist of m operations. At any time each machine may process at most one job at a time and one job may be processed by at most one machine at a time. In openshop problems there are no precedence relations between the operations of each job and there is no restriction on the order in which the operations for a job are to be processed (Coffman, 1976; Conway et al., 1968). Therefore, the order of operations is immaterial and no order is given at all in the openshop problem.

- ***Group shop environments***: It is a generalization of the classical jobshop and openshop scheduling problems. In Group Shop Scheduling Problems (GSP) there are m machines and n jobs where each job consists of a set of operations that must be processed on specified machines without interruption. The operations of each job are partitioned into g groups $G = \{G_1, \ldots, G_g\}$ on which a total precedence order is given. The operations in the same group are unrestricted while operations in two distinct groups most satisfy the precedence relationship between the groups. There is a given total order on the groups of each job. For example, the operations of the first group of a job have to be processed before the operations of the second group.

Assembly systems and distributed manufacturing systems are two powerful manufacturing systems that consist of different compositions of the previously reviewed scheduling problems. These two manufacturing systems are used in the studied problems of this thesis. Therefore, an introduction to each system is provided in the following sections.

2.3 Assembly scheduling problems

In the recent years, assembly systems have been profusely employed in mass production. They have been widely deployed in various manufacturing systems to increase the flexibility and the capability of producing a larger variant of products in order to meet a wider array of market demands. These types of problems are referred to as Assembly Scheduling Problems (ASP).

An assembly system is a flow oriented manufacturing system. Various operations are performed independently to produce different components, where these components and the bill-of-material parts are attached one-by-one to a unit in a sequential way by a series of workers or machines to manufacture a finished product. All the operations and the assembly program to fully produce the product are identified and the processing times are evenly assigned to workers/machines. Normally, a high variety of finished products are made from different combinations of produced components which are produced in assembly systems.

There are several types of assembly systems such as: Single model assembly, batch assembly, mixed model assembly for make-to-stock, mixed model assembly for make-to-order, postponement assembly and one station assembly (Thomopoulos, 2014). In a single model assembly, the assembly line is dedicated to assemble just one model. In batch assembly, the models are assigned to the line in pre-assigned sizes, where each model is run until its inventory is at a specified level. Mixed model assembly for make-to-stock occurs when more than one model of a product is assigned to the same assembly line at the same time. Mixed model assembly for a make-to-order manufacturer occurs when the manufacturer's product is offered with a series of features and options. Postponement assembly is a supply chain strategy that could apply to manufacturers that fabricate products with a series of features and options. This assembly strategy is applied for manufactures where the customer orders are for a particular combination of the options. One station assembly is applied when all needed assembly work to complete a unit is assigned to one person in one station.

In this thesis we work with the mixed model assembly for make-to-order which is carried out in one station assembly. In this assembly system, each customer order specifies the option for each feature of the product. This way, every customer order is unique and often no two orders are the same. The customer order is called a product. In the studied problems in this thesis, the assembly operations are performed at a single location on a single machine. The typical operation in the single-station assembly machine involves the placement of the base part at the workstation where various components are added to the base.

For manufacturers, the benefits of assembly line production are enormous such as producing diverse products, reducing the skill requirements for line workers, increasing production and better uniformity. Additionally, scheduling in distributed manufacturing systems is studied in this thesis and a brief introduction is presented in the next section.

2.4 Distributed manufacturing systems

On today's global and highly competitive market, enterprises must be aware of current market opportunities and react to customers' demands quickly and properly. Increasing product diversity over time expands associated risks and costs, which are sometimes prohibitive. Therefore, the existence of multiple entities for distributing responsibilities, results in a scenario in which risks and costs become acceptable and market opportunities can be reaped. In most of the cases, a single production center cannot adequately answer the market requirements such as a rising variety and complexity of products or product individualization because of the structural rigidity, deterministic approach to decision making and hierarchical allocation of competencies (Sluga et al., 1998).

In order to face the mentioned challenges Distributed Manufacturing Systems (DMS) is one of the possible alternatives. The DMS environments are constructed based on the new organizational structures. These new structures are mostly geographically distributed, composed by several independent commercial partners or production centers with various machines and tools running in parallel so that each of them works with its own specialization and resources dedicated to specific functions in the product life cycle (Mishra and Shah, 2009; Chan and Chan, 2010; Mikos et al., 2011). As for the benefits of DMS we can mention higher product quality, lower production costs, reduced risks and offshore outsourcing (Wang, 1997; Kahn et al., 2004; Chan et al., 2005b; Mahdavi et al., 2008).

Scheduling in DMS is more complicated than scheduling single manufacturing systems. In a single manufacturing system, only job schedules for each set of machines has to be defined. In DMS, factory selection (process plan) for each job is also added as an important decision. Therefore in a DMS, two decisions have to be taken: 1) Allocation of jobs to factories, 2) Scheduling of the jobs allocated to each factory.

2.5 Optimization

An optimization problem consists of finding the solution or solutions which are optimal or near-optimal among all feasible solutions with respect to some goals and constraints.

Scheduling problems are a type of combinatorial optimization problems. A combinatorial optimization problem is either a minimization or a maximization problem over a discrete combinatorial structure. These problems are defined by decision variables, constraints and objective functions. A domain of defined set of values is determined for decision variables that are the unknowns of the problem and must be fixed. Through the constraints, assignments of the variables are defined and the optimization function decides the best assignments.

To find high-quality solutions for a problem is the purpose of optimization algorithms. In an optimization problem, the objective is, identifying either optimal solutions x^*, near-optimal solutions $x \in X$, where $f(x) - f(x^*)$ is small (in the case of minimization).

Most scheduling problems are computationally difficult and hard to solve optimally and need complex algorithms. The complexity of the problem is of great interest. Computational complexity theory deals with the time requirements to solve a given problem. In terms of scheduling problems, we are concerned with the time required to solve the problem to optimality. The running time of the algorithms developed to solve these problems is considered. The running time describes the number of operations that must be performed as a function of the number of input variables to the algorithm. A problem is polynomial (class $\mathcal{P}$), if the algorithm to solve it can be run in polynomial time on a deterministic Turing machine. The problem is considered $\mathcal{NP}$ (Non-deterministic Polynomial) if its solution can be found in polynomial time on a non-deterministic Turing machine.

The hardest class of problems to solve in $\mathcal{NP}$ are known as $\mathcal{NP}$-complete problems. Cook (1971) defines a problem as $\mathcal{NP}$-complete if it is $\mathcal{NP}$, and every other problem in $\mathcal{NP}$ is reducible to it in polynomial time. $\mathcal{NP}$-complete problems are considered to be harder than other problems in $\mathcal{NP}$ because, presently, all known $\mathcal{NP}$-complete problems have exponential running times with respect to the problem size, and are therefore likely not to be in $\mathcal{P}$.

An $\mathcal{NP}$-hard problem is a combinatorial problem whose decision version in $\mathcal{NP}$-complete. It has been proven that most classes of scheduling problems, even greatly simplified ones, belong to $\mathcal{NP}$-hard class (Karp, 1972; Engels et al., 2001). There are currently no polynomial time algorithms known to solve most scheduling problems to optimality. Therefore many different techniques have been developed for tackling them. Many researchers have instead focused on developing heuristics or approximation techniques that lead to sub-optimal solutions but in shorter CPU times (Ng et al., 2010; Zachariadis and Kiranoudis, 2010; Rudek, 2011; Lee et al., 2014).

2.6 Solution methods

There are various methods proposed for solving scheduling problems. In the following sections, some of the methods of interest in this thesis are briefly reviewed.

2.6.1 Exact solution procedures

Exact methods are guaranteed to find an optimal solution, but typically become impractical when faced with problems of any significant size or large sets of constraints. There are several types of exact methods and tools such as, Mathematical Programming (MP), Bounded Enumeration (BE), Branch and Bound (B&B), etc. In this thesis we are interested in mathematical programming.

Mathematical programming is a very useful tool for solving complex problem such as these than can be modeled as an objective function with a set of mathematical constraints. A wide variety of research disciplines currently use

MP techniques to aid in complicated decision-making. Because mathematical programming is concerned with finding optimal ways of using limited resources to achieve an objective, it is often simply referred to as optimization. Scheduling problems may be solved using Mathematical Programming, in particular Linear Programming (LP) or Mixed Integer-Linear Programming (MILP).

Linear Programming (LP) problems are MP formulations where the objective function and constraints are linear functions.

There are MP problems where it is necessary to restrict the decision variables to integer or binary values. These problems are called Mixed Integer-Linear Programming (MILP), and are often much harder to solve than LP problems. This is because instead of having feasible solution points at the easily computed extreme points of the feasible region, they are instead usually internal and more difficult to locate. In most cases the Branch and Bound technique is used to find the integer solution in MILP. Using MILP technique has several advantages. First of all MILP produces exact optimal solutions instead of approximate ones. Second, being a general-purpose optimization method, software solver tools such as CPLEX (cpl, 2014) or GUROBI (gur, 2015) are available to efficiently solve an MILP problem once it has been formulated. The disadvantage of using MILP techniques is that they are $\mathcal{NP}$-hard, and therefore may be infeasible to use for solving larger scheduling problems.

2.6.2 Heuristics

Whereas exact solution methods are guaranteed to find the optimal solution, heuristic methods sometimes find optimal solutions, but more often find simply "good" solutions. Heuristic methods typically require far less time and/or space than exact methods. Heuristics are rules for deciding which action has to be taken at any step.

Heuristics in scheduling are often referred to as scheduling rules or dispatch rules (Pinedo, 2012). The definition of these rules is often quite complex, and most are tailored for a specific type of problem with a very specific set of constraints and assumptions. Heuristics may be deterministic (they end up with the same result every time) (Johnson, 1954; Palmer, 1965; Campbell et al., 1970; Gupta, 1971, 1972; Nawaz et al., 1983) or they may be stochastic (each time they are run they may produce a different result)(Dresbach, 1994; Ciuprina et al., 2002; Boardman and Trappenberg, 2006). They may execute one rule at a time, or they may be capable of parallel decisions. Hybrid algorithms may combine multiple heuristics (Williams, 1983; Prosser, 1993; Atighehchian et al., 2009).

Scheduling heuristics operate on a set of tasks and determine when each task should be executed. If a task may be executed in more than one execution mode or on any one of a set of resources, the heuristic must also determine which resources and/or execution mode to use.

2.6.3 Metaheuristics

As mentioned before, there is no guarantee to find an optimal solution or any guaranteed bound through any exact solution method (deterministic) in a "reasonable" time limit for larger problems. Among the methods to find solutions for many hard optimization problems, metaheuristics are widely recognized as efficient procedures and can be used to find satisfactory solutions for such problems. A metaheuristic is an algorithm which is designed to find solution for approximately a wide range of hard optimization problems. The algorithm of the metaheuristic does not require problem specific knowledge. "Meta", the greek prefix which is presented in the name, indicates that these types of algorithms are "higher level" heuristics, in contrast with the heuristics designed for specific problems. Complex problems in a wide range of settings from industry, services, finance, transportation to production management, etc, can be solved with metaheuristics.

There are some shared characteristics for most the metaheuristics such as: They might be inspired from nature or physical processes; stochastic components involving random variables; the gradient or Hessian matrix of the objective function is not usually applied; there are several parameters in the metaheuristic algorithms that need to be calibrated to the problem at hand (Glover and Kochenberger, 2003; Dréo et al., 2006; Siarry and Michalewicz, 2008; Gendreau and Potvin, 2010).

There is a great interest on using metaheuristics in the last thirty years. We can try to point out some of the steps that have marked the history of metaheuristics. Kirkpatrick et al. (1983) was one of the pioneers in the proposition of the Simulated Annealing method (SA) as a metaheuristic. Glover (1986) and Farmer et al. (1986) proposed Tabu Search (TS) and the artificial immune system, respectively. The first patent on genetic programming was presented by Koza, in 1988 and then published in 1992. A well known genetic algorithm book is published by Goldberg (1989). The innovative work on ant colony optimization was presented by Dorigo (1992). Walker et al. (1993) proposed a algorithm based on bee colonies for the first time also the particle swarm optimization by Kennedy and Eberhart (1995). Iterated Greedy (IG) a very simple and effective metaheuristic is proposed by Jacobs and Brusco (1995). In 1997, Mladenović and Hansen proposed Variable neighborhood search (VNS).

The development of metaheuristics has been fostered by the increasing processing power of computers, the development of parallel architectures and hardware improvements (Boussaïd et al., 2013). Metaheuristics might be divided in to two groups: Single-Solution Based Metaheuristics and Population-Based Metaheuristics.

Single-solution based metaheuristics, sometimes also called trajectory based methods, start with a single initial solution and move in the search space by describing a trajectory to improve the objective value. Some of them can be seen as “intelligent” extensions of local search algorithms. For example we can mention Simulated Annealing, Tabu Search, Greedy Randomized Adaptive Search Procedure (GRASP), Variable Neighborhood Search, Guided Local

Search and Iterated Local Search, and their variants.

Population-based metaheuristics deal with more than one solution (i.e., a population). Most used methods in population-based metaheuristics are related to Evolutionary Computation (EC) and Swarm Intelligence (SI). EC algorithms are inspired by Darwin's evolutionary theory, where a population of solutions change through mutation and crossover operators. The idea of SI is to generate computational intelligence through exploiting simple analogies of social interaction, rather than purely individual cognitive abilities.

2.7 Conclusions

The concepts and the related issues to the considered problems are reviewed in this chapter. This serves as a reduced basic background on the concepts and features of the presented models. The chapter starts with an explanation on the concept of scheduling and the classification of scheduling problems. The concept of assembly scheduling and distributed manufacturing systems are explained. A brief explanation on optimization is presented. A brief discussion on the existing solution methods to solve problems has been presented.

CHAPTER 3

THE DISTRIBUTED ASSEMBLY PERMUTATION FLOWSHOP SCHEDULING PROBLEM

“ The contents of this chapter are taken from the publication: Hatami, S., Ruiz, R., and Andrés-Romano, C. (2013). The distributed assembly permutation flowshop scheduling problem. *International Journal of Production Research*, 51(17):5292–5308.”

Nowadays, improving the management of complex supply chains is key to become competitive in the twenty-first century global market. Supply chains are composed of multi-plant facilities that must be coordinated and synchronized to cut waste and lead times. This chapter proposes a Distributed Assembly Permutation Flowshop Scheduling Problem (DAPFSP) with two stages to model and study complex supply chains. This problem is a generalization of the Distributed Permutation Flowshop Scheduling Problem (DPFSP). The first stage of the DAPFSP is composed of f identical production factories. Each one is a flowshop that produces jobs to be assembled into final products in a second assembly stage. The objective is to minimize the makespan. We present first a Mixed Integer Linear Programming model (MILP). Three constructive algorithms are proposed. Finally, a Variable Neighborhood Descent (VND) algorithm has been designed and tested by a comprehensive ANOVA statistical analysis. The results show that the VND algorithm offers good performance to solve this scheduling problem.

3.1 Introduction

Assembly systems have been widely studied in the last decade given their practical interest and applications. An assembly flowshop is a hybrid production system where various production operations are independently and concurrently performed to make parts that are delivered to an assembly line (Koulamas and Kyparisis, 2001). In assembly systems, a wide variety of final products can be made from a given number of different assembled parts. Assembly programs represent relationships between the different parts which must be assembled from a set of suppliers.

Nowadays a single supplier or production factory is rare. As a matter of fact, production systems with more than one production center (named distributed manufacturing systems) are quite usual as they play an important role in practice (Moon et al., 2002). The benefits of distributed manufacturing systems include achieving higher product quality, lower production costs and fewer management risks (Wang, 1997; Kahn et al., 2004; Chan et al., 2005b). From a manager's point of view, scheduling in distributed systems is more complicated than in single-factory scheduling problems. In single-factory problems, the only objective is to find a job schedule for a set of machines, while an important additional decision in the distributed problem is allocating jobs to suitable factories. Therefore, two decisions have to be made; job allocation to factories and job scheduling at each factory. Different job allocations to different factories result in different production schedules, which consequently affects supply chain performance (Chan et al., 2005b).

This chapter contemplates flowshop scheduling as a production system for each factory or supplier in the distributed problem. The flowshop scheduling problem (FSP) is composed of a set of M of m machines where each job of a set N of n jobs must be processed in each machine. The number of operations per job is equal to the number of machines. The i^{th} operation of each job is processed in machine i. Therefore, one job can start in machine i only after it has been completed in machine $i - 1$, and if machine i is free. The processing times of each job in the machines are known in advance, non negative and deterministic.

In FSPs, a number of assumptions are made (Baker, 1974): all jobs are available for processing at time 0; machines are continuously available (no breakdowns); each machine can process only one job at a time; each job can be processed in only one machine at a time; once the processing of a given job has started in a given machine, it cannot be interrupted and processing continues until completion (no preemption); set-up times are sequence independent and are either included in the processing times or ignored; infinite in-process storage is allowed. In the FSP, there are $n!$ possible job permutations for each machine. Therefore, the total number of solutions for a flowshop problem with m machines is $(n!)^m$. To simplify the problem, it is assumed that all machines have the same job permutation. In other words, if one job is at the j^{th} position on machine 1, then this job has to be at the j^{th} position on all other machines as well. With this simplifying assumption the FSP is referred to as Permutation Flowshop Scheduling Problem (PFSP) with $n!$ possible solutions.

This chapter studies the Distributed Assembly Flowshop Scheduling Problem (DAPFSP). It is a combination of the DPFSP and the Assembly Flowshop Scheduling Problem (AFSP), and consists of two stages: production and assembly. The first stage consists of a set F of f identical factories or production centers where a set N of n jobs have to be scheduled. All factories are capable of processing all jobs and each factory is a PFSP with a set M of m machines. Factories are assumed to be identical. Processing times are denoted by p_{ij}, $i \in M, j \in N$. The second stage is a single assembly factory with an assembly machine, M_A, which assembles jobs by using a defined assembly program to make a set T of t different final products. Each product has a defined assembly program; in other words, each product consists of some defined jobs. N_h and J_j are used, respectively, to represent product h assembly program and the jobs that belong to the product h assembly program, $N_h : \{J_j\}, j \in N_h$. Each product h has $|N_h|$ jobs and job j is needed for the assembly of one product. Therefore, $\sum_{h=1}^{t} |N_h| = n$. Product h assembly can start only when all jobs that belong to N_h have been completed in the factories. The considered objective is to minimize the makespan at the assembly factory.

The next section presents a short literature review. Section 3.3 provides a Mixed Integer Linear Programming (MILP) model to solve the considered problem. Section 3.4 introduces three constructive heuristics, while Section 3.5 presents an iterative method based on Variable Neighborhood Descent (VND) to improve results further. Section 3.6 describes a complete computational evaluation of the MILP model and proposed algorithms, where the performance of the proposed approaches is discussed in order to assess the influence of the number of jobs, machines, factories, products and some solver options on the results. Finally, Section 3.7 offers conclusions and remarks for this chapter.

3.2 Literature review

The DPFSP can be viewed as a generalized version of the PFSP. This problem is one of the most researched topics in the scheduling literature (Dong et al., 2009; Zobolas et al., 2009; Laha and Sarin, 2009; Vallada and Ruiz, 2010; Xu et al., 2011; Zhang and Li, 2011; Chen et al., 2012; Pan and Ruiz, 2012; Pinedo, 2012).

In the PFSP, more attention has been paid to makespan minimization. The practical implication is obvious: minimizing the makespan leads to the minimization of the total production run (Framinan et al., 2002). There are some proposed effective rules and algorithms for the PFSP (Johnson, 1954; Nawaz et al., 1983). A comprehensive review and evaluation has been made by Ruiz and Maroto (2005), Vallada et al. (2008) and Pan and Ruiz (2013).

Regarding the assembly scheduling problem, Lee et al. (1993) presented a three-machine assembly-type flowshop scheduling problem by considering makespan minimization as the objective function. In their considered model, each product is composed of two types of jobs, where type a and b are processed by machine M_a and M_b, respectively, and machine M_2 assembles the two jobs into a product. These authors also present a branch-and-bound solution scheme and an approximate solution procedure. Later, Potts et al. (1995) extended the model of Lee et al. (1993) by considering m parallel production machines instead of the first two production machines. They apply the compact vector summation technique to find approximated solutions with worse-case absolute performance guarantees. Hariri and Potts (1997) developed a branch-and-bound algorithm for the same model as Potts et al. (1995). Moreover, Tozkapan et al. (2003) considered a two-stage assembly scheduling problem by minimizing the total weighted flow time as an objective function. They developed a lower bound and a dominance criterion, and incorporated them into a branch-and-bound procedure. They also presented a heuristic procedure to find an initial upper bound. Al-Anzi and Allahverdi (2006) addressed the model presented by Tozkapan et al. (2003) and minimized the total completion time of all the jobs. They used metaheuristics to solve their model and proposed simulated annealing (SA), tabu search (TS), and hybrid tabu search heuristics for general cases.

Despite the innumerable literature related to PFSP and AFSP, there are few studies about the distributed problems. Jia et al. (2002) reported a web-based system to enable production scheduling (a job shop problem) for the distributed manufacturing environment and a Genetic Algorithm (GA) was adopted to solve the problem. Jia et al. (2003) presented a modified GA to deal with distributed job shop scheduling problems. Later, Jia et al. (2007) proposed a new approach to determine good combinations of factories to manufacture jobs. An adaptive GA for distributed scheduling problems was proposed by Chan et al. (2005b). The same authors proposed a GA with dominant genes for solving distributed scheduling problems in an FMS environment in Chan et al. (2006a). Furthermore, Chan et al. (2006b) proposed a GA to deal with distributed flexible manufacturing system (FMS) subject to machine maintenance constraints. Naderi and Ruiz (2010) introduced the DPFSP for the first time. They developed six different MILPs for the considered problem and proposed two simple factory assignment rules and 14 heuristics based on dispatching rules, effective constructive heuristics and VND methods. Liu and Gao (2010) proposed an electromagnetism-like mechanism (EM) algorithm for the same problem. The same authors, in Gao and Chen (2011a) proposed a GA-based algorithm, denoted by GA-LS, Gao and Chen (2011b) a constructive heuristic algorithm enhanced with a dispatching rule, Gao et al. (2012b) a knowledge-based genetic algorithm and Gao et al. (2012a) a Variable Neighborhood Descent (VND) algorithm. Later, Naderi and Ruiz (2014) presented a scatter search (SS) method for the DPFSP. This SS was shown to outperform existing methods.

To the best of our knowledge, no further literature exists on DAPFSP, so this is the first effort that considers the assembly flowshop problem in a distributed manufacturing setting.

3.3 Mixed Integer Linear Programming model

A mathematical model is an abstract and good approach that uses mathematical language to describe in detail a problem. There are many papers related to the flowshop problem which use MILP modeling; for example, we can cite Stafford et al. (2005); Tseng and Stafford (2008); Ching-Jong and Li-Man (2008) and Naderi and Ruiz (2010), to name just a few.

We first define the model indexes, parameters and variables in Table 3.1, and present the MILP afterwards. The proposed MILP model is inspired by the fifth mathematical model that is presented in Naderi and Ruiz (2010) for the DPFSP that was shown to outperform the other models tested in that paper.

Index	Description
k, j	denotes jobs, $k, j = 0, 1, \ldots, n$, where 0 presents a dummy job
i	denotes machines at each factory, $i = 1, \ldots, m$
l, s	denotes products, $l, s = 0, 1, \ldots, t$, where 0 presents a dummy product
M	A sufficiently large positive number, $M = 100000$
Parameters	**Description**
n	number of jobs
m	number of machines
f	number of factories
t	number of products
p_{ij}	processing time of job j on machine i
pp_s	processing time of product s at the assembly stage
G_{js}	Binary parameter equal to 1 if job j belongs to product s, and 0 otherwise
Variable	**Description**
X_{kj}	binary variable equal to 1 if job k is an immediate predecesor of job j
Y_{ls}	binary variable equal to 1 if product l is an immediate predecesor of product s
C_{ij}	completion time of job j on machine i
CA_s	completion time of product s on assembly stage
$C_{\max}$	makespan

Table 3.1: indexes, parameters and variables used in MILP mathematical model.

The objective function of the model is to minimize a makespan: Min $C_{\max}$ and the constraints of the model are:

$$\sum_{k=0, k\neq j}^{n} X_{kj} = 1 \qquad \forall j \tag{3.1}$$

$$\sum_{j=0, k\neq j}^{n} X_{kj} \leq 1 \qquad \forall k \tag{3.2}$$

$$\sum_{j=1}^{n} X_{0j} = f \tag{3.3}$$

$$\sum_{k=1}^{n} X_{k0} = f - 1 \tag{3.4}$$

$$X_{kj} + X_{jk} \leq 1 \qquad \forall j \in \{1, \ldots, n-1\}, j > k \tag{3.5}$$

$$C_{ij} \geq C_{i-1j} + p_{ij} \qquad \forall i, j \tag{3.6}$$

$$C_{ij} \geq C_{ik} + p_{ij} + (X_{kj} - 1) \cdot M \qquad \forall k, j \neq k, i \tag{3.7}$$

$$\sum_{l=0, l\neq s}^{t} Y_{ls} = 1 \qquad \forall s \tag{3.8}$$

$$\sum_{s=1, l\neq s}^{t} Y_{ls} \leq 1 \qquad \forall l \tag{3.9}$$

$$Y_{ls} + Y_{sl} \leq 1 \qquad \forall l \in \{1, \ldots, t-1\}, s > l \tag{3.10}$$

$$CA_s \geq (C_{mj} \cdot G_{js}) + pp_s \qquad \forall j, s \tag{3.11}$$

$$CA_s \geq CA_l + pp_s + (Y_{ls} - 1) \cdot M \qquad \forall l, s \tag{3.12}$$

$$C_{\max} \geq CA_s \qquad \forall s \tag{3.13}$$

$$X_{kj} \in \{0, 1\} \qquad \forall k, j, k \neq j \tag{3.14}$$

$$Y_{ls} \in \{0, 1\} \qquad \forall l, s, l \neq s \tag{3.15}$$

$$C_{ij} \geq 0 \qquad \forall i, j \tag{3.16}$$

$$CA_s \geq 0 \qquad \forall s \tag{3.17}$$

Note that $C_{0j} = CA_0 = 0, \forall j$. Constraint set (3.1) controls and ensures that each job must have exactly one predecessor. Constraint set (3.2) indicates that each job has one succeeding job at the most. Constraint set (3.3) enforces that dummy job 0 has to have f predecessor in the final sequence. Constraint set (3.4) also enforces that dummy job 0 must be a successor $f - 1$ times (there is no dummy job at the end of the sequence). Constraint set (3.5) controls and ensures that a job cannot be both a predecessor and successor of another job at the same time. Constraint set (3.6) enforces the processing of job j in machine i when the processing at machine $i - 1$ is completed. Constraint set (3.7) determines that if job j is placed immediately after job

k, its processing at machine i cannot start before the processing of job k in machine i finishes. Constraints (3.8) and (3.9) force that each product should have one predecessor and at most one succeeding product in the assembly factory, respectively, constraint (3.10) controls that a product cannot be both a predecessor and a successor of another product at the same time. Constraint (3.11) implies that each product h cannot begin its assembly before all the jobs in its assembly program are completed in the last machine m. Constraint set (3.12) determines that if product s is placed immediately after product l, its processing on assembly machine cannot start before the processing of product l in assembly machine finishes. Constraint (3.13) defines the makespan, while constraints (3.14)-(3.17) define the domain of the decision variables.

The significant point of this model is that there is no index for factories. Sequence-based variables are hence used with a set of f dummy jobs. These dummy jobs divide all the jobs into subsequences and assign them to each factory (i.e., all jobs placed between the first dummy job and the second dummy job belong to the first factory, and so on). For example, if one of the possible solutions for a problem with $n = 8$ and $f = 3$ is $X_{0,2} = X_{2,3} = X_{3,5} = X_{5,0} = X_{0,6} = X_{6,1} = X_{1,4} = X_{4,0} = X_{0,7} = X_{7,8} = 1$, then the sequence is $\{0, 2, 3, 5, 0, 6, 1, 4, 0, 7, 8\}$, where partial job sequences $\{2, 3, 5\}$, {6, 1, 4} and {7, 8} are assigned to factories 1, 2 and 3, respectively.

3.4 Heuristic methods

As mentioned in the paper of Naderi and Ruiz (2010), the DPFSP is an NP-Complete problem (if $n > f$); accordingly, the DAPFSP with an additional assembly stage as a further stage is certainly a NP-Complete problem. Therefore, it is necessary to develop a heuristic approach to solve large-sized problems. In order to solve instances of realistic size in this problem, three constructive simple heuristics are proposed.

For the assignment of jobs to factories, the two rules, of Naderi and Ruiz

(2010) are used.

1. Assign job j to the factory which has the lowest current $C_{\max}$, (NR$_1$).
2. Assign job j to the factory which has the lowest $C_{\max}$ after including job j, (NR$_2$).

Using these two factory allocation rules, three heuristics are presented to schedule jobs.

3.4.1 Heuristic 1

We first introduce some necessary notation. An example with $n = 9$, $m = 2$, $f = 2$ and $t = 3$, this is, 9 jobs, 2 factories with a flowshop of two machines each and three products to assemble, is employed to explain expressions and heuristics in detail. Table 3.2 shows the processing times of the jobs and assembly processing times of products. The products' assembly programs are: $N_1 = \{3, 4, 6\}$, $N_2 = \{1, 2, 8, 9\}$ and $N_3 = \{5, 7\}$. π represents a *product sequence*, e.g., $\pi : \{1, 3, 2\}$ is a possible product sequence for the given example. As mentioned before, each product h is made up of $|N_h|$ jobs and π_h is the *partial job sequence* of product h, e.g., $\pi_1 : \{6, 4, 3\}$, $\pi_2 : \{1, 9, 8, 2\}$ and $\pi_3 : \{7, 5\}$. A *complete job sequence*, π_T, is constructed by putting together all partial job sequences, following the product sequence π, e.g., $\pi_T : \{6, 4, 3, 7, 5, 1, 9, 8, 2\}$.

The shortest processing time (SPT) is a well-known dispatching rule for the PFSP. In the SPT, the job with the shortest processing time is processed first. This rule tends to reduce the work-in-process inventory, the average throughput time, and average job lateness (Vollmann et al., 2005). Hence the SPT is used to determine the product sequence in the assembly machine.

Heuristic 1 begins by applying the SPT rule for the assembly operation times to obtain π. A heuristic which is based on Framinan and Leisten (2003) heuristic (FL) is applied on the jobs that belong to a given product, to obtain a good partial job sequence for each product. The heuristic evaluates the completion times of the jobs that belong to product h. Set R_h is made by sorting jobs in ascending order of completion times. The first two jobs of R_h are selected and inserted into S_h. When there are only two jobs in S_h, all pairwise exchanges are checked and S_h is updated with the one that results in the best makespan. The next step is removing the third job in R_h and inserting it in all possible positions of S_h. The sequence with the best makespan will be selected. All possible sequences by carrying out pairwise exchanges between jobs are evaluated again. The process continues until all jobs have been considered. S_h is the partial job sequence for product h, (π_h). π_T is constructed by putting together all π_h and jobs are assigned to factories from π_T by using NR$_1$ or NR$_2$, which respectively result in the H$_{11}$ or H$_{12}$ heuristics.

Pseudocode 1 explains heuristics H$_{11}$ and H$_{12}$ in detail:

Pseudocode 1 Outline of the H$_{11}$/H$_{12}$ heuristic.

```
- Obtain product sequence $\pi$ after applying the SPT rule on product assembly processing times, $\pi = \{\pi(1), \pi(2), \ldots, \pi(t)\}$; ($\pi(1)$: The first product in product sequence)
- Determine partial job sequence for all products using the proposed algorithm based on FL heuristic ($\pi_h$: partial job sequence for product $h$)
- Construct complete job sequence ($\pi_T$) by putting together all partial job sequences ($\pi_h$), following the product sequence, $\pi$
- Assign all jobs in $\pi_T$ to factories using $NR_1$ to make H$_{11}$ and using $NR_2$ to make H$_{12}$
```

Let us now apply proposed heuristics to the example. $\pi : \{1, 3, 2\}$ is the product sequence obtained after applying the SPT rule to the assembly processing times of the products. The next step is to find a good partial job sequence for each product. As mentioned before, each product has a defined assembly program that includes a defined set of $|N_h|$ jobs. Completion time

	Jobs								
Machines	1	2	3	4	5	6	7	8	9
M_1	1	5	7	9	9	3	8	4	2
M_2	3	8	5	7	3	4	1	3	5
	Product 1			Product 2			Product 3		
M_A	6			19			12		

Table 3.2: Processing times of the jobs and assembly processing times of the products for the example.

for each job at the production stage is the summation of each job processing times on all machines, $\sum_{i=1}^{m} p_{ij}$. Therefore, completion times for set of jobs of the product 1, $N_1 = \{3,4,6\}$ are $C_{23} = 12$, $C_{24} = 16$, $C_{26} = 7$. Set R_1 is obtained by arranging jobs in an increasing completion time order; $R_1 = \{6,3,4\}$. The first two jobs of R_1 are selected and included into S_1. All possible sequences resulting from pairwise exchanges of the first two jobs in S_1 are calculated: {6, 3} and {3, 6} which result in makespans values of 15 and 16, respectively. The sequence with the minimum makespan is $S_1 : \{6,3\}$. The third job in R_1, (4) is inserted into all possible positions of S_1. The obtained partial job sequences are: $\{4,6,3\}$, $\{6,4,3\}$ and $\{6,3,4\}$ and their makespans in the production stage are: 25, 24, 26, respectively. As a result, the second is the best position for job 4 and S_1 is updated to $\{6,4,3\}$. In the next step, general pairwise exchanges are carried out on the updated S_1; hence, the partial job sequences are: $\{4,6,3\}$, $\{6,3,4\}$ and $\{3,4,6\}$ and, subsequently, their makespans in the production stage are, 25, 26, 27, respectively. If a better makespan is obtained, then S_1 is updated. This process continues until all jobs have been inserted into S_1. π_1 is the final updated S_1, which is equal to $\{6,4,3\}$. By following the same method, the partial job sequences for the other products are: $\pi_2 = \{1,9,8,2\}$ and $\pi_3 = \{5,7\}$ with partial makespans of 20 and 18, respectively. Hence π_T is $\{6,4,3,5,7,1,9,8,2\}$. The final step is to assign jobs in π_T to factories by using NR_1/NR_2 to obtain H_{11}/H_{12}. $C_{\max}$

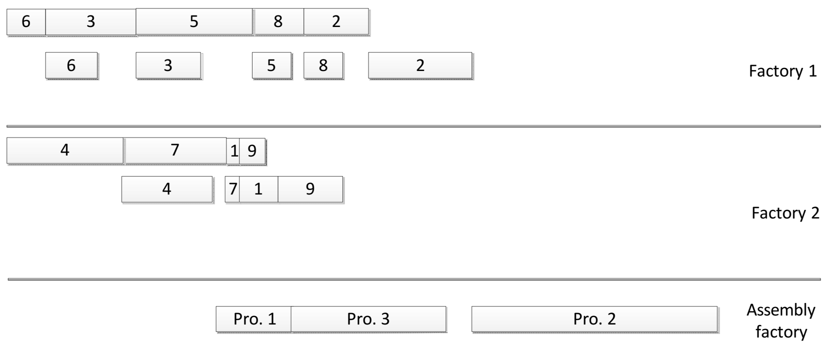


Figure 3.1: Gantt chart of H_{11} for the example.

of H_{11} and H_{12} are 55 and 53, respectively. The Gantt chart of the considered example after applying H_{11} is shown in Figure 3.1.

3.4.2 Heuristic 2

The idea of the second heuristic is to give priority to products whose jobs are completed in the production stage sooner. This concept is noted as the earliest start time to assemble product h, E_h. The procedure that is used in H_{11} and H_{12} to find partial job sequences of products (π_h) also is used in heuristic 2. E_h, is calculated by using NR_1 or NR_2 to assign jobs in each partial job sequence to factories. π is built by sorting E_h in ascending order. A detailed explanation is shown in Pseudocode 2.

The last example data is also used to clarify the second proposed heuristic. E_h is calculated by applying job assignment rules (NR_1 for the H_{21} and NR_2 for the H_{22}) for the partial job sequence of product h. Therefore, the earliest start times for assembling products by considering NR_2 are $E_1 = 15$, $E_2 = 15$ and $E_3 = 12$. The product sequence π is obtained by sorting E_h in ascending order, π :$\{3, 2, 1\}$. As a result, the complete job sequence, π_T, will be: $\{5, 7, 1, 9, 8, 2, 6, 4, 3\}$. The final results of $C_{\max}$ for H_{21} and H_{22} are equal to 51 and 50, respectively.

Pseudocode 2 Outline of the H_{21}/H_{22} heuristic.

```
- Determine partial job sequences of products using proposed algorithm based on FL heuristic ($\pi_h$: partial job sequence for product $h$)
- Calculate the earliest start time to assemble each product $h$, $E_h$, using $NR_1$ and $NR_2$ to assign jobs of the partial job sequences, respectively for $H_{21}$ and $H_{22}$
- Sort $E_h$ in ascending order for all the products to obtain product sequence, $\pi$: $\{\pi(1), \pi(2), \ldots, \pi(t)\}$
- Construct complete job sequence ($\pi_T$) by putting together all partial job sequences ($\pi_h$), following the product sequence, $\pi$
- Assign all jobs of $\pi_T$ to factories using $NR_1$ to make $H_{21}$ and using $NR_2$ to make $H_{22}$
```

3.4.3 Heuristic 3

The third proposed heuristic is similar to the second one. The difference is in the construction of the partial job sequences of each product (π_h). While heuristic 2 uses a heuristic based on FL, heuristic 3 employs the more simple SPT rule. Our intention is to test if a simpler constructive heuristic gives similar results.

Table 3.3 shows the C_{mj} of the jobs, the partial job sequence for each product, after applying the SPT rule and E_h of product h in the columns for the example.

Product sequence π is $\{3, 2, 1\}$ after sorting E_h in ascending order. The complete sequence π_T after putting together the partial jobs sequences of each product is: $\{7, 5, 1, 9, 8, 2, 6, 3, 4\}$. After applying NR_1 to this sequence we obtain a $C_{\max}$ of 51. The $C_{\max}$ for NR_2 is 50.

3.5 Variable Neighborhood Descent (VND)

We now present a Variable Neighborhood Descent (VND) method (Hansen and Mladenović, 2001). VND is an enhanced local improvement strategy based on

Product (h)	Job (j)	C_{2j}	π_h	E_h
1	3	12	6, 3, 4	19
	4	16		
	6	7		
2	1	4	1, 9, 8, 2	15
	2	13		
	8	7		
	9	7		
3	5	12	7, 5	12
	7	9		

Table 3.3: Job completion times on the last machine of production stage, products partial job sequence and earliest start time for assembling each product for the example.

the systematic exploration of different neighborhood structures $N_1, \ldots, N_q$. A VND starts with the first structure N_1 by performing a local search until no further improvements are possible. From this local optimum, it continues the local search with neighborhood structure N_2. If an improved solution is found with this structure, the VND goes back to N_1; otherwise, it continues with N_3, and so forth. If the last structure N_q has been applied and no further improvements are possible, the solution represents a local optimum with respect to all neighborhood structures and the VND terminates.

3.5.1 Solution representation and VND initialization

In order to represent a solution, a complete sequence of all jobs π_T is considered, like in the PFSP. We limit the representation so that all jobs from a product are never separated. The jobs in the complete sequence are assigned to factories using NR$_1$ or NR$_2$. An example of a solution representation can be: $\{6, 4, 3, 1, 9, 8, 2, 5, 7\}$ which is a equal to product sequence of $\{1, 2, 3\}$ with respect to the last example.

The VND approach needs an initial solution. Although a random solution can be used as an initial solution, it is better to use heuristics (Ruiz and Stützle, 2007; Naderi and Ruiz, 2010; Vallada and Ruiz, 2010). Our approach uses the six proposed constructive heuristics to obtain the initial solution. Later we will test six VND versions, each one starting from the result of each heuristic.

3.5.2 Neighborhoods and acceptance criterion

Our proposed VND heuristic employs two neighborhood structures, and both are applied to the complete sequence π_T.

The first is referred to as LS_P and is a product local search. It attempts to improve the objective function by examining different product sequences. LS_P works as follows: (1) It provides a list of product sequences by removing a single product from π and inserting it in all the possible $t-1$ positions of current π; (2) It evaluates the list of obtained product sequences by converting them into π_T and assigning the jobs of π_T to factories via NR$_1$ or NR$_2$; (3) If one of the obtained π in the list has a better $C_{\max}$, then π is updated to the better product sequence and all the products are reinserted again (a local search until a local optimum), otherwise the search continues with the next product.

The second neighborhood is LS_J, tries to find different partial job sequences for each product to improve the objective function. LS_J works as follows: (1) LS_J starts with the first product h, then the local search starts by removing the first job of π_h and inserting it in all the possible $|N_h|-1$ positions of π_h; (2) Evaluate π_T with all the newly obtained partial job sequences for product h; (3) If a better objective function is obtained, then π_h is updated and all jobs in π_h are reinserted again until a local optimum is found. Otherwise, the search continues with the next job in π_h; (4) LS_J will continue with the next product until all products have been considered.

Pseudocodes 3 and 4 show the product and the job local search, respectively.

Pseudocode 3 Product Local Search, LS$_P$.

```
$l = 1$
while $l \leq t$ do
    - Remove product $a$ which is placed in position $l$ of $\pi$
    - Insert $a$ into all $t - 1$ possible positions of $\pi$
    - Evaluate all obtained $\pi$ by converting them into $\pi_T$
    if a better $C_{\max}$ is obtained then
        - update $\pi$
    else
        $l = l + 1$
    end if
end while
```

Pseudocode 4 Job Local Search, LS$_J$.

```
$h = 1$
while $h \leq t$ do
    $j = 1$
    while $j \leq N_h$ do
        - Remove job $b$ which is placed at position $j$ of $\pi_h$
        - Insert $b$ into all $|N_h| - 1$ possible positions of current $\pi_h$
        - using the new $\pi_h$, convert it to $\pi_T$
        if a better $C_{\max}$ is obtained then
            - Select the partial job sequence with the best result as the new $\pi_h$
        else
            $j = j + 1$
        end if
    end while
    $h = h + 1$
end while
```

3.6 Computational evaluation

Two complete sets of instances have been generated to test the MILP model and the proposed heuristics. Due to the complexity of the problem, and given the number of different characteristics considered, four instance factors and three test factors are combined at the levels provided in Table 3.4 for small instances. The test factors are: two commercial solver packages (*Solver*) are used as solving tools, the number of CPU threads (*Thread*), where we have tested 1 thread (serial computing) and 2 threads (parallel computing) and a time limitation *TimeLimit* for the stopping criterion. The heuristics are also tested in a set of larger instances, which differ in the factors as listed in Table 3.5.

Instance factor	Symbol	Number of levels	Values
Number of jobs	n	5	8, 12, 16, 20, 24
Number of machines	m	4	2, 3, 4, 5
Number of factories	f	3	2, 3, 4
Number of products	t	3	2, 3, 4
Test factor	Symbol	Number of levels	Values
Solver	$Solver$	2	CPLEX 12.3, GUROBI 4.6.1
Thread	$Thread$	2	Serial computing (1), Parallel computing (2)
Time limitation	$TimeLimit$	2	$900s, 3600s$

Table 3.4: Instance and test factors for the small instances.

Instance factor	Symbol	Number of levels	Values
Number of jobs	n	3	100, 200, 500
Number of machines	m	3	5, 10, 20
Number of factories	f	3	4, 6, 8
Number of products	t	3	30, 40, 50

Table 3.5: Instance factors for the large instances.

Processing times in the production stage are fixed to $U[1, 99]$ as it is usual in the scheduling literature. The assembly processing times depend on the number of jobs assigned to each product h as $U[1 \times |N_h|, 99 \times |N_h|]$. The total number of combinations in the small and large instances are $5 \times 4 \times 3^2 = 180$ and $3^4 = 81$, respectively. There are 5 replications per combination for small instances and 10 replications for every large combination. Therefore, the total number of instances is 900 and 810, respectively. All the instances are available at `http://soa.iti.es`.

3.6.1 MILP model evaluation

A linear programming model has been constructed for each small instance. It is solved with all the combinations of the test factors, using CPLEX 12.3 and GUROBI 4.6.1 solvers, serial and parallel computing and two time limits ($900s$ and $3600s$). All the tests are carried out in a high performance computing cluster with 30 blades, each one containing 16 GBytes of RAM memory and two Intel XEON E5420 processors running at 2.5 GHz. Note that each processor has 4 physical computing cores (8 per blade). The 30 blade servers are used only to divide the workload and experimentations. Experiments are carried out in virtualized Windows XP machines, each with one virtualized processor with two cores and 2 GB of RAM memory.

A categorical variable named "response type" with two values, 0 and 1, is reported. Value 0 means that an optimum solution is found in the given time with $C_{\max}$ value as a result, and 1 means that in 900s or 3600s, a feasible integer solution is found and reported, but it has not been proven to be optimal. Moreover, the gap between this solution and the best MILP bound is also reported. In the CPU time allowed, the LP model with all 900 small instances is able to find 516 optimum solutions (57.33 %). Table 3.6 summarizes the results, which are categorized by factors of solver, threads and time limit. The comparison criteria are: the percentage of optimum solutions found ($\%opt$), the average gap as a percentage for the cases in which the optimum solution is not found ($GAP\%$) and the average time required in seconds. Later we will carry out statistical testing to ascertain the significance of the observed differences.

It is clear that GUROBI is able to find more optimal solutions than CPLEX, and its average gap and average CPU time consumption are smaller than CPLEX. Overall, time limit of 3600 seconds and parallel computing (2 threads) results in a larger number of optimal solutions, in comparison with time limit of 900 seconds and serial computing (1 thread). CPLEX with parallel computing (2 threads) results in a greater average gap in comparison with serial computing, but this trend is reversed with GUROBI. Among all the eight combinations of test factors, GUROBI with two threads and 3600 seconds time limitation finds more optimum solutions than the others.

Solver	Time Limit	$900s$		$3600s$	
	Thread	1	2	1	2
CPLEX	% opt	59.44	61.22	63.11	61.89
	GAP%	29.62	30.77	32.23	36.46
	Av Time (s)	390.41	380.69	1426.53	1441.80
GUROBI	% opt	66.89	68.33	70.78	73.00
	GAP%	2.19	2.04	1.81	1.70
	Av Time (s)	328.15	315.57	1152.36	1089.00

Table 3.6: Performance results for solvers, threads and time limit for the small instances.

Automatic Interaction Detection (AID) is an advanced statistical technique for multivariate analysis, which was developed by Morgan and Sonquist (1963). It seeks to find explanatory variables and combinations of these variables which are important for lowering variance in the dependent variables. AID is a stepwise procedure that subdivides experimental data according to one factor through a series of dichotomous splits into a number of mutually exclusive subgroups. The initial AID was improved by Kass (1980) by including statistical significance testing in the partition process and by allowing multiway splits of data resulting in the so-called Chi-squared Automatic Interaction Detection (CHAID). A modification to the basic CHAID algorithm, called an exhaustive CHAID, introduced by Biggs et al. (1991), performs a more thorough merging and testing of factor variables.

An exhaustive CHAID is used to draw a decision tree to analyze the effect and interactions of the factors for the averages observed in Table 3.6. AID techniques are used in different areas like market research, psychology, education, scheduling, etc. Recently, CHAID was employed by Ruiz et al. (2008) to analyze a complex non distributed scheduling problem MILP model. Also, Ruiz and Andrés-Romano (2011) employed CHAID to analyse a MILP in a problem with unrelated parallel machines with resource-assignable sequence-dependent setup times. Naderi and Ruiz (2010) also used CHAID to analyze several models for the distributed permutation flowshop scheduling problem.

The exhaustive CHAID method is used to analyze the MILP results, which were previously presented. The factors, either serial computing or parallel computing (*Threads*), *solver*, n, m, f and t, are controlled. We introduce all the data of both stopping CPU time criteria so the factor time is controlled as well. The response variable is the type of solution reached by CPLEX and GUROBI with two possible values (0 and 1). We use the PASW statistics version 18 software and set a high confidence level for splitting of 99.9%, as well as a Bonferroni adjustment for the multi-way splits, which compensates the statistical bias in multi-way paired tests.

In Figure 3.2, the root node contains the total percentage of the cases were instances were solved optimally (type 0) and the total number of cases. The most significant factor is the number of jobs or n, and the next level is divided into one node for each possible n value. The p-value obtained for this split comes very close to 0 and the result of the χ^2 statistic is very high, meaning that the split is done with a very high level of confidence; i.e., n is the most influential factor on the response variable with a very statistically significant effect.

Among the resulting five nodes, as the n value increases, the number of cases for which an optimal solution is found decreases. As a matter of fact, for n = 20 and 24, only 35.6% of the instances are optimally solved. After this first multi-way split, nodes are split into the number of factories factor, except for n =8. It is logical that when there is a larger amount of factories, jobs have more options for allocations, and the completion time of jobs also shortens. Hence, the earliest possible time to start product assembly also shortens, and the possibility of finding a better solution increases. The number of products t is the third next important factor, except for node $n = 12$ / $f = 3$, where number of machines is a significant factor. No further statistically significant divisions are found and the stopping criterion for branching is met for nodes $n = 12$ / $f = 4$ and $n = 24$ / $f = 2$. The number of products factor shows the same trend as the second important factor (number of factories); that is, a higher percentage of optimal solutions is found when there is a larger number of products. If the number of jobs is constant and the number of products increases, fewer jobs will be dedicated to each product on average, so finding a better partial job sequence for each product is easier.

As seen, apart from a few isolated cases, the effect of type of solver, one thread (serial computing) and two threads (parallel computing) and time limit ($900s$ and $3600s$) are not statistically significant.

3.6.2 Heuristics evaluation

The twelve proposed methods (H$_{11}$, H$_{12}$, H$_{21}$, H$_{22}$, H$_{31}$, H$_{32}$, VND$_{H_{11}}$, VND$_{H_{12}}$, VND$_{H_{21}}$, VND$_{H_{22}}$, VND$_{H_{31}}$ and VND$_{H_{32}}$) are now tested. As the proposed heuristics are not expected to find an optimal solution, the Relative percentage deviation (RPD), is measured for comparisons. We measure RPD as follows: using the optimal solution or the best known solution, which is found through all heuristics and the MILP model (OPT_{best}) and ALG_{SOL}, which reports the makespan obtained by a given algorithm for a given instance:

$$RPD = \frac{ALG_{SOL} - OPT_{best}}{OPT_{best}} \times 100$$

Table 3.7 provides the summarized results of the MILP and the average algorithm deviations from the best known solution for the small instances. They are categorized by n and f.

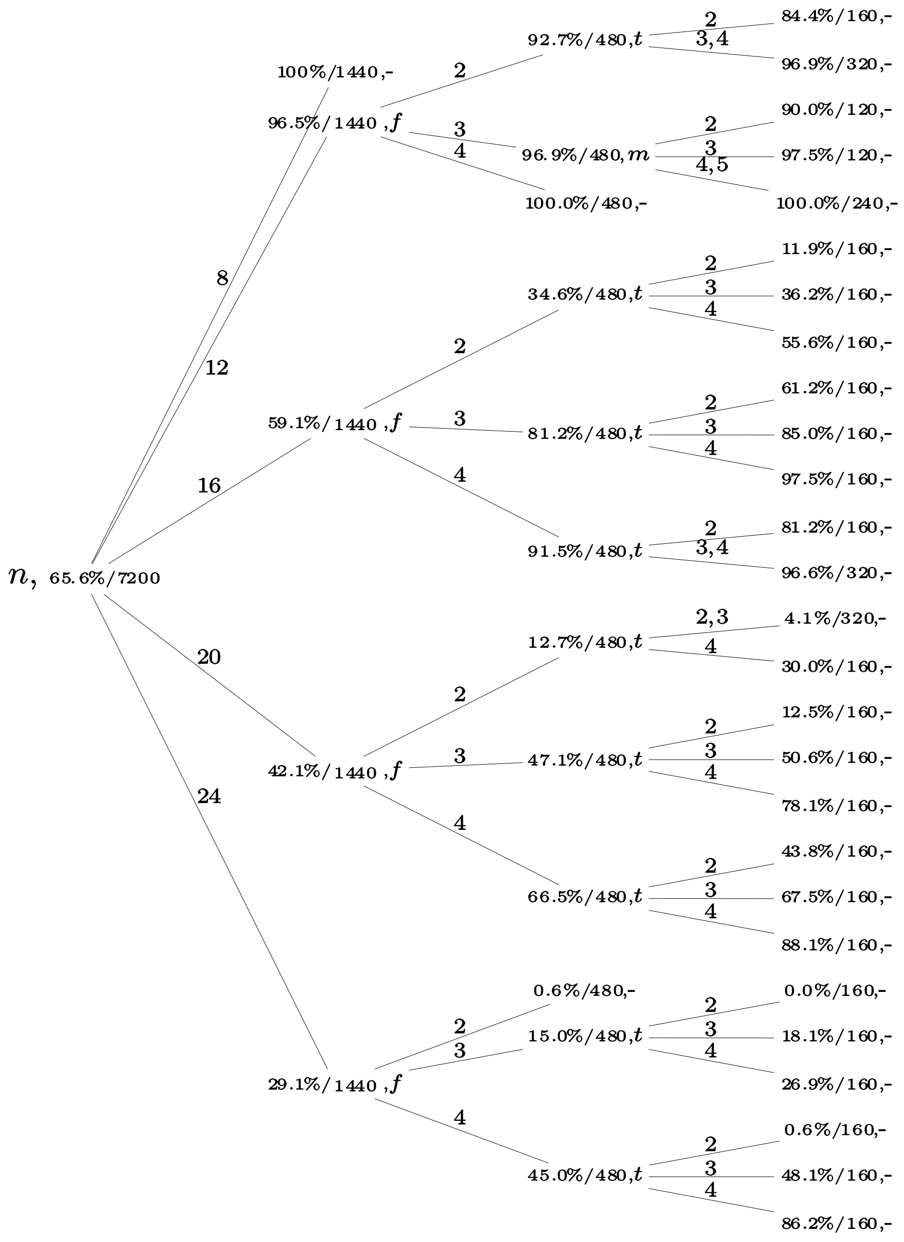


Figure 3.2: Decision tree for the MILP model evaluation.

		Algorithms											
$f \times n$	MILP	H_{11}	H_{12}	H_{21}	H_{22}	H_{31}	H_{32}	$VND_{H_{11}}$	$VND_{H_{12}}$	$VND_{H_{21}}$	$VND_{H_{22}}$	$VND_{H_{31}}$	$VND_{H_{32}}$
2×8	0.00	14.62	13.61	6.91	5.99	13.55	12.17	1.00	0.76	1.00	0.76	1.02	0.78
2×12	0.02	13.70	12.78	5.74	5.17	11.58	11.05	0.93	0.87	0.93	0.87	0.93	0.87
2×16	0.45	12.52	11.40	5.77	5.10	10.00	9.16	0.73	0.55	0.72	0.53	1.09	0.53
2×20	1.55	10.23	9.59	4.55	3.78	8.96	8.46	0.53	0.36	0.51	0.37	0.57	0.37
2×24	3.42	8.71	8.34	5.00	4.74	7.54	7.15	0.54	0.21	0.54	0.21	0.54	0.21
3×8	0.00	11.35	9.96	4.57	3.15	8.92	7.79	1.09	0.70	1.15	0.76	1.15	0.76
3×12	0.02	9.96	9.13	3.03	2.55	8.72	7.50	0.44	0.28	0.44	0.28	0.44	0.28
3×16	0.05	10.10	9.16	3.77	3.14	9.59	8.73	0.86	0.56	0.91	0.56	0.91	0.56
3×20	0.40	9.86	8.93	2.72	2.19	8.53	7.84	0.43	0.43	0.43	0.43	0.43	0.43
3×24	1.16	7.77	6.48	3.11	2.52	7.24	6.32	0.64	0.33	0.64	0.33	0.64	0.33
4×8	0.00	9.03	8.01	2.16	1.25	6.41	5.25	1.08	0.63	0.99	0.63	0.99	0.63
4×12	0.00	5.63	4.53	1.82	1.38	4.58	3.58	0.74	0.47	0.74	0.47	0.74	0.56
4×16	0.03	7.21	6.34	2.86	2.27	6.14	5.18	0.59	0.28	0.59	0.28	0.59	0.28
4×20	0.21	6.80	6.00	2.96	2.61	5.66	5.04	1.10	0.63	1.10	0.63	1.10	0.63
4×24	0.40	5.14	4.43	2.02	1.60	4.87	4.19	0.57	0.26	0.57	0.26	0.57	0.26
Average	0.51	9.51	8.58	3.80	3.16	8.15	7.29	0.75	0.49	0.75	0.49	0.78	0.50

Table 3.7: Relative Percentage Deviation (RPD) of MILP and proposed algorithms over the best known solution for the small instances.

As we can see in Table 3.7, it is clear that the mathematical model is unable to find an optimum or best solution for all the small instances considered. By increasing the number of jobs (n) and by decreasing the number of factories (f), the problem becomes harder for the MILP to solve. All VND algorithms perform better than the constructive algorithms. NR_2 works better than the first one as a rule to assign jobs to factories. In order to know if the differences observed in Table 3.7 are statistically significant, a multifactor ANOVA of the results of the VND algorithms has to be done. The average RPD value for all the simple constructive heuristics is 6.75%, and this amount lowers to 0.63% for the VND methods. The RPD factor difference between simple constructive heuristics and VND heuristics is very high. For this reason, we separated the statistical analysis in two ANOVAs: one for the simple heuristics and the other one for the VND methods. As explained before, there are 900 small instances, and each ANOVA considers six simple constructive heuristics or six VND methods with $6 \times 900 = 5400$ data.

As with all parametric analyses, ANOVA requires some assumptions to be met. These are normality, homocedasticity and independence of residuals. While a slightly strong tailed normal distribution of the residuals is observed, residuals are clearly homoscedastic and independent, and according to the recent results of Basso et al. (2007) and Rasch and Guiard (2004), this is not a major problem. The response factor is again the RPD and the controlled factors are n, m, f, t and *algorithms*. All the controlled factors in the ANOVA analysis, except m and t in six simple constructive heuristics, and except f factor in six VND methods result in strong statistically significant differences in the RPD response variable, with p-values coming very close to zero. The results are not shown here due to reasons of space. In order to identify the best algorithm, the means plot and Tukey's Honest Significant Difference (HSD) intervals (99% confidence) for the six simple constructive heuristics and VND methods are shown in Figures 3.3 and 3.4, respectively.

As it is clear in Figure 3.3, the second heuristic performs better in comparison with the other simple constructive heuristics and there is no significant differences between the rules used to assign jobs to factories. However, it is obvious in Figure 3.4 that the rules for allocating jobs to factories are important, and NR_2 is statistically different from NR_1. It is clear that the VND algorithm almost improves all the initial solutions equally and that the kind of initial solution to start the VND is not important for algorithms with the same job assignment rule. No significant differences between the three VND considered algorithms using NR_2 is found.

The CPU times to solve small instances with the considered algorithms are negligible; for example, the $\mathrm{VND}_{H_{32}}$ algorithm with 0.004693 seconds, has the largest average consumed CPU time for the small instances.

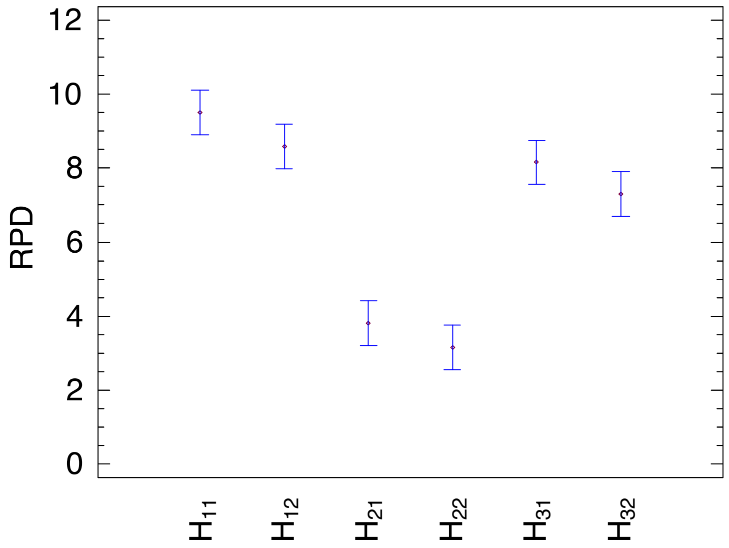


Figure 3.3: Means plot and 99% confidence level Tukey's HSD intervals for simple constructive heuristic methods and small instances.

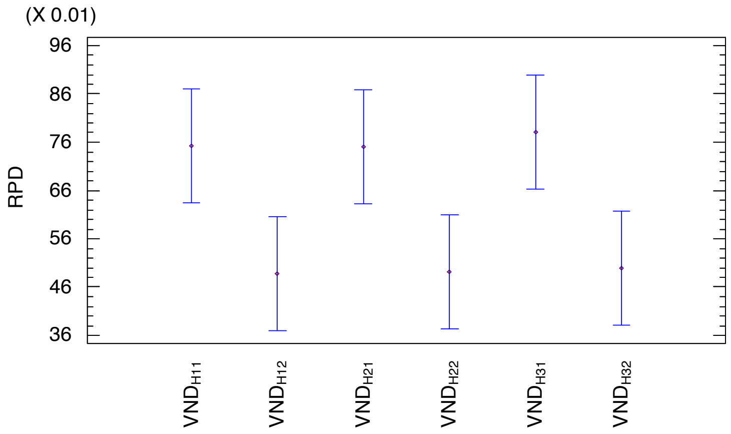


Figure 3.4: Means plot and 99% confidence level Tukey's HSD intervals for VND methods and small instances.

3.6.3 Heuristics evaluation on large instances

In this case, for calculating the RPD, the best solution (OPT_{best}) is the best solution found among all twelve algorithms because, in large instances, good MILP bounds are not known. A summarized result of the average RPD, considering number of factories, number of products and number of jobs, is shown in Table 3.8. Algorithms can be categorized into two groups: VND algorithms, H_{21} and H_{22}, in one group, which perform better, and the rest in another group. On the other hand, algorithms with NR_2 work better than those with NR_1.

			Algorithms											
			H_{11}	H_{12}	H_{21}	H_{22}	H_{31}	H_{32}	VND$_{H_{11}}$	VND$_{H_{12}}$	VND$_{H_{21}}$	VND$_{H_{22}}$	VND$_{H_{31}}$	VND$_{H_{32}}$
Relative Percentage Deviation	Factories (f)	4	5.57	5.09	0.32	0.19	2.96	2.56	0.06	0.03	0.05	0.01	0.05	0.01
		6	3.77	3.29	0.11	0.06	1.64	1.31	0.03	0.01	0.02	0.00	0.02	0.00
		8	3.09	2.66	0.04	0.02	1.21	0.93	0.02	0.00	0.01	0.00	0.01	0.00
	Products (t)	30	3.78	3.34	0.21	0.11	2.23	1.86	0.03	0.01	0.04	0.01	0.04	0.01
		40	4.30	3.85	0.15	0.10	1.94	1.62	0.04	0.02	0.02	0.01	0.02	0.01
		50	4.36	3.85	0.11	0.05	1.65	1.32	0.04	0.01	0.02	0.00	0.02	0.00
	Jobs (n)	100	6.30	5.61	0.17	0.08	2.02	1.58	0.05	0.02	0.03	0.01	0.03	0.01
		200	3.76	3.28	0.15	0.07	1.92	1.55	0.03	0.01	0.02	0.00	0.02	0.00
		500	2.37	2.16	0.14	0.10	1.87	1.67	0.03	0.01	0.03	0.01	0.03	0.01
		Aver	4.14	3.68	0.16	0.09	1.94	1.60	0.04	0.01	0.03	0.01	0.03	0.01
CPU time (sec.)	Factories (f)	4	0.01	0.01	0.01	0.01	0.01	0.01	4.39	6.79	2.90	7.67	2.55	42.87
		6	0.01	0.01	0.01	0.01	0.01	0.01	3.49	7.73	2.85	8.94	1.95	6.11
		8	0.01	0.01	0.01	0.01	0.01	0.01	3.26	9.56	1.86	10.21	1.83	20.64
	Products (t)	30	0.01	0.01	0.02	0.02	0.01	0.01	3.64	8.05	3.14	11.00	2.70	45.20
		40	0.01	0.01	0.01	0.01	0.01	0.01	3.59	7.12	2.45	8.05	1.96	5.54
		50	0.01	0.01	0.01	0.01	0.01	0.01	3.91	8.91	2.02	7.77	1.66	18.88
	Jobs (n)	100	0.00	0.00	0.00	0.00	0.00	0.00	1.09	2.84	0.27	0.72	0.24	0.43
		200	0.00	0.00	0.00	0.00	0.00	0.00	2.02	3.85	0.58	2.22	0.66	1.37
		500	0.03	0.02	0.03	0.04	0.03	0.02	8.03	17.39	6.76	23.88	5.41	67.81
		Aver	0.01	0.01	0.01	0.01	0.01	0.01	3.71	8.03	2.54	8.94	2.11	23.20

Table 3.8: Relative Percentage Deviation (RPD) and CPU times of proposed algorithms for the large instances.

The second group does not report good results if compared to the first one, so it has been eliminated from the statistical analysis. A multifactorial ANOVA has been carried out with only the first group to know if there are any significant differences between results. Figure 3.5 shows a means plot (99% confidence level Tukey's HSD intervals) for the first group of algorithms. It is clear that the algorithms which use NR_2 as a job assignment rule, report better results. Moreover, the type of initial solution for the VND algorithms does not play an important role. Finally, there is no significant difference between the VND algorithms that use the same job allocation rule.

It is obvious that heuristic 2 performs better than heuristic 3 in both small and large instances.

The interaction between algorithms and n has no significant effect on the response variable. An increase in the number of machines always complicates problems, thus there is no interest in showing these interactions. Interaction between algorithms and the number of factories f is interesting. By increasing the number of factories, the problem becomes easier, as it is shown in Figure 3.6.

Neither the number of products nor the number of jobs factors have a significant effect, and only an increase in either makes the problem easier to solve for simple constructive algorithms. However, neither one has a significant effect on the VND algorithms.

In all the results, the RPD of $\mathrm{VND}_{H_{22}}$ is consistently lower than that of the other algorithms. Thus with more samples, it is expected that it will eventually become statistically better than the others. $\mathrm{VND}_{H_{22}}$ is better than $\mathrm{VND}_{H_{21}}$ because NR_2 checks all the factories when assigning a job and finally chooses the best one. It takes longer than NR_1, which just places the job at the first available factory. However, when the number of factories increases, the algorithms that use NR_1 do not report good results.

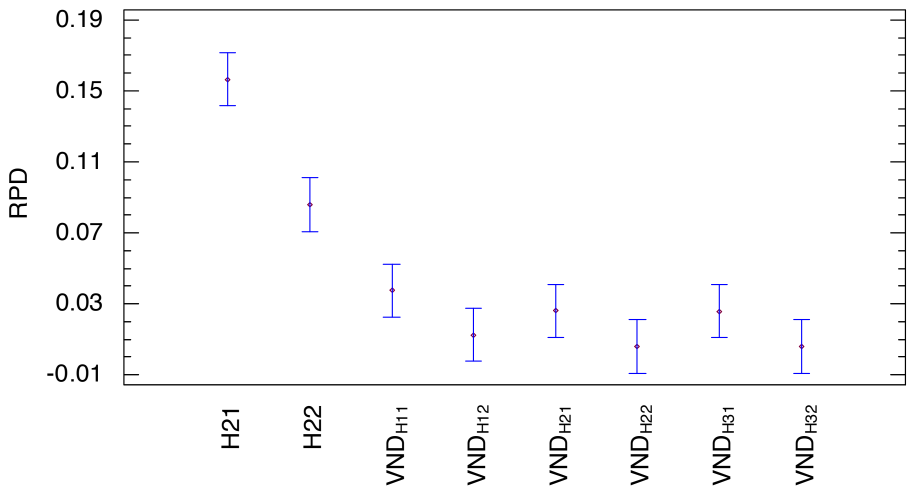


Figure 3.5: Means plot and 99% confidence level Tukey's HSD intervals for algorithms and large instances.

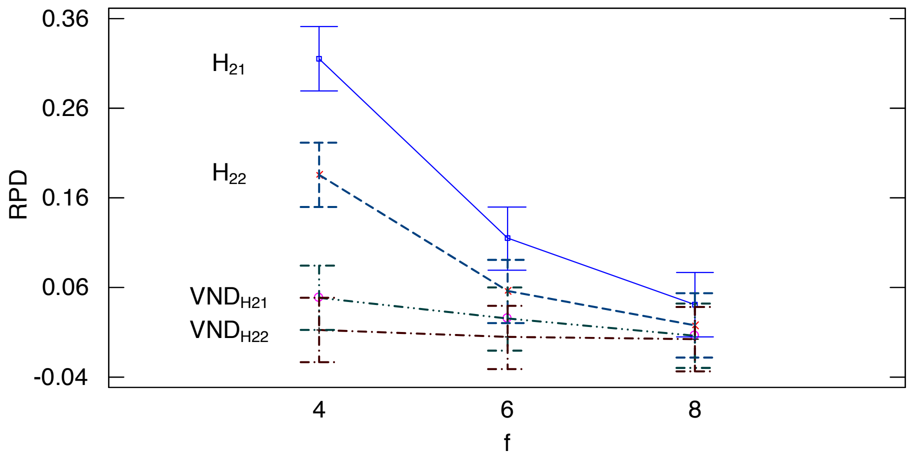


Figure 3.6: Means plot and 99% confidence level Tukey's HSD intervals for interaction between algorithms and number of factories f and large instances.

The algorithms' CPU time consumption is summarized in Table 3.8. Simple constructive algorithms use a very short time in order to solve problems, while, as expected, the VND algorithms use more time compared to simple constructive algorithms. VND$_{H_{32}}$ consumes an average of 23.20 seconds, the longest CPU time consumption compared to other algorithms. As Table 3.8 shows, in the VND$_{H_{32}}$ algorithm, factors $n = 500$, $t = 30$ and $f = 4$ are the most CPU time consuming.

The VND methods try to improve the output of simple constructive algorithms and it is logical that take more time than simple constructive algorithms to solve problems. To compensate, VND algorithms report smaller RPD values than simple constructive algorithms. As Table 3.8 shows, the minimum RPD reported by a simple constructive algorithm is nine times larger than the largest reported RPD by VND algorithms that use NR$_2$.

If the quality of the solution is more important than CPU time consumption, then VND algorithms are the best options. Otherwise, a simple constructive algorithm can be a good choice when only CPU time consumption is more important. However, it is worth waiting a maximum time of almost 24 seconds to obtain a good solution. All the experimental results and the best solutions can be found at `http://soa.iti.es`.

3.7 Conclusions of this chapter

To the best of our knowledge, the results of this chapter are the first attempt to generalize the Distributed Permutation Flowshop Scheduling Problem to the Distributed Assembly Permutation Flowshop Scheduling Problem, where there is more than one production center to process jobs and a single assembly center to make final products from produced jobs. A mathematical model is presented and two solvers are used to solve it. Three constructive algorithms and three VND algorithms are proposed. Computational evaluations were performed with two groups of small and large instances, and ANOVAs were used to analyze results. Results show that the VND algorithms report the best results. On the other hand, simple constructive algorithms consume little CPU time and still produce reasonable solutions.

CHAPTER 4

DISTRIBUTED ASSEMBLY PERMUTATION FLOWSHOP SCHEDULING PROBLEM WITH SEQUENCE DEPENDENT SETUP TIMES

"The contents of this chapter are taken from: Hatami, S., Ruiz, R., and Andrés-Romano, C. (2015). Heuristics and Metaheuristics for the Distributed Assembly Permutation Flowshop Scheduling Problem with Sequence Dependent Setup Times. *International Journal of Production Economics* (Accepted)".

In this chapter, we consider a Distributed Assembly Permutation Flowshop Scheduling Problem with sequence dependent setup times and the objective of makespan minimization. The problem consists of two stages, production and assembly. The first stage comprises f identical factories, where each factory is a flowshop that produces jobs which are later assembled into final products through an identical assembly program in a second assembly stage made by a single machine. Both stages have sequence dependent setup times. This is a realistic and complex problem and therefore, we propose two simple heuristics and two metaheuristics to solve it. A complete calibration and analysis through a Design Of Experiments (DOE) approach is carried out. In the process, important knowledge of the studied problem is obtained as well as some simplifications for the powerful Iterated Greedy methodology which results in a simpler approach with less parameters. Finally, the performance of the proposed methods is compared through extensive computational and statistical experiments.

4.1 Introduction

An assembly production floor typically contains two differentiated stages; a production and an assembly section. In this chapter we study a distributed assembly flowshop with many potential applications. Assembly flowshops have been widely studied recently and constitute a hot topic for research. The scheduling setting considered in this chapter is composed of a production section that is a distributed flowshop problem in itself where jobs are manufactured in a set of machines that are disposed in series. After individual jobs are produced, they are assembled in a single assembly machine to form final products. These production systems are referred to as Assembly Flowshop Scheduling Problems (AFSP) according to Koulamas and Kyparisis (2001). The AFSP applications range from fire engine assembly Lee et al. (1993) to personal computer manufacturing (Potts et al., 1995). As pointed out in Koulamas and Kyparisis (2001), AFSP settings are capable of producing large product varieties by using modular structures at a controlled cost.

We also consider several extensions to the studied problem so as to bring it as close as possible to the reality of production shops. For example, single factories are not common in practice and many companies operate several factories working as distributed production environments (Chan et al., 2005a). Distributed production is key in modern manufacturing (Moon et al., 2002). Additionally, distributed manufacturing leads to high quality production and other benefits such as reduced production costs, decreased management risks and more (Wang (1997); Jia et al. (2003); Kahn et al. (2004); Chan et al.

(2005a), among others). As a first extension we consider several distributed assembly flowshops to reap these benefits.

The second extension considered is the addition of setup times. Unlike processing times, setups are non-productive periods of time in between the production of successive jobs in machines where cleaning, configurations, adjustments and other procedures are carried out. Setups are broadly classified into Sequence Independent Setup Times (SIST) and Sequence Dependent Setup Times (SDST). This last category is more realistic and general and appears when the amount of setup time depends on the job that has been finished by the machine and the job that is to be produced next. Scheduling with setup times is a very important area of research and a large number of review papers have been published, such as Yang and Liao (1999), Allahverdi et al. (1999, 2008) or Cheng et al. (2000).

More precisely, the flowshop problem consists of scheduling a set N of n jobs in a set M of m machines. Jobs have to visit a predetermined machine sequence which is, without loss of generality, $\{1, 2, \ldots, m\}$. The machines are disposed in series and a job is broken down into m tasks, one per machine. The processing time of a given job at a machine is a known, deterministic and non-negative quantity referred to as p_{ij}, $i \in M, j \in N$, which is furthermore usually an integer. The objective is to obtain a sequence of the jobs in the machines so that a criterion is optimized. There are n tasks per machine and any ordering is possible. Therefore, there are $(n!)^m$ possible solutions in this problem. In order to reduce the search space, the most studied variant of this problem is the so called Permutation Flowshop Scheduling Problem or PFSP. In this case, job passing is not allowed and once a production sequence of the jobs is determined for the first machine, it is maintained for all other machines, reducing the search space to $n!$ solutions or sequences. The PFSP comes with some assumptions: A task from a given job can only start at a machine i when the processing of the task of the same job at the previous machine $i - 1$ has finished and also only when machine i is free after processing the previous task in the sequence. No breakdowns are experienced by the machines and they are always available. Each machine can only process one job at a time and each job can only be processed by one machine at the same time. The first task of each job on machine 1 is ready for processing at time 0. There is no preemption, i.e., once a task begins processing in a machine it cannot be stopped until completion. Finally, jobs can wait indefinitely in between machines and an infinite storage of in-process products exists (Baker, 1974). If we define by C_j the time at which job $j \in N$ is completed at the last machine m, the most commonly studied criterion is the minimization of the maximum completion time, commonly referred to as makespan or $C_{\max}$. The PFSP with this criterion has been studied extensively in the scheduling literature. Some reviews are Framinan et al. (2004), Ruiz and Maroto (2005), Hejazi and Saghafian (2005) and Gupta and Stafford (2006).

The extension of the PFSP to distributed manufacturing, referred to as the Distributed Permutation Flowshop Problem (DPFSP) was studied for the first time in Naderi and Ruiz (2010). In this extension, we have a set F of f identical factories. Each factory is a PFSP. Each job has to be first assigned to one of the factories and the problem then consists of solving f PFSPs while minimizing the maximal $C_{\max}$ among the f factories. It is assumed that once a job $j \in N$ is assigned to a factory $f \in F$, it is completed there and no reassignments are possible. The authors of Hatami et al. (2013) recently studied the Distributed Assembly Permutation Flowshop Scheduling Problem or DAPFSP for the first time. In this problem, the first stage is a distributed flowshop and the second stage is a single assembly machine. The authors presented a Mixed Integer Linear Programming Model (MILP), several constructive heuristics and simple local search based Variable Neigborhood Descent (VND) methods. In this chapter we further generalize the DAPFSP with the addition of sequence dependent setup times both in the distributed flowshop production stage as well as in the single machine assembly stage. We improve on the previous VND and also present an effective Iterated Greedy

(IG) method. IG has shown excellent performance in the regular PFSP (Ruiz and Stützle, 2007) and also in the PFSP with SDST (Ruiz and Stützle, 2008) and hence is chosen as a promising approach.

This DAPFSP with sequence dependent setup times (DAPFSP-SDST) is now explained in detail. There is a set T of t (unrelated) products that are manufactured through an assembly of n jobs, each fabricated in the PFSP factories of the production stage. There is a defined assembly program for each product $h \in T$ carried out on a single assembly machine, referred to as M_A. Each product $h \in T$ is assembled from a subset $N_h, N_h \subseteq N$ of jobs that need to be assembled into product h. Therefore, product h consists of $|N_h|$ jobs. Each job belongs to a single assembly program of a given product and therefore we have $\sum_{h=1}^{t} |N_h| = n$. A product h can be assembled at the single machine assembly stage only after all jobs in N_h have been completed in the f distributed factories. The assembly processing time in the single machine assembly stage is referred to as p_h. Furthermore, S_{ijk} denotes the sequence dependent setup time that is needed at machine i of any of the f factories after having processed job j and before processing job k. This setup time is separable from the processing time. There is also an initial setup time. As a result, a $(n + 1 \times n)$ setup time matrix is considered for each production machine. Setup time matrices do not change from factory to factory as factories are assumed to be identical. We also consider sequence dependent setup times in the single machine assembly stage. We denote by SA_{ls} the setup between the assembly of products l and s, $l \neq s, l, s \in T$. Note that an initial setup is also needed to prepare the assembly machine for the assembly of the first product $h \in T$, referred to as SA_{0h}. Again, a $(t + 1 \times t)$ assembly setup time matrix is required. All setups are non-negative integers that are known in advance and deterministic.

The chapter is arranged as follows: Section 4.2 presents a brief literature review on previous and related research. Section 4.3 introduces two simple constructive heuristic methods for the considered problem. Sections 4.4

and 4.5 describe the proposed VND and IG methods, respectively. In section 4.6, the proposed methods are calibrated. Section 4.7 presents a complete computational evaluation of the proposed algorithms. Finally, Section 6.5 concludes the chapter and presents some future research questions.

4.2 Literature review

The DAPFSP is a combination of the assembly (AFSP) and distributed (DPFSP) permutation flowshop problems. Together with the regular flowshop, the literature is extensive. The reader is again referred to the many existing reviews (Framinan et al., 2004; Ruiz and Maroto, 2005; Hejazi and Saghafian, 2005; Gupta and Stafford, 2006).

As regards the AFSP, there is also a significant amount of existing results. In Lee et al. (1993) a three-machine assembly-type flowshop scheduling problem with makespan minimization is presented. Each product consists of two jobs, each to be produced in the first and second machine respectively, where the third machine assembles the two jobs into a product. The authors present a branch-and-bound exact method and an approximate solution procedure. In Potts et al. (1995) m parallel production machines in the first stage are considered. A compact vector summation technique to find approximated solutions with worse-case absolute performance guarantees is applied. In Hariri and Potts (1997) a branch-and-bound algorithm for the same model is developed. A two-stage assembly scheduling problem is considered in Tozkapan et al. (2003). A lower bound and a dominance criterion are developed and incorporated into a branch-and-bound procedure, this time with total weighted flow time minimization as an objective. A heuristic procedure to find an initial upper bound is also proposed. In Al-Anzi and Allahverdi (2006) the same model is studied and metaheuristics such as simulated annealing (SA), tabu search (TS), and hybrid tabu search heuristics to solve the problem are proposed. In Al-Anzi and Allahverdi (2009) a two-stage AFSP is considered and

TS, particle swarm optimization (PSO), and self-adaptive differential evolution (SDE) are applied to minimize the weighted sum of makespan and maximum lateness. In Sun et al. (2003) powerful heuristics for minimizing the makespan in a fixed three machine assembly-type flowshop problem are presented.

The literature about the distributed permutation flowshop problem is comparatively small, especially when compared with that of the AFSP and PFSP. The DPFSP is introduced in Naderi and Ruiz (2010) for the first time. They developed six different Mixed Integer Linear Programming (MILP) models and proposed two simple factory assignment rules and 14 heuristics based on dispatching rules, effective constructive heuristics and VND methods. More recently, in Gao et al. (2013) a TS algorithm with a better performance when compared to previous algorithms presented by the same authors is presented. The authors of Lin et al. (2013) have proposed an effective Iterated Greedy method and in Wang et al. (2013b) an Estimation of Distribution algorithm is proposed. The authors of Hatami et al. (2013) introduced for the first time the DAPFSP and proposed a MILP, three constructive algorithms and a VND. To the best of our knowledge, the DAPFSP with a single assembly machine has not been studied by any other authors in the literature.

Setup times are also considered in the non-distributed assembly flowshop literature (and much more in the regular flowshop). The authors of Yokoyama (2004) presented a two-stage production system, where there is a single production machine with setup times that produces parts and a single assembly machine. A near-optimal schedule is obtained by using a pseudo-dynamic programming method and a tight lower bound is proposed to evaluate its accuracy. The objective function considered is the minimization of the mean completion time. The same author built upon the previous model in Yokoyama (2008) by extending the single machine manufacturing stage to a flowshop with setup times. A pseudo-dynamic programming method and a branch-and-bound procedure are presented. The authors of Al-Anzi and Allahverdi (2007) addressed the two-stage AFSP with sequence independent setup times. They derived a dominance relation and applied SDE, PSO, TS and Earliest Due Date heuristics to minimize the maximum lateness. The same model is considered in Allahverdi and Al-Anzi (2009), where the authors presented a dominance relation and proposed three heuristics to minimize the makespan. The authors of Hatami et al. (2010) presented a three stage AFSP by considering a transfer stage as a middle stage and SDST in the first stage. They presented a mathematical model, a lower bound and two heuristics (TS and SA) to solve the problem. In Mozdgir et al. (2013) the authors also addressed the two-stage AFSP by considering multiple non-identical assembly machines and SDST in the first production stage. They developed a MILP and a hybrid VNS heuristic to minimize the weighted sum of makespan and mean completion time. Comprehensive reviews of the state-of the art of scheduling with setup times are carried out in Yang and Liao (1999), Cheng et al. (2000), Ruiz and Maroto (2005), Allahverdi et al. (1999, 2008) and Allahverdi (2015). As can be seen, the DAPFSP with SDST considered in this chapter has not been, to the best of our knowledge, studied before in the scheduling literature.

4.3 Simple constructive heuristic methods

The DPFSP is an $\mathcal{NP}$-Hard problem if $(n > f)$ Naderi and Ruiz (2010). Therefore, the DAPFSP with sequence dependent setups is also $\mathcal{NP}$-Hard as the DPFSP is a special case. As a result, the design of heuristic methods for obtaining good solutions in reasonable CPU times is necessary. In the following we present two simple constructive heuristics.

We first present a simple example problem that will be used to illustrate the proposed heuristics. The example consists of eight jobs ($n = 8$), three products ($t = 3$), two factories ($f = 2$) with a two machine flowshop each ($m = 2$). The assembly programs of the three products are: $N_1 = \{1, 6, 7\}$, $N_2 = \{2, 5\}$ and $N_3 = \{3, 4, 8\}$. Tables 4.1 to 4.3 present job processing times at factories, product assembly times at the single machine assembly stage and assembly

	Job							
Machine	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
M_1	46	48	94	2	4	47	50	33
M_2	47	2	83	13	69	42	26	95
	Product 1			Product 2			Product 3	
M_A	30			60			89	

Table 4.1: Job and product assembly times for the example.

Product	Product		
	T_1	T_2	T_3
T_0	6	3	1
T_1	0	4	5
T_2	3	0	6
T_3	7	2	0

Table 4.2: Assembly stage setup time matrix for the example.

and production machine setup matrices, respectively.

We introduce some necessary notation. π represents a *product sequence*, that is, a possible sequence for the assembly of the products, e.g., $\pi = \{1, 3, 2\}$. Each product h is composed of a number of jobs and a possible sequence for these jobs is referred to as π_h, denoting the *job sequence for product* h, e.g., $\pi_1 = \{7, 6, 1\}$, $\pi_2 = \{2, 5\}$ and $\pi_3 = \{8, 3, 4\}$ are possible job sequences for the three products in the example. A *Complete job sequence*, π_T, represents a possible sequence of the all jobs, and is the result of concatenating all job sequences for the products after the master product sequence π, e.g., $\pi_T = \{7, 6, 1, 8, 3, 4, 2, 5\}$ following the example. To start processing the first job at each factory and for assembling the first product at the assembly stage, an initial setup is necessary. We use J_0 and T_0 to represent the first dummy job and product, respectively.

To assign jobs to factories, the two job to factory assignment rules pre-

Job	Machine 1								Machine 2							
	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8
J_0	2	7	3	8	4	1	9	3	8	9	7	1	9	8	7	4
J_1	0	5	9	4	9	1	3	2	0	1	8	8	9	2	6	4
J_2	3	0	7	3	2	1	6	7	7	0	2	5	2	1	3	6
J_3	4	4	0	5	5	8	3	4	5	9	0	4	9	1	6	9
J_4	8	2	4	0	2	1	3	3	4	2	5	0	8	8	1	1
J_5	5	3	3	4	0	3	7	5	1	4	3	2	0	6	1	5
J_6	8	1	8	4	3	0	1	3	1	4	8	2	7	0	6	6
J_7	5	5	8	3	7	4	0	3	7	7	5	7	4	2	0	5
J_8	9	3	8	2	7	8	7	0	7	7	1	8	1	5	6	0

Table 4.3: Production stage setup time matrix for the example.

sented in Naderi and Ruiz (2010) are considered in this chapter. The first one, referred to as (NR_1), assigns job j to the factory with the lowest current $C_{\max}$, not considering job j. The second rule (NR_2) assigns job j to the factory with the lowest $C_{\max}$ after scheduling job j.

4.3.1 Heuristic 1

The first heuristic obtains a complete job sequence π_T and consists of three simple steps. The first obtains a product sequence (π) on the single assembly machine. The product with the minimum sum of initial setup and assembly time is scheduled first in π. The remaining $h - 1$ products are scheduled one by one, each time selecting the product with the smallest completion time after being scheduled, considering the sequence dependent setup time. Once all products are scheduled the second step in the heuristic determines the job sequence (π_h) of each individual product h. The jobs of each product h are considered one by one. Initially all factories are empty. Therefore, the first f jobs with the minimum completion times (initial setup plus processing time) are the first f jobs on π_h and occupy the first positions in the f factories. Of course, if $|N_h| \leq f$, π_h is equal to the the assembly program of product h, N_h. Otherwise, after f initial jobs are scheduled, the other $|N_h| - f$ jobs from product h are considered. Among the remaining jobs of product h, the job that is scheduled next is the one resulting in the smallest completion time after applying either the NR_1 or NR_2 job to factory assignment rules. The process continues until all jobs of product h have been considered. This second step is applied to each product separately to determine the job sequence for each individual product. After all products have been considered, the third step constructs the complete job sequence π_T by putting together all obtained π_h, following the product order established in π. At this point there are two possibilities: to assign all jobs in π_T to factories using the NR_1 or NR_2 rules. Depending on the case we have the proposed heuristic CH_{11} or CH_{12}, respectively. The sequence of products in the assembly stage is simply determined by ordering the products by increasing completion time of all the jobs in the production stage. To better illustrate the heuristic, all steps are explained through the previous example.

Product 1 is considered as the first product to be included into π. It is scheduled first in the single assembly machine which results in a completion time of 6+30=36 (considering the initial setup and the assembly times). The same procedure is carried out for the remaining products 2 and 3 which result in completion times of 3+60=63 and 1+89=90, respectively. Since product 1 results in the shortest completion time, it is scheduled first in π. Now we have to reconsider products 2 and 3 in the single assembly machine. They are scheduled now after product 1 which has been already scheduled. The completion times are 36+4+60=100 (completion time of product 1 plus the setup time in the assembly stage between products 1 and 2 and processing time of product 2) for product 2 and 36+5+89=130 for product 3. Therefore, product 2 is scheduled after product 1. Finally, no additional calculations are needed for scheduling the last product 3 in the third position. As a result, the product sequence π is $\{1, 2, 3\}$. Note that this first step of the heuristic is carried out $\frac{t(t+1)}{2} - 1$ times and therefore has a computational complexity of $\mathcal{O}(t^2)$. The next step is to find a good job sequence for each product h, π_h.

Recall that there are $|N_h|$ jobs that belong to product h. We consider product 1 as an example that consists of jobs $\{1, 6, 7\}$ in the example.

We begin by calculating the completion times of jobs 1, 6 and 7, separately. Since the two available factories are empty, we consider the initial setups and the completion times on the two machines at the flowshop of each factory. For example, the completion time of job 1 is 2 (initial setup at machine 1)+46 (processing time at machine 1)+47 (processing time at machine 2)=95. Note that the initial setup of 8 units in machine 2 can be performed before job 1 arrives to that machine. Applying the same procedure we calculate the completion times for jobs 6 and 7 which are 90 and 85, respectively. Since we have $f = 2$ factories, we select the f jobs with smallest completion times and schedule them. In this case jobs 7 and 6 are scheduled in factories 1 and 2, respectively and occupy the first two positions of the product sequence for product 1 (π_1). To schedule the remaining jobs in π_h, each one should be tested at each factory using either the NR_1 rule for CH_{11} or NR_2 for CH_{12}. The job resulting in the minimum completion time is scheduled next in π_h. This process continues until all jobs in N_h have been scheduled and is repeated for all the product sequences. In this example only job 1 remains and therefore occupies the last position in π_1. Therefore π_1 is $\{7, 6, 1\}$. Applying the same procedure results in the job sequences for products 2 and 3 to be $\pi_2 = \{2, 5\}$ and $\pi_3 = \{4, 8, 3\}$, respectively. Note that for each product h this second step requires to first calculate the minimum completion times of all jobs ($|N_h|$ steps) plus ordering these jobs according to these completion times ($|N_h| \log(|N_h|)$ steps) and assigning them to the first f factories (f steps). The remaining $|N_h| - f$ jobs are inserted one at a time using NR_1 or NR_2. This has $\frac{(|N_h|-f)(|N_h|-f+1)}{2}$ steps which are multiplied by f if using NR_2. It is difficult to calculate the computational complexity for this step as usually $|N_h|$ is not expected to be orders of magnitude larger than f and therefore the term $-f$ in the previous expression is important. However, if we assume that $|N_h| \gg f$ and that there is a single product where $|N_h| = n$ then the computational complexity of this second step is $\mathcal{O}(n^2)$ for NR_1 and $\mathcal{O}(n^2 f)$ for NR_2. Note however that this is a pathological worst case and the empirical complexity is expected to be much lower. Finally, in the third step the complete job sequence π_T is completed by concatenating all job sequences following the product sequence π. This sequence is therefore $\pi_T = \{7, 6, 1, 2, 5, 4, 8, 3\}$. In order to calculate the maximal makespan among the factories, the individual jobs in π_T are assigned to factories in the order they appear in π_T, using the rules NR_1 or NR_2 for heuristics CH_{11} and CH_{12}, respectively. This last step has a computational complexity of $\mathcal{O}(nf)$. Therefore, considering that $n \gg f$, the overall worst case computational complexity of this first heuristic is $\mathcal{O}(n^2)$ for NR_1 or $\mathcal{O}(n^2 f)$ for NR_2. In the considered example, we obtain the makespan value of 386 for CH_{11} and 387 for CH_{12}. The solution given by CH_{11} is represented as a Gantt chart in Figure 4.1. Additionally, a flowchart of heuristic 1 is given in Figure 4.2.

4.3.2 Heuristic 2

This heuristic is based on the second constructive method presented in Hatami et al. (2013). The idea is to consider the production stage and to sequence all jobs of each product and construct the different π_h sequences so that priority is given to products whose jobs have small completion times. In this way, the single assembly machine is occupied as soon as possible. In order to obtain good job sequences π_h for all products, the second step of the previous heuristic 1 is used. After all jobs for a given product h are scheduled, we calculate the earliest assembly start time for product h, denoted by E_h which is equal to $\max_{j=1}^{|N_h|}\{C_j\}$. After all individual product job sequences are determined, the product sequence π, is formed by sorting all t products according to ascending values of E_h. Finally, the complete sequence π_T is obtained after concatenating all job sequences π_h following the product sequence established in π. Similarly to heuristic 1, jobs in π_T are assigned to factories using NR_1 or NR_2 which results in heuristics CH_{21} and CH_{22},

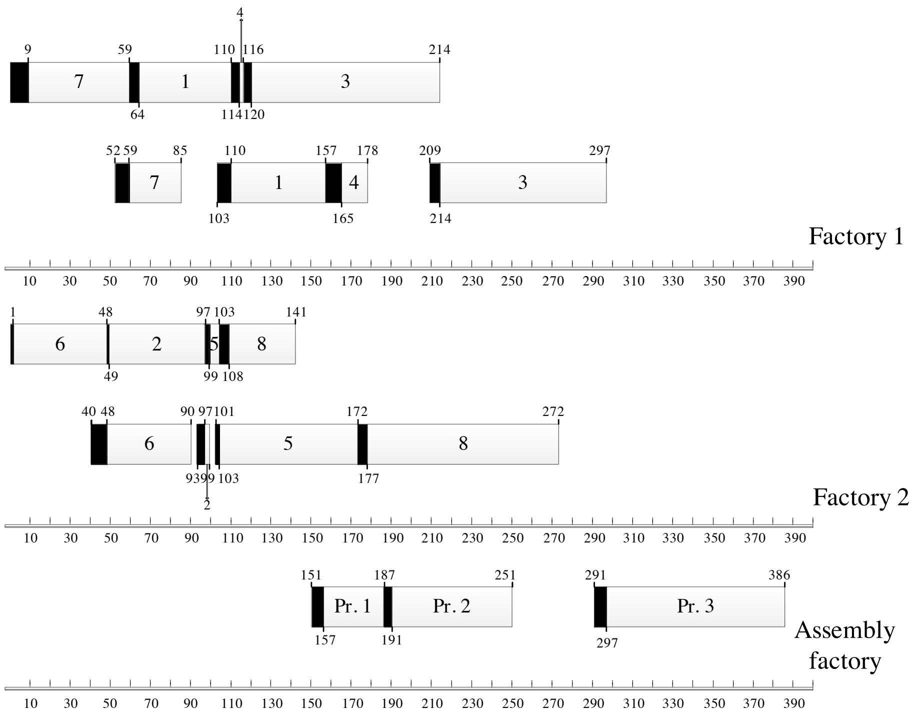


Figure 4.1: Gantt chart with the result of CH_{11} for the example problem.

respectively. The sequence of products for the assembly stage is obtained as in heuristic 1. The computational complexity of this second heuristic is dominated by the second step, which corresponds to the second step of the previous heuristic 1. Therefore, the computational complexity is the same in the worst case: $\mathcal{O}(n^2)$.

Following the job sequences obtained for the three products in the example of the previous heuristic, the earliest assembly start times of the products are $E_1 = 157$, $E_2 = 78$ and $E_3 = 191$. Therefore the product sequence π is $\{2, 1, 3\}$ by sorting all E_h in ascending order. The complete sequence π_T is therefore $\{2, 5, 7, 6, 1, 4, 8, 3\}$. After assigning each job to factories we obtain

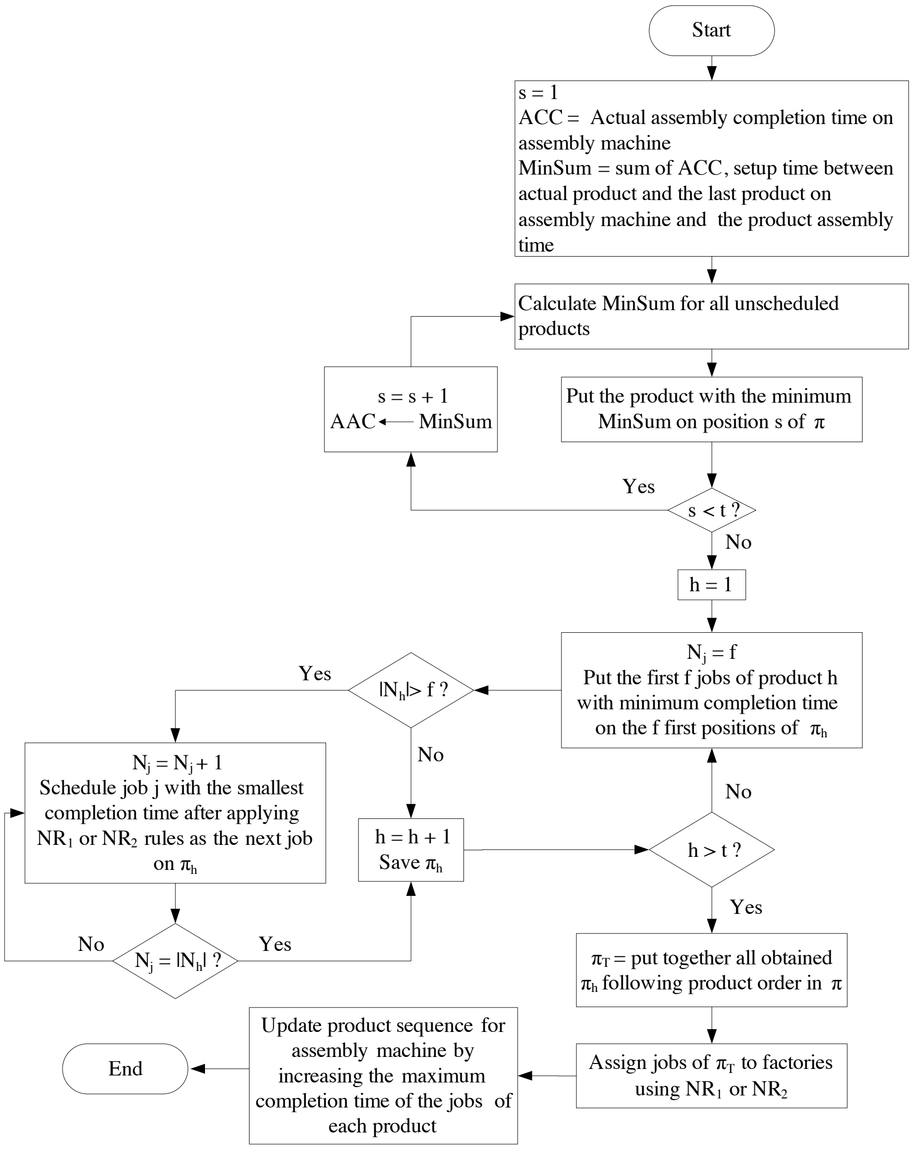

Start
s = 1
ACC = Actual assembly completion time on assembly machine
MinSum = sum of ACC, setup time between actual product and the last product on assembly machine and the product assembly time
Calculate MinSum for all unscheduled products
s = s + 1
AAC ← MinSum
Put the product with the minimum MinSum on position s of π
Yes
s < t ?
No
h = 1
$N_j = f$
Put the first f jobs of product h with minimum completion time on the f first positions of π_h
Yes
$|N_h| > f$?
No
$N_j = N_j + 1$
Schedule job j with the smallest completion time after applying NR_1 or NR_2 rules as the next job on π_h
h = h + 1
Save π_h
No
h > t ?
Yes
No
$N_j = |N_h|$?
Yes
π_T = put together all obtained π_h following product order in π
Update product sequence for assembly machine by increasing the maximum completion time of the jobs of each product
End
Assign jobs of π_T to factories using NR_1 or NR_2

Figure 4.2: Flowchart of Heuristic 1.

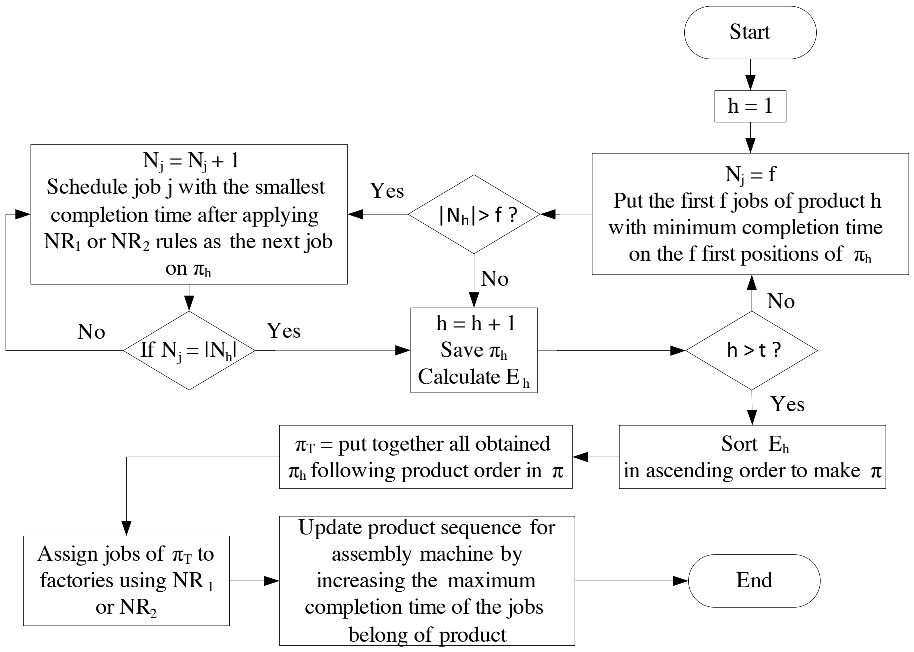

Start
h = 1
N_j = f
Put the first f jobs of product h with minimum completion time on the f first positions of π_h
|N_h| > f ?
Yes
No
N_j = N_j + 1
Schedule job j with the smallest completion time after applying NR_1 or NR_2 rules as the next job on π_h
If N_j = |N_h|
No
Yes
h = h + 1
Save π_h
Calculate E_h
h > t ?
No
Yes
Sort E_h in ascending order to make π
π_T = put together all obtained π_h following product order in π
Assign jobs of π_T to factories using NR_1 or NR_2
Update product sequence for assembly machine by increasing the maximum completion time of the jobs belong of product
End

Figure 4.3: Flowchart of Heuristic 2.

makespan values of 387 for heuristic CH_{21} (NR_1 rule) and 391 for heuristic CH_{22} (NR_2 rule). A flowchart of this heuristic 2 is given in Figure 4.3.

The four proposed heuristics will be tested later on as seed solutions of the other proposed approaches for solving the DAPFSP-SDST problem.

4.4 A simple Variable Neighborhood Search

Variable Neighborhood Descent (VND) is the simplest variant of the more general Variable Neighborhood Search (VNS) of Hansen and Mladenović (2001). Starting from an initial solution, VND explores different neighborhood structures, $N_1, \ldots, N_q$. These are usually explored in increasing cardinality starting with the smallest neighborhood N_1. The search continues with N_2 only after a local optimum has been obtained in N_1. If the local optimum obtained after exploring N_2 is different from the one obtained after analyzing N_1, the search goes back to exploring N_1. The process ends when all neighborhoods, including N_q, have been searched and the final solution is a local optimum with respect to all neighborhood structures. VND is very simple yet it performs well for the distributed flowshop and DAPFSP problems as shown in Naderi and Ruiz (2010) and in Hatami et al. (2013). In the following we summarize the proposed VND which employs two different solution representations and two neighborhood structures.

4.4.1 Solution representation

In this work, and differently from Hatami et al. (2013), we consider two different solution representations. The base encoding is a permutation of all jobs, i.e., we work with the complete job sequence π_T. Using this encoding we define the full permutation solution representation or (Pr1) as the ordering of the n jobs regardless of the products to which they belong. Hence, $n!$ different job permutations are possible with this representation.
Pr1 is a relaxation of the more restricted representation given in Hatami et al. (2013). This second representation, referred to as multi-permutation or Pr2 is also a complete job sequence but the jobs belonging to the same product are never separated and intermingled with jobs belonging to other products. Following the previous example, if we have a product sequence $\pi = \{2, 3, 1\}$, two possible representations could be $\{2, 5, 8, 3, 4, 7, 6, 1\}$ or $\{5, 2, 4, 8, 3, 7, 1, 6\}$. However, $\{2, 8, 5, 3, 4, 7, 6, 1\}$ is not valid as job 8, which belongs to product 3 is scheduled before job 5 which belongs to product 2 and the product sequence π forces all jobs of product 2 to be scheduled before all jobs of product 3. Note that Pr2 is smaller than Pr1 as in total Pr2 contains $t! \times \prod_{h=1}^{t} |N_h|!$ possible solutions.

4.4.2 Pr1 neighborhoods

Two neighborhoods are considered after the work of Naderi and Ruiz (2010), the first one, referred to as LS_1, works at each factory by extracting each job and reinserting it in all possible positions of the PFSP at that factory. The process continues until all jobs have been examined with no improvements in the $C_{\max}$ for all factories. The second neighborhood, LS_2, takes all jobs assigned to each factory and inserts them at all possible positions in all other factories looking for a makespan improvement at the involved factories. For more details, the reader is referred to Naderi and Ruiz (2010).

4.4.3 Pr2 neighborhoods

Again two neighborhoods are employed. These are based on the VND proposed in Hatami et al. (2013). The first neighborhood is referred to as LS_P and works over the product sequence π. It extracts and reinserts each product into all possible $t-1$ positions of π. Note that this is equivalent to extracting and inserting the block of consecutive jobs that correspond to each product h in π_T. The second neighborhood is referred to as LS_J. It is also an insertion neighborhood but in this case all jobs that make a product are extracted and inserted into all possible positions of the job sequence for product h, i.e., all t products are considered and all of their $|N_h|$ jobs are extracted and inserted into all job sequences. After each insertion and in both neighborhoods we obtain a complete job sequence π_T, therefore, all jobs need to be assigned to factories using the NR_1 or NR_2 assignment rules. More details are given in Hatami et al. (2013).

4.5 Iterated Greedy algorithm

Iterated Greedy (IG) was first applied to the regular permutation flowshop problem by Ruiz and Stützle (2007) with the objective of minimizing makespan. The good results obtained have encouraged the application of the

IG methodology to other scheduling problems. Regular flowshops with blocking constraints were approached by Ribas et al. (2011). No-wait flowshop was successfully solved with IG algorithms by Pan et al. (2008). IG showed excellent performance in no idle and mixed no-idle flowshops recently in Pan and Ruiz (2014). The SDST PFSP was tackled with IG methods in Ruiz and Stützle (2008). Also, other objectives apart from makespan have been considered, like tardiness Framinan and Leisten (2008) and total flowtime Pan and Ruiz (2012). Multiobjective flowshops have also been adequately solved with IG techniques in Minella et al. (2011) or even with the addition of setup times in Ciavotta et al. (2013). Finally, and as commented in Section 4.2, the DPFSP has been also solved with IG methods by Lin et al. (2013). Given all these previous successes, applying IG to the DAPFSP-SDST seems promising. The most relevant characteristic of the IG methodology is its simplicity which does not preclude obtaining competitive results for most tested scheduling settings. IG has very few parameters and does not employ specific problem knowledge. As with most metaheuristics, IG starts from a high-quality initial solution. This starting solution is initially equal to both the incumbent and the best solution. Then, usually four phases are iteratively applied to the incumbent solution until a user set termination criterion is reached. The first phase is a partial destruction of the incumbent solution where some elements of it are (usually randomly) removed. The second phase consists of the reconstruction of the incumbent solution. The removed elements are reinserted in the solution following a greedy heuristic. The result is a new complete solution. The third phase is a local search where the complete solution is improved. The fourth and last operator is the application of an acceptance criterion to decide if the new solution replaces the incumbent one.

In the proposed IG we will test which one of the four proposed heuristics (CH_{11}, CH_{12}, CH_{21} or CH_{22}) will serve as a method to construct the initial solution. In the following sections we explain the four phases of the proposed IG. Note that there are differences depending on the solution representation

Pr1 or Pr2.

4.5.1 Destruction, reconstruction and local search for Pr1

After the initial solution has been obtained we have a starting complete job sequence π_T along with a list of all jobs assigned to each factory. Let us denote by π_f the sequence of jobs assigned to a factory $f \in F$. In the destruction phase, a percentage of the n jobs ($d\%$) jobs are randomly selected, without repetition, removed from the factories and inserted into a list in the order in which they were selected. Note that according to Naderi and Ruiz (2010), no factory must be left empty when minimizing makespan. Therefore, a selected job will not be removed from a factory if it is its last job. The destruction procedure, explained in Pseudocode 5, returns the list of removed jobs D and all sequences of jobs assigned to factories, after the removal of the jobs.

Pseudocode 5 Destruction_Pr1(d)

```
$i \leftarrow 0$;
**while** $i < (d \cdot n/100)$ **do**
    $a \leftarrow$ Job randomly selected among the remaining $n - i$ jobs;
    $f \leftarrow$ Factory where job $a$ is assigned;
    **if** $|\pi_f| > 1$ **then**
        $D \leftarrow$ Insert job $a$;
        $\pi_f \leftarrow$ Remove job $a$ from $\pi_f$;
        $i \leftarrow i + 1$;
    **end if**
**end while**
**return** $D$ and all $\pi_f, f \in F$;
```

In the construction phase, jobs in D are selected, one by one, and reinserted into all possible positions in all factories. Among all positions, the one resulting in the sequence with the smallest $C_{\max}$ is chosen for the job. This process is repeated $d \cdot n/100$ times until D is empty. The local search operator used in the IG is the LS_1 procedure explained in section 4.4.2. In this local search, for each factory f, jobs are removed from π_f and reinserted into all $|\pi_f| - 1$ possible positions in factory f.

4.5.2 Destruction, reconstruction and local search for Pr2

The destruction operator is different from Pr1 in a small but important respect. Each one of the $d \cdot n/100$ removed jobs belong to a product h and we do not allow job sequences for any product h (π_h) to be empty, so at least one job must remain in the job sequence of products. In the reconstruction procedure, each job is inserted into all positions of its corresponding job sequence. To decide the best placement for each job in D, all job sequences π_h are coalesced into a complete job sequence π_T and jobs are assigned to factories following the NR_1 or NR_2 job to factory assignment rules. The process is finished when D is empty and all product job sequences contain all the jobs. For the local search we use a product inter-exchange variant of the aforementioned LS_P local search of Pr2. We denote this local search by LS_{PI} and all $t \times (t-1)$ pairs of products are interchanged in the product sequence π. Duplicate moves are ignored and the inter-exchange resulting in the best improving $C_{\max}$ is carried out. The process is repeated until all movements result in non-improving makespan values.

4.5.3 Acceptance criteria

Similar to most existing IG literature, including the previously cited papers, once the first three phases (destruction, reconstruction and local search) are carried out over the incumbent solution, we obtain a possibly different schedule and must determine if it replaces the incumbent one. It is known that a simple descent acceptance criterion, i.e., accepting new solutions only if they improve the best found $C_{\max}$ value, results in IG methods that are prone to stagnation and premature convergence. In the initial work of Ruiz and Stützle (2007) it was proposed that a simulated annealing-like type of acceptance criterion with a constant temperature, based on the earlier work of Osman and Potts (1989) is enough to avoid premature convergence. This acceptance criterion is as follows. Let us denote by π'_T to the incumbent complete solution after the first three phases have been applied and by π_T to the previous solution. Obviously if $C_{\max}(\pi'_T) < C_{\max}(\pi_T)$ then the new solution π'_T is directly accepted. If this is not the case, then solution π'_T is probabilistically accepted following the expression $random \leq e^{-\frac{C_{\max}(\pi'_T)-C_{\max}(\pi_T)}{\mathit{Temp}}}$ where $random$ is a random number uniformly distributed between 0 and 1. Note that $Temp$ is another expression that was proposed originally by Osman and Potts (1989) as $\mathit{Temp} = T \cdot \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}}{n \cdot m \cdot 10}$ where T is a factor that needs to be calibrated. This constant temperature simulated annealing-like type of acceptance criterion has been extensively used in the IG literature. For example Ruiz and Stützle (2008) used the same acceptance criterion albeit their problem considered sequence dependent setup times. There are at least three potential improvements to this acceptance criterion when applying it to our DAPFSP-SDST problem. First, $Temp$ is not correctly calculated as it does not consider the distributed factories, assembly stage, number of products or setup times. It is not clear how to extend this calculation to obtain a sensible parameter. Second, as shown in Ruiz and Stützle (2007), Ruiz and Stützle (2008) and other authors, the T factor inside the calculation of $Temp$ proved not to be statistically significant in a wide range of values in extensive calibration tests. Third, in the temperature calculation of Osman and Potts (1989), the final probability of accepting a worse solution basically depends only on the difference $C_{\max}(\pi'_T) - C_{\max}(\pi_T)$. Let us examine this in detail. The expression $\mathit{Temp} = T \cdot \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} p_{ij}}{n \cdot m \cdot 10}$ can be reduced to just $\mathit{Temp} = T \cdot 5$, this is because processing times p_{ij}, as we will detail later, are commonly obtained from a uniform distribution in the range 1, 99 in most of the scheduling literature. The average of such a uniform distribution is $(1 + 99)/2 = 50$, therefore, we have that the numerator of *Temp* approximates to $n \cdot m \cdot 50$. Considering the denominator, $\mathit{Temp} = T \cdot \frac{n \cdot m \cdot 50}{n \cdot m \cdot 10}$ reduces to the stated $T \cdot 5$. There is a potential problem in this approach. The final probability of accepting a final solution depends on the size of the instance and on the magnitude of the $C_{\max}$ value. Take two instances A and B with corresponding $C_{\max}$ values of the incumbent and new solutions as $C_{\max}(\pi_{T_A}) = 100$, $C_{\max}(\pi_{T'_A}) = 110$,

$C_{\max}(\pi_{T_B}) = 1000$ and $C_{\max}(\pi_{T'_B}) = 1010$. Both new solutions for A and B are worse than the incumbent by 10 units. However, for instance A these 10 units translate into a 10% solution quality deterioration whereas for instance B, the same 10 units are only a 1% deterioration. The problem with the calculation given in Osman and Potts (1989) is that both cases have the same probability of acceptance.

To remedy these three potential shortcomings, and as an additional contribution of this chapter, we propose two additional acceptance criteria. The first one, and similarly to the one of Osman and Potts (1989) is very simple. We basically substitute the difference $C_{\max}(\pi'_T) - C_{\max}(\pi_T)$ for the Relative Percentage Difference (RPD) between the makespan value of these two solutions which is calculated as RPD $= \frac{C_{\max}(\pi'_T) - C_{\max}(\pi_T)}{C_{\max}(\pi_T)} \times 100$. This results in an acceptance criterion calculation as $random \leq e^{-\frac{RPD}{Temp}}$.
The second proposed acceptance criterion, and in order to avoid the statistically insignificant T factor is further simplified as follows: $random \leq e^{-RPD}$.

In total we will test three different acceptance criteria. The original in Osman and Potts (1989) as described, denoted as AC_1 and the two newly proposed ones, referred to as AC_2 and AC_3, respectively. We will later use sound statistical techniques to test if the two new proposed ones result in better solutions for the DAPFSP-SDST problem.

4.6 Calibration of the proposed VND and IG methods

For further clarification, a flowchart of the proposed VND and IG is shown in Figure 4.4.

We proceed with the calibration of the proposed methods. We are not interested in a high quality and fine tuned process. Instead, we will use some statistical tools to achieve a coarse calibration. The technique of choice is the

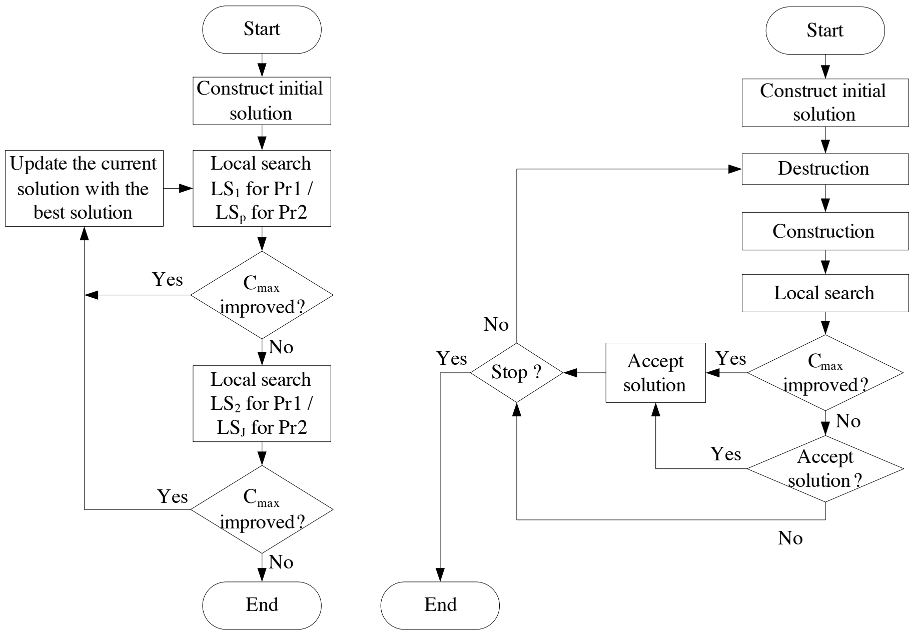

Start
Construct initial solution
Update the current solution with the best solution
Local search LS_1 for Pr1 / LS_p for Pr2
Yes
C_{max} improved?
No
Local search LS_2 for Pr1 / LS_J for Pr2
Yes
C_{max} improved?
No
End
Start
Construct initial solution
Destruction
Construction
Local search
No
Yes
Stop ?
Accept solution
Yes
C_{max} improved?
No
Yes
Accept solution ?
No
End

Figure 4.4: Flowchart of VND (left) and IG (right).

Design Of Experiments (DOE) approach Montgomery (2012) where we will basically be using screening factorial designs which are sound statistical techniques but still result in an exploratory calibration. The literature on calibration methodologies for metaheuristic methods is slowly gaining traction. Much more advanced methods are given in Bartz-Beielstein et al. (2010). We decide to use simpler approaches in order to have a clearer picture of the performance of the proposed methods. Should an advanced tuning methodology be used, it would be difficult to conclude if the proposed methods behave well because they are good for the problem studied or just because a fine tuning calibration has been carried out. The results of the experimental designs are examined by means of the Analysis of Variance technique (ANOVA). ANOVA is a robust parametric tool and at least three main hypotheses must be checked. Some are less important but others are crucial. From more to less important the hypotheses are; independence of the residuals, homoscesdasticity of the factor's levels (homogeneity of variance) and normality in the residuals. All these hypotheses are satisfied in all the following tests but it must be noted in any case that ANOVA has been proven to be extremely robust as stated in Basso et al. (2007). Other authors, like Rasch and Guiard (2004) study ANOVA in detail and test it against other non-parametric approaches with data that significantly departs from the three main hypotheses and conclude that ANOVA is preferable to non-parametric approaches most of the time. Furthermore, the most important hypothesis, the independence of the residual, is easy to satisfy in a controlled computational experimentation environment according to Ridge and Kudenko (2010). Therefore, the calibration methodology employed should give us a fair, not over-tuned and at the same time sound result.

A set of instances is generated to calibrate the proposed VND and IG. Calibrating methods with the same test instances that will be used in the computational evaluations is ill-advised. When a given method is calibrated with the same test instances later used for comparisons there is a big risk of having a bias in the results (over-fitting). There is no guarantee that with a different benchmark results will hold. Therefore, we calibrate the proposed methods with a different calibration benchmark. 60 instances are generated randomly with the following combinations of number of jobs (n), machines (m), factories (f), products (t) and distributions for the setup times of production and assembly machines. More specifically, n is tested at two levels (100, 200), m at three (5, 10, 20), f and t are also tested at three levels each, (4, 6, 8) and (30, 40, 50), respectively. Job processing times at the distributed flowshops in the production stage are generated according to a uniform distribution in the range $[1, 99]$ as is common in the scheduling literature. Finally, the product assembly times in the single machine assembly stage depend on the number of jobs assigned to each product h and follow a uniform distribution in the range $[1 \times |N_h|, 99 \times |N_h|]$. Finally, for the setup times we test two uniformly distributed intervals, $[1, 50]$ and $[1, 125]$ for production and assembly setups. All the calibration instances are available at `http://soa.iti.es`.

The response variable studied in the experiments is the Relative Percentage Deviation (RPD), where $RPD = \frac{SOL_{ALG} - BEST_{TOTAL}}{BEST_{TOTAL}} \times 100$. $BEST_{TOTAL}$ is the best known solution obtained over the course of this research for each calibration instance and SOL_{ALG} is the makespan value obtained by any algorithm tested over the same instance. Experimentation is performed in a scientific computation cluster with 30 blades. Each one with 16 GBytes of RAM memory and two Intel XEON E5420 2.5 GHz processors. Each processor has 4 physical computing cores (8 per blade) but no parallel computing is employed in this chapter as the 30 servers are only used to split the experimentation work and reduce the total time to obtain results. At each blade we use Windows XP virtual machines with one virtual processor with two cores and 2 GB of RAM memory.

4.6.1 VND calibration

The proposed VND mainly has three factors or algorithm features that should be tested. The first is the type of solution representation. This factor will be referred to as Pr and is tested at two variants, which correspond to the two different proposed solution representations of Section 4.4.1 (Pr1 and Pr2). The second factor is the two different job to factory assignment rules (NR) which is tested at two variants NR_1 and NR_2. The third and last factor is the simple constructive heuristic used for initialization (INI), tested at four variants (CH_{11}, CH_{12}, CH_{21} and CH_{22}). The response variable is the RPD and we carry out a multifactor ANOVA to analyze experiments. The number of treatments is the result of all the combinations of all previous factors ($2 \times 2 \times 4 = 16$) and each treatment is tested with all 60 calibration instances so the total number of experiences is $16 \times 60 = 960$. There is no need for replicates as the proposed VND methods are deterministic.

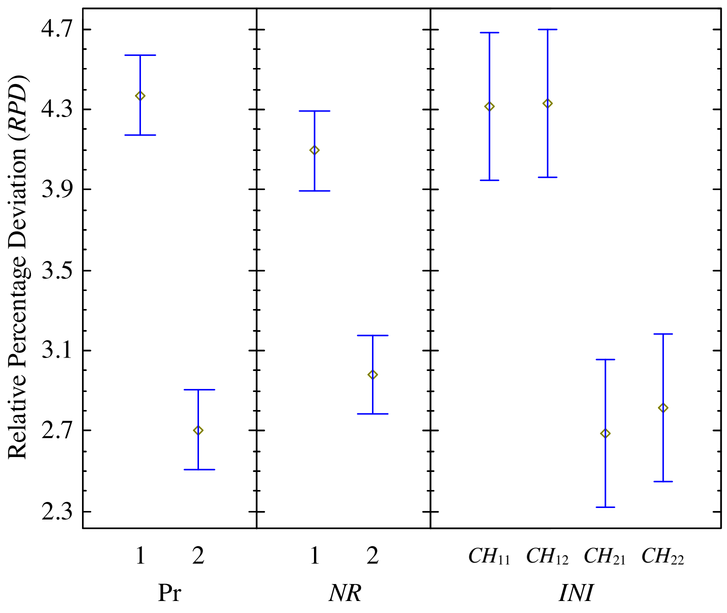


Figure 4.5: Means plot and 99% confidence level Tukey's HSD intervals for the type of solution presentation Pr, job assignment rules NR, and initial solutions INI for the proposed VND methods.

The analysis and ANOVA table shows that, all studied factors (Pr, NR and INI) are statistically significant. The most significant is the representation (Pr), then job to factory ass ignment rule (NR) and lastly the initial solution (INI). The means plot and 99% confidence level Tukey's Honest Significance Differences (HSD) intervals for all three factors are given in Figure 4.5.

The second solution representation, as well as the second job to factory assignment rules result in statistically better performance. As regards the solution representation, the larger cardinality of the solution space in the first representation deteriorates performance, possibly indicating that more neighborhoods or larger neighborhoods are needed. Our experiments confirm that the second job to factory assignment rule works better, which is in line with previous findings Naderi and Ruiz (2010), Hatami et al. (2013). However, this assignment rule does not have an effect on the constructive heuristics which only depend on the representation. In the end we select Pr$= 2$, $NR = 2$ and $INI = CH_{21}$.

4.6.2 Experimental parameter tuning of the IG

IG has three factors in common with VND to calibrate (Pr, NR and INI). These are tested at the same variants as before. Furthermore, there are three additional factors: percentage of jobs to destruct in the destruction phase (d), type of acceptance criterion (AC) and the value of T used in the calculation of $Temp$ (T). As explained in Section 4.5.3, we propose three different acceptance criteria. The first two (AC_1 and AC_2) do depend on the aforementioned parameter T, whereas the third (AC_3) does not have a T factor. As a result, we have to carry out two different experiments. In the first one we test two variants for Pr (Pr1 and Pr2), two variants for NR (NR_1 and NR_2), four variants for initial solution (CH_{11}, CH_{12}, CH_{21} and CH_{22}), three levels for d $(5, 10, 15)\% \times n$, two levels for acceptance criterion (AC_1 and AC_2) and three levels for $T : (0.5, 1, 2.5)$. This results in $2\times2\times4\times3\times2\times3 = 288$ algorithm configurations. Each one of the 60 calibration instances is run for five different replicates in each configuration resulting in $288 \times 5 = 1,440$ treatments as IG is an stochastic algorithm. Since each treatment is tested with all 60 calibration instances the total number of experiences is $1,440 \times 60 = 86,400$. Additionally, as IG is a metaheuristic with a stopping criterion, we set the elapsed CPU time as a termination criterion, which is fixed at $n \cdot m \cdot f \cdot 45$ milliseconds. This way of setting the termination criterion as a function of the size of the instance helps in decoupling the effect of the instance size in the results. Additionally, all algorithm configurations have the same CPU budget. Not doing so would result in a calibration biased for more time consuming configurations. We employ the same computers for this test as before. With this first experiment, the idea is to set the value of the parameter T for the first

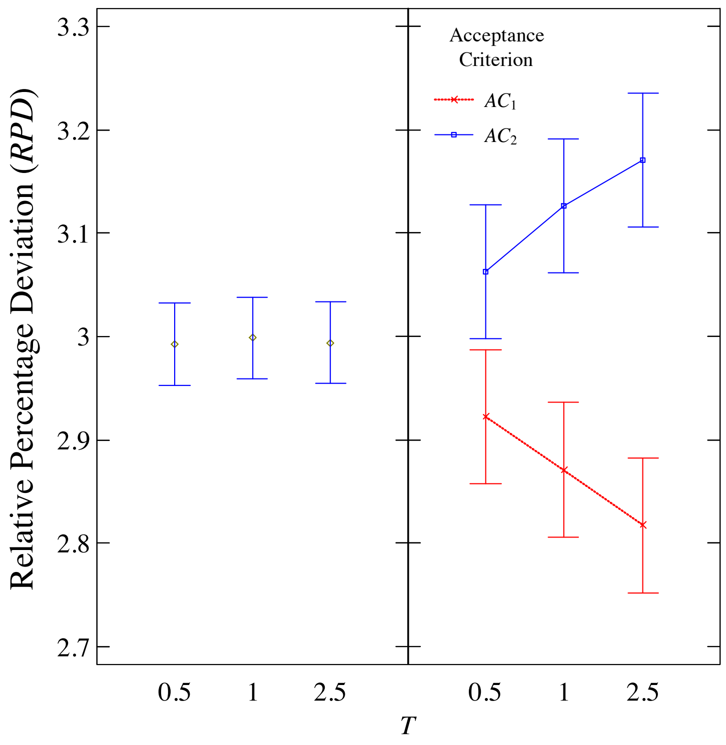


Figure 4.6: Means plot and 99% confidence level Tukey's HSD intervals for the temperature T parameter and the interaction between the temperature T parameter and acceptance criterion (AC) for the first calibration experiment for the proposed Iterated Greedy methods.

two levels of the acceptance criterion only $(AC_1$ and $AC_2)$. Once T is fixed, we will be able to analyze the three different acceptance criterion together in a second experiment. The results of the first experiment (not shown here due to reasons of space) indicate that the only non-significant factor is T with a p-value very close to 1. However, the interaction between T and the type of acceptance criterion (AC) is significant with a p-value of 0.0004. Both means plots, for the single factor as well as for the interaction are given in Figure 4.6.

As we can see, the single factor T is not significant as the three levels in the means plot completely overlap. The interaction is significant as the behavior of the T factor greatly depends on the type of acceptance criterion. For AC_1, which recall is the original Ruiz and Stützle (2007) type of acceptance criterion, increasing the value of T results in better solutions. Originally, Ruiz and Stützle (2007) tested values of T of 0, 0.1, 0.2, 0.3, 0.4 and 0.5. Here we have tested larger values but the three intervals overlap, meaning that even though solutions improve, the improvement is not consistent enough so as to be statistically significant. The situation is just the opposite for the second acceptance criterion AC_2 as increasing the value of T deteriorates solutions. This together with the fact that overall T is not significant and the previous studies into the IG methodology where T has been shown to be statistically insignificant reinforces our idea that T should be removed from the acceptance criterion. For the next experiment we set T at 2.5 for AC_1 and to 0.5 for AC_2.

The second experiment involves all previous factors and all three acceptance criteria but having fixed T as mentioned for the first two acceptance criteria. Therefore, the total number of experiences is now 43,200. The ANOVA results indicate that the interaction between the solution representation (Pr) and the job to factory assignment rule (NR) factors is the most significant effect. This interaction is shown in Figure 4.7.

Similar to VND, for the proposed Iterated Greedy method the second solution representation and the second job to factory assignment rule result in the best performance by a significant margin. Actually, with the exception of the percentage of jobs to destruct in the destruction phase (d), all other factors are not significant. The initial solution INI is not statistically significant with a p-value close to 0.25. However, this is across all instances. Some statistically significant differences are found in some instance groups when using CH_{21} or CH_{22}. Therefore, and again similar to VND, INI is set to CH_{21}. Of particular interest is the statistical insignificance of the type of acceptance criterion factor (AC) with a very large p-value of more than

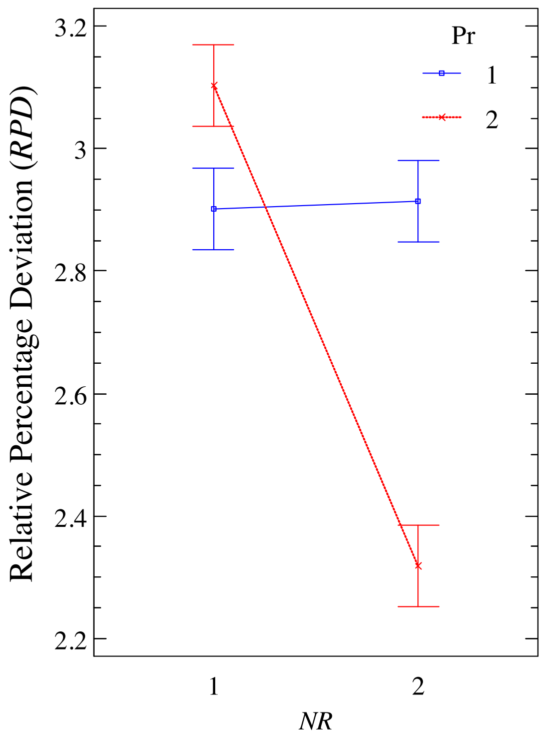


Figure 4.7: Means plot and 99% confidence level Tukey's HSD intervals for the interaction between the solution representation (Pr) and the job to factory assignment rule (NR) factors in the second calibration experiment for the proposed Iterated Greedy methods.

0.85. This means that there are very little (if any) differences between the three proposed acceptance criteria. The third proposed criterion does not employ a temperature factor. As a result, it is preferable to employ AC_3 as it is equivalent performance wise and at the same time simpler with one less parameter. In any case, for the final experiments we will also test the original Ruiz and Stützle (2007) acceptance criterion (AC_1) to conclude in a sound way if our new acceptance criterion is actually equivalent or not. Finally, d is marginally significant, offering different results when related with the instance factors (n, m, f and t). Since we want to avoid an instance-specific factor level, we finally settle for $d = 5\%$ regardless of instance size.

4.7 Computational evaluation

We are now ready to computationally test the proposed approaches. We are going to compare first the four proposed simple constructive heuristics CH_{11}, CH_{12}, CH_{21} and CH_{22}. These are very fast methods and take very little CPU time. We employ the same computing platform used for the calibration in the tests.

As mentioned, the benchmark of test instances is different from the previous calibration instances. Recall that in the calibration instances we have 60 random combinations of number of jobs (n), machines (m), factories (f), products (t) and distributions for the setup times of production and assembly machines. In the test instances we consider all possible combinations ($2 \times 3^3 \times 2 = 108$). For each combination we generate five different instances resulting in a total of 540 test instances. For all tested methods we calculate the Relative Percentage Deviation from the best solution known. This solution is the best obtained throughout the course of this chapter. All instances as well as the best solutions are available at `http://soa.iti.es`.

Table 4.4 shows the results of the four tested heuristics. There are 540 instances and four tested heuristics. Therefore, the total number of results is 2,160. We have grouped these by instance characteristics. CPU times are not reported as they are extremely small. As a matter of fact, among the 2,160 observed CPU times in the results, the maximum reported is just 0.079 seconds. The average observed CPU time in all results is only 0.008 seconds. It can be concluded that the reported heuristics are almost instantaneous even for the largest tested instances of 200 jobs, 20 machines, 8 factories and 50 products.

As can be seen, all four heuristics provide similar results. The average

		RPD			
		CH_{11}	CH_{12}	CH_{21}	CH_{22}
n	100	21.17	20.02	22.36	22.12
	200	13.11	12.24	13.22	13.31
	5	16.01	14.82	18.04	17.94
m	10	16.95	16.24	18.06	17.94
	20	18.46	17.34	17.27	17.25
	4	18.69	17.66	18.65	18.41
f	6	16.89	15.47	17.55	17.53
	8	15.83	15.27	17.16	17.19
	30	15.58	14.58	14.53	14.47
t	40	17.92	16.77	18.37	18.37
	50	17.92	17.04	20.47	20.30
Setup	$U[1, 50]$	12.70	12.11	10.43	10.31
interval	$U[1, 125]$	21.58	20.15	25.15	25.11
Average		17.14	16.13	17.79	17.71

Table 4.4: Average Relative Percentage Deviation (RPD) over the best known solution, grouped by instance characteristics of the proposed constructive heuristics.

deviations are between a little more than 16% and below 18%. Although not detailed here, there is a large variability in the results as well. The minimum observed RPD is just 3.59% and the maximum 51.51%. In order to closely analyze these results, we carry out an ANOVA statistical test on the obtained results. We consider all instance factors (n, m, f and t) as non-controllable factors as well as a single factor which is the heuristic, at four variants. The results of the ANOVA, which are not shown here due to reasons of space, indicate that three non-controllable factors n, t and f are very significant, in this order. This is expected as with more jobs and products the instances are harder to solve. Note, however, that a larger number of factories results in easier instances as there are less jobs per factory. As for the algorithms, the result is that CH_{12} is statistically better than the rest, followed by CH_{11} which is in turn better than CH_{21} and CH_{22}. There are no statistically significant differences between these last two methods. Note that this is not a contradictory result. While in the heuristic testing, CH_{12}, is the best heuristic, the calibration experiments for VND and IG resulted in CH_{21} being the best initialization method. We should not assume that the best heuristic should be used as an initialization for a metaheuristic as the initialization interacts with all other algorithm parameters.

In a separate experiment we test the more time consuming methods. The algorithms to compare are the VND with the parameters obtained in the calibration (Pr= 2, $NR = 2$ and $INI = CH_{21}$) and two similarly configured IG methods also from the calibration result. These differ only in the acceptance criterion. The common parameters are Pr= 2, $NR = 2$, $INI = CH_{21}$ and $d = 5\%$. In the first tested IG, referred to as IG$_1$, we employ the original Ruiz and Stützle (2007) acceptance criterion, AC_1 (which is, in turn, based on the criterion of Osman and Potts (1989). Since we need a value for T in this acceptance criterion, we use $T = 2.5$ as per the result of the calibration. The second tested IG, referred to as IG$_3$, uses the third proposed acceptance criterion AC_3 which does not have a T parameter.

The two Iterated Greedy methods need a termination criteria which is tested at two levels: $n \cdot m \cdot f \cdot 30$ and $n \cdot m \cdot f \cdot 60$ milliseconds elapsed CPU time ($\rho = 30, 60$). Additionally, since IG is stochastic, we run it five times for each instance and CPU time termination. Conversely, VND is deterministic and does not have a termination criterion and is therefore run only once with each instance. In total we have 540 results for the VND and 2,700 for each IG method and termination criterion (10,800 results). We first present the average Relative Percentage Deviation over the best solutions known for each instance. Table 4.5 shows these results, grouped by instance characteristics, among other information regarding CPU times.

As can be seen, VND results in relatively good solutions which average a RPD of 5.33% in all tests. The average CPU time needed is a little more than 37 seconds. Note how the CPU times clearly depend on the size of the instance (number of jobs n, number of machines m and number of products t). The

		RPD					CPU times (sec.)		
		IG_1		IG_3			$IG_{1/3}$	$IG_{1/3}$	
		$\rho = 30$	$\rho = 60$	$\rho = 30$	$\rho = 60$	VND	$\rho = 30$	$\rho = 60$	VND
n	100	2.39	1.33	2.23	**1.26**	8.02	210	420	18.04
	200	0.73	0.43	0.67	**0.37**	2.64	420	840	56.87
m	5	1.77	0.97	1.66	**0.93**	5.35	135	270	23.12
	10	1.54	0.88	1.44	**0.82**	5.45	270	540	32.91
	20	1.36	0.79	1.26	**0.69**	5.20	540	1080	56.33
f	4	1.43	0.77	1.33	**0.72**	4.37	210	420	37.44
	6	1.60	0.88	1.50	**0.83**	5.49	315	630	39.26
	8	1.64	0.99	1.52	**0.89**	6.14	420	840	35.65
t	30	0.65	0.49	0.58	**0.41**	3.18	315	630	15.42
	40	1.41	0.85	1.32	**0.79**	5.59	315	630	33.11
	50	2.61	1.30	2.45	**1.24**	7.23	315	630	63.82
Setup	$U[1, 50]$	0.93	0.53	0.89	**0.49**	3.12	315	630	37.55
interval	$U[1, 125]$	2.18	1.23	2.01	**1.14**	7.55	315	630	37.35
Average		1.56	0.88	1.45	**0.81**	5.33	315	630	37.45

Table 4.5: Average Relative Percentage Deviation (RPD) over the best known solution, grouped by instance characteristics and average CPU times of the proposed algorithms. Bold values indicate the best obtained average relative percentage deviations.

proposed Iterated Greedy methods are tested at two termination criteria and it is clear that with double the CPU time, the results improve. An interesting conclusion is that the third acceptance criterion (AC_3), albeit simpler and with one less parameter, gives better results when compared with the regular acceptance criterion. It is safe to conclude that IG_3, a simpler version with only one main parameter compared to the original version of Ruiz and Stützle (2007), works better for the studied problem.

We also carry out a multi-factor ANOVA to check if the observed average differences from Table 4.5 are indeed statistically significant. Once again we consider all instance characteristics as non-controllable factors. Preliminary tests indicate that VND is clearly not statistically better than the IG methods. Therefore, to avoid lack of normality in the residuals and to have a clearer picture of the performance of the IG methods, VND is removed from the final statistical test. We control two factors, the type of IG at two variants (IG_1 and IG_3) and the termination time ρ at two levels (30 and 60). The results of the ANOVA indicate that IG_3 is statistically better than IG_1, this is further illustrated in Figure 4.8. Upon a closer analysis we have modified the chapter. More in detailes, considering the two different stopping times and five replicates per instance and the 540 test instances we have 5,400 results for IG_1 and another 5,400 results for IG_3. Comparing these 10,800 cases we find that in 2,244 IG_1 is better than IG_3, in 418 both give the same makespan and in 2,738 IG_3 is better than IG_1. The fact that IG_3 is statistically better, on average, than IG_1 is because in the cases where IG_1 is better than IG_3 it is so by a small margin. However, when IG_3 is better than IG_1 the difference is larger. In any case, the differences are not very large. Note that in Table 4.5 for the large CPU time of $\rho = 60$, the grand average of IG_1 is 0.88 whereas for IG_3 the average is 0.81 so the difference is small.

From the results we have shown that the proposed heuristics provide reasonable results almost instantaneously whereas the presented VND method gives much better results which deviate, on average, about a 5% from the best known solutions. When doing so they require a larger, but still acceptable CPU time. The presented Iterated Greedy algorithms are of a much higher quality but need more CPU time. This time, however, can be set by the decision maker. With all these tools, plant managers have a wide range of algorithms with different CPU time demands and solution qualities to suit the needs of each moment.

4.8 Conclusions of the chapter

We have addressed the addressed Distributed Assembly Permutation Flowshop Scheduling Problem with the additional consideration of sequence dependent setup times at both production and assembly stages. This results in a con-

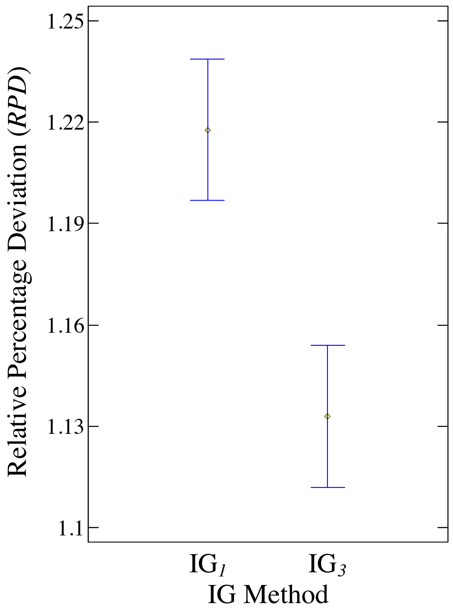


Figure 4.8: Means plot and 99% confidence level Tukey's HSD intervals for the type of Iterated Greedy method in the final test experiments.

siderably more realistic and applicable problem setting. The objective is the minimization of the makespan at the assembly stage.

We have presented two constructive heuristics, which are combined with two existing job to factory assignment rules. Furthermore, a simple and relatively fast metaheuristic based on Variable Neighborhood Descent (VND) is proposed, calibrated and analyzed. Additionally, we present an Iterated Greedy (IG) algorithm that has also been extensively analyzed. While IG is a very simplistic metaheuristic, we have simplified it further by proposing an acceptance criterion that does not consider a simulated annealing-like temperature as is common in the IG literature Ruiz and Stützle (2007). The result is a parameter-less acceptance criterion.

Sound and detailed statistical techniques have been employed to calibrate and to analyze the performance of all presented methods. The result is a battery of approaches that range from very fast (almost instantaneous) constructive heuristics that produce reasonably good results to more time consuming methods like VND or IG that reach close to optimality performance. Given the applicability of the researched problem and the range of proposed approaches, the work carried out in this chapter represents a solid step forward in solving more realistic distributed scheduling problems.

CHAPTER 5

THE DISTRIBUTED PARALLEL MACHINE AND ASSEMBLY SCHEDULING PROBLEM WITH ELIGIBILITY CONSTRAINTS

“The contents of this chapter are taken from the publication: Hatami, S., Ruiz, R., and Andrés-Romano, C. (2015). The distributed assembly parallel machine scheduling problem with eligibility constraints. *International Journal of Production Management and Engineering*, 3(1):13–23. ”

In this chapter we jointly consider several realistic scheduling extensions: First we study the distributed unrelated parallel machines problem where there is a set of identical factories with parallel machines in the production stage. Jobs have to be assigned to factories and to machines. Additionally, there is an assembly stage with a single assembly machine. Finished jobs at the manufacturing stage are assembled into final products in this second assembly stage. These two joint features are referred to as the Distributed Parallel Machine and Assembly Scheduling Problem or DPMASP. The objective is to minimize the makespan in the assembly stage. Due to technological constraints, machines cannot be idle and some jobs can be processed only in certain factories. We

propose a mathematical model and two high-performing heuristics. The model is tested with two state-of-the-art solvers and, together with the heuristics, 2220 instances are solved in a comprehensive computational experience. Results show that the proposed model is able to solve moderately-sized instances, and that one of the heuristics is fast, giving optimal solutions close to optimum in less than half a second in the worst case.

5.1 Introduction

Nowadays, the manufacturing industry faces many challenges, namely globalization, increasing product variety, complexity and customer demands, shorter product life cycles, higher demand of customized goods instead of mass production, uncertain and dynamic global market, etc. Of course, the strong competition from emerging and established economies has to be considered as well. One of the many tools to face these challenges and to meet customer's demands is to increase the product variety that companies offer. A wide product portfolio and diversified offer is a key asset to stay competitive in such an unpredictable and ever evolving market. Product variety has been defined by many authors as a number or collection of different things of a particular class of the same general kind (ElMaraghy et al., 2013). In recent years, assembly systems are such as techniques that are mostly used mass production. They have been also employed in various manufacturing systems so as to increase flexibility and the capability to increase product variety. These types of manufacturing settings are referred to as Assembly Scheduling Problems (ASP).

In an assembly system, different operations are performed independently, and potentially in parallel, to produce different components which are later assembled into finished products in assembly lines. A high variety of finished products, made from different combinations of produced components, can be produced in assembly systems.

Existence of more than one manufacturing facility in different geographical places may decrease some costs related to the production. To offset these costs, companies must operate different and specialized factories in what is known as Distributed Manufacturing Systems (DMS). In a DMS environment, several independent production centers or factories are run in parallel at potentially different geographical places. Furthermore, distributed manufacturing allows for greater flexibility and resiliency (Sluga et al., 1998). Other benefits of DMS are: higher product quality, lower production costs, reduced risks (Kahn et al., 2004; Chan et al., 2005b; Mahdavi et al., 2008). However, scheduling in DMS is more complicated than in a single production factory. In single production centers a job schedule for each set of machines has to be defined, while in DMSs, there are two interrelated decisions to be made: factory selection for each job and then scheduling at each factory.

As a conclusion, and in order to reap the benefits of both assembly systems (ASP) and distributed manufacturing (DMS), both aspects must be jointly considered.

In the studied problem of this chapter we consider two manufacturing stages: production and assembly. For production we have a set of distributed factories and for assembly there is a single assembly facility. Each one of the f distributed production centers (factories) has unrelated parallel machines as a shop configuration whereas the assembly stage consists of a single machine. Transportation time for transferring jobs from production centers to assembly stage is assumed negligible. By considering the above model we define the studied problem in this chapter as the Distributed Parallel Machine and Assembly Scheduling Problem (DPMASP).

More in specifically, in the DPMASP there is a set N of n jobs that has to be processed on a set F of f identical factories. Note that all factories are identical and have the same number of machines. Each factory has a set M of m unrelated parallel machines. Each job has to be processed at exactly one machine at one factory. Furthermore, there are eligibility constraints.

$LF_j \subseteq F$ is the subset of factories where job j can be assigned, where $f \geq |LF_j| \geq 1, j = 1, \dots, n$, job j can only be assigned to an eligible factory. There is a set T of t independent products. Each product is assembled at the single assembly machine M_A. For the assembly of product $h, h = 1, \dots, t$ a subset $N_h \subseteq N$ of jobs must have been produced at the distributed factories beforehand. Each job can only belong to an assembly program of a product, i.e., $\sum_{h=1}^{t} |N_h| = n$. The assembly of product h can only start when all jobs in N_h have been completed at the distributed factories. For the processing at the distributed manufacturing stage, p_{jk} denotes the processing time of job j at machine k of any factory. Note that all factories are identical and have the same number of machines. For the assembly stage, p_h denotes the assembly time of product h. All processing times are positive, deterministic and known integer quantities. The objective in the proposed DPMASP is to assign jobs to machines at factories in the distributed manufacturing stage, to schedule all assigned jobs to each machine at each factory and to schedule products at the single machine assembly stage while minimizing the makespan at this assembly stage. As regards the computational complexity of the DPMASP we can conclude that it is an $\mathcal{NP}$-Hard problem if $n >> f$ since the regular parallel machines problem (even in the case where there are two identical machines, i.e., the $P2//C_{\max}$ problem) is already $\mathcal{NP}$-Hard according to the results of Lenstra et al. (1977).

As we will later show, the DPMASP is an important generalization of existing problems that has not been studied before to the best of our knowledge. In this chapter we propose a mathematical model to solve the problem. The model is solved with two state-of-the-art commercial solvers and results are compared. Two high performing heuristics are proposed and are shown to give results that are, in many cases, close to the optimal ones. The rest of the chapter is organized as follows: In the next section we present a short literature review on related problems. In Section 5.3 we present a Mixed Integer Linear Programming (MILP) model to solve the considered problem.

Section 5.4 describes two simple constructive heuristics. Section 5.5 presents a comprehensive computational evaluation of the proposed MILP and simple constructive heuristics. Finally, some concluding remarks are provided in Section 5.6.

5.2 Literature review

As mentioned, the DPMASP contains parts from distributed manufacturing, assembly and parallel machines. As such, a complete literature review on each one of these three topics is clearly outside the scope of this chapter. Some of the closely related research will be reviewed instead. Regarding the assembly part of the proposed DPMASP, Lee et al. (1993) considered a three machine assembly-type flowshop problem (non distributed). The problem comprises two stages; in the first stage there are two production machines that produce two components for each single product. The second stage is a single assembly machine that assembles the two produced components to make each final product. They present a branch and bound algorithm and also an approximate procedure. Makespan minimization is considered as an objective function. Later, Potts et al. (1995) considered m parallel machines instead of the two production machines in the first stage. They produced approximated solutions with worse-case absolute performance guarantees. For the same problem of Lee et al. (1993), Hariri and Potts (1997) proposed a branch-and-bound algorithm, and Sun et al. (2003) presented different powerful heuristic algorithms. Also, Sung and Kim (2008) tried to expand the model presented by Lee et al. (1993) by adding multiple-assembly machines in the second stage. The objective is to minimize the sum of completion times. They proposed a lower bound and employed it in a branch-and-bound algorithm. An efficient and simple heuristic was also proposed. As mentioned, we consider eligibility constraints for assigning jobs to factories in distributed manufacturing stage. To the best of our knowledge, Lin and Li (2004) have a similar job to machine eligibility constraints. In this paper, the parallel machine scheduling problem with unit processing times is studied and polynomial algorithms are presented.

For the distributed part of the DPMASP we have to note that DMS is a general and broad manufacturing term. Focusing only on distributed scheduling problems, there are few studies about, distributed flowshops and jobshops. For example, the distributed permutation flowshop scheduling problem (DPFSP) was introduced for the first time by Naderi and Ruiz (2010). They proposed six different alternative MILP models, two simple factory assignment rules, fourteen heuristics and variable neighborhood descent methods. Later, Lin et al. (2013) and Wang et al. (2013b) proposed an effective Iterated Greedy (IG) method and an Estimation of Distribution algorithm on DPFSP, respectively. Later, Naderi and Ruiz (2014) presented a scatter search (SS) method for the DPFSP. This SS was shown to outperform existing methods. For an updated literature review on the DPFSP, the reader is referred to this paper of Naderi and Ruiz (2014). Recently, Fernandez-Viagas and Framinan (2015) have presented a modified iterated greedy algorithm for the DPFSP, which is shown to outperform the initial algorithms of Naderi and Ruiz (2010). However, there is no comparison between the SS of Naderi and Ruiz (2014) and this modified iterated greedy. The distributed jobshop problem considering two different criteria is studied first by Jia et al. (2002) and Jia et al. (2003) where they proposed Genetic Algorithm (GA) to solve the problem. Later, Jia et al. (2007), refined the previous GA. Chan et al. (2006b) studied the distributed jobshop with makespan objective, also using GA.

The only papers that we are aware of that jointly consider the assembly and distributed aspects are Hatami et al. (2013) which recently introduced the Distributed Assembly Permutation Flowshop Scheduling Problem (DAPFSP). In this problem, there are f distributed flowshop production centers and a single assembly center with a single machine. A MILP, several constructive heuristics and simple local search based Variable Neigborhood Descent (VND) methods were proposed. Xiong et al. (2014) presented a distributed two-stage assembly system with setup times. The authors considered f distributed factories where each factory has the same m processing parallel machines at the first stage and the same assembly machine at the second stage. Each assembled product consists of m components produced by parallel machines. They developed heuristic methods and three hybrid meta-heuristics to minimize the total completion time. The problem studied by Xiong et al. (2014) is different from the studied DPMASP. First, we consider a separated assembly stage, not an assembly operation at each factory. Second, we allow the different jobs composing a product to be produced in different factories. Third, each product might have a number of jobs (components) different from m.

As we can see, and to the best of our knowledge, there is no literature on the DPMASP.

5.3 Mixed Integer Linear Programming model

We present a mathematical model to solve the proposed DAPMSP. First we detail the indexes, parameters and variables are used:

Index	Description
i, j	denotes jobs, $i, j = 0, 1, \dots, n$, where 0 represents a dummy job
k	denotes machines, $k = 1, \dots, m$
q	denotes factories, $q = 1, \dots, f$
l, s	denotes products, $l, s = 0, 1, \dots, t$, where 0 represents a dummy product
M	a sufficiently large positive number, $M = 100000$
Parameter	**Description**
n	number of jobs
m	number of machines
f	number of factories
t	number of products
p_{jk}	processing time of job j on machine k
p_s	processing time of product s at the assembly stage
G_{js}	binary parameter equal to 1 if job j belongs to product s, and 0 otherwise

Variable	Description
X_{ijkq}	binary variable equal to 1 if job i is an immediate predecessor of job j on machine k in factory q
Y_{ls}	binary variable equal to 1 if product l is an immediate predecessor of product s at the assembly machine
C_j	completion time of job j at the production stage
CA_s	completion time of product s on assembly stage
$C_{\max}$	makespan

The objective function of the model is to minimize the makespan:
Min $C_{\max}$

subject to the following constraints:

$$\sum_{i=0,i\neq j}^{n} \sum_{k=1}^{m} \sum_{\substack{q=1\\ q\in LF_j\\ q\in LF_i}}^{f} X_{ijkq} = 1 \qquad \forall j \tag{5.1}$$

$$\sum_{j=0,j\neq i}^{n} \sum_{k=1}^{m} \sum_{\substack{q=1\\ q\in LF_i\\ q\in LF_j}}^{f} X_{ijkq} = 1 \qquad \forall i \tag{5.2}$$

$$\sum_{\substack{j=1\\ q\in LF_j}}^{n} X_{0jkq} = 1 \qquad \forall k, q \tag{5.3}$$

$$\sum_{\substack{i=1\\ q\in LF_j}}^{n} X_{i0kq} = 1 \qquad \forall k, q \tag{5.4}$$

$$\sum_{\substack{j=1,j\neq i\\ q\in LF_j}}^{n} (X_{ijkq} - X_{jikq}) = 0 \qquad \forall i, k, q, q \in LF_i \tag{5.5}$$

$$\sum_{k=1}^{m}\sum_{\substack{q=1\\ q\in LF_i\\ q\in LF_j}}^{f} X_{ijkq} + \sum_{k=1}^{m}\sum_{\substack{q=1\\ q\in LF_i\\ q\in LF_j}}^{f} X_{jikq} \leq 1 \qquad \forall i \in \{1,\ldots,n-1\}, j > i \tag{5.6}$$

$$C_j \geq C_i + p_{jk} + M(1 - X_{ijkq}) \qquad \forall i,j,k,q,q \in LF_i, LF_j \tag{5.7}$$

$$\sum_{l=0,l\neq s}^{t} Y_{ls} = 1 \qquad \forall s \tag{5.8}$$

$$\sum_{s=1,l\neq s}^{t} Y_{ls} \leq 1 \qquad \forall l \tag{5.9}$$

$$Y_{ls} + Y_{sl} \leq 1 \qquad \forall l \in \{1,\ldots,t-1\}, s > l \tag{5.10}$$

$$CA_s \geq (C_j \cdot G_{js}) + p_s \qquad \forall j, s \tag{5.11}$$

$$CA_s \geq CA_l + p_s + M(1 - Y_{ls}) \qquad \forall l, s, l \neq s \tag{5.12}$$

$$C_{\max} \geq CA_s \qquad \forall s \tag{5.13}$$

$$X_{ijkq} \in \{0,1\} \qquad \forall i,j,k,q,i \neq j, q \in LF_i, q \in LF_j \tag{5.14}$$

$$Y_{ls} \in \{0,1\} \qquad \forall l, s, l \neq s \tag{5.15}$$

$$C_j \geq 0 \qquad \forall j \tag{5.16}$$

$$CA_s \geq 0 \qquad \forall s \tag{5.17}$$

Note that $C_0 = CA_0 = 0$. Constraint sets (5.1) and (5.2) ensure that each job must have exactly one preceding and succeeding job, respectively. Sets (5.3) and (5.4) enforce that each machine at each factory has to have a dummy job 0 as predecessor and successor, respectively. Note that this is a special constraint, as we do not allow any machine at any factory to be empty due to technological or economic constraints. This also requires the total number of jobs in the shop (n) to be greater or equal than $f \cdot m$. Constraint set (5.5) ensures that if a job is sequenced on a machine, then its predecessor and successor must be processed on the same machine. Constraint set (5.6) controls that a job cannot be both a predecessor and successor of another job at the same time. Constraint set (5.7) determines that if job j is placed immediately after job i, its processing at machine k cannot start before the processing of job i in machine k finishes. Constraints (5.8) and (5.9) force that each product should have one predecessor and at most one succeeding product in the assembly factory, respectively. Constraint (5.10) controls that a product cannot be both a predecessor and a successor of another product at the same time in the assembly machine. Constraint (5.11) determines that each product h cannot begin to be assembled before all its jobs are completed in the corresponding machine. Constraint set (5.12) determines that if product s is placed immediately after product l, it cannot start to be assembled on the assembly machine before the assembling of product l in assembly machine has finished. Constraints (5.13) and (5.14)-(5.17) define the makespan and the domain of the decision variables, respectively. Note that only the necessary variables are defined, i.e., eligibility constraints are implicitly considered in the model.

5.4 Constructive heuristic methods

Let us first introduce a DPMASP example that will guide the exposition of the proposed heuristics. The example consists of fourteen jobs ($n = 14$), three products ($t = 3$), two factories ($f = 2$) with two unrelated parallel machines in each factory ($m = 2$). The assembly programs for each product are: $N_1 = \{2, 7, 8\}$, $N_2 = \{1, 3, 4, 10, 12, 13\}$ and $N_3 = \{5, 6, 9, 11, 14\}$, i.e., jobs 2, 7 and 8 must be finished in order to assemble product 1. Table 5.1 contains the job processing times on each machine at the production stage and eligibility constraints. Processing times for assembling products 1 to 3 are 3, 12 and 7, respectively.

Some additional notation is the following: A product sequence is represented by π, e.g., $\pi = \{2, 1, 3\}$. To assign all jobs belonging to the assembly program of product h to the unrelated parallel machines at the different facto-

		Job													
		J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}	J_{11}	J_{12}	J_{13}	J_{14}
Machine	M_1	7	3	4	3	1	3	7	4	9	7	8	3	4	7
	M_2	1	6	5	4	5	9	2	1	6	8	4	9	1	3
LF_j		1,2	1,2	1	2	1,2	2	2	1	2	1	2	1,2	2	1,2

Table 5.1: Job processing times and factory eligibility constraints for the example.

ries, a job to machine-factory assignment method is needed. After the application of this assignment procedure we obtain a job to machine-factory sequence for product h, referred to π_h, e.g., $\pi_1 = \{0, 8; 7, 2\}$, $\pi_2 = \{1-10, 3; 12, 4-13\}$ and $\pi_3 = \{14, 5; 6-9, 11\}$ as a possible job to machine-factory sequence for products of the example. At each π_h, each factory is separated by ";", each machine by "," and the sequence of jobs at each machine is separated by "-". A machine that is still empty (which can only occur in a partial solution) is denoted by "0" in its sequence. Following the previous example for π_2 we have that jobs 1, 10 and 3 are assigned to the first factory. Jobs 1 and 10 are assigned to the first machine in this factory in this order and job 3 to the second machine. Since π_h presents the job to machine-factory sequence of a single product h, π_T, referred to as the final job sequence, is the concatenation of the different π_h following the product sequence π. Following the previous example, $\pi_T = \{1-10-14, 3-8-5; 12-7-6-9, 4-13-2-11\}$. Once all jobs in the assembly program of a product h are completed in the production stage, it can be assembled on the assembly stage. Earliest assembling time of product h is denoted as E_h.

In this chapter two methods are employed to construct the product sequence π. The first one uses the Shortest Processing Time heuristic (SPT). This dispatching rule is known to reduce the average number of jobs in the system, in-process inventories and average job tardiness (Stafford et al., 2005). We obtain the SPT order using the product assembly times and refer to this method as PS_1. The second method, referred to as PS_2, sorts the products in ascending order of the earliest assembling times (E_h).

In the method to make job to machine-factory assignments for products, we need first some additional notation. We refer to U_h to the set of unscheduled jobs of product h assembly program, i.e., those jobs not yet assigned to machines at factories. S_{kq} is the set of jobs already scheduled at machine k inside factory q. With this in mind, the job to machine-factory assignment considers, for a product h, all jobs inside its assembly program, assigning first the unscheduled job with the earliest completion time at any machine in every eligible factory. More in details, we assign job $j^* \in U_h$ to machine k^* at factory q^* satisfying:

$$\{j^*, k^*, q^*\} = \underset{k \in m, q \in LF_j, j \in U_h}{\operatorname{argmin}} \left\{ \sum_{i \in S_{kq}} p_{ik} + p_{jk} \right\}$$

The process is applied until all jobs in the assembly program of product h are scheduled.

Both proposed constructive heuristics consist of three main steps: In the first step, the product sequence π is constructed. In the second step, the jobs inside the assembly program of each product are assigned following the previous job to machine-factory assignment procedure, following the order of products given in π. Finally, in the third step the sequence of products for the assembly stage is obtained by sorting products according to E_h in ascending order. We propose two heuristics with identical second and third steps and with a different process to build the product sequence π in the first step.

5.4.1 Heuristics PJ_1 and PJ_2

In heuristic PJ_1, PS_1 is used to determine the product sequence π. After processing all jobs in the production stage, E_h for each product h is calculated.

The product sequence on the assembly machine is updated by sorting E_h in ascending order and the final makespan is calculated. Pseudocode 6. explains PJ_1 in detail:

Pseudocode 6 Outline of the PJ_1 heuristic.

```
- Obtain product sequence, $\pi$, applying $PS_1$
- Use the job to machine-factory assignment procedure to assign all jobs of
  each product following the product order in $\pi$
- Calculate earliest assembling time of each product $h$, $E_h$
- Determine the product sequence $\pi$ on the assembly stage by sorting $E_h$ in
  ascending order
```

The second heuristic PJ_2 needs some careful explanation. It uses method PS_2 in the first step to make the product sequence π. However, PS_2 requires sorting products in increasing order of E_h. To calculate E_h, all jobs must be assigned to factories and machines. In heuristic PJ_2, each product's E_h is calculated in isolation. To calculate E_h of each product h, only jobs belong to product h are considered. Once E_h is calculated for all products, they are sorted in increasing order to form the product sequence π. This product sequence π is in turn used to apply again the job to machine-factory assignment for all products, which in the end gives us the final makespan.

The difference between heuristic PJ_2, and the first heuristic PJ_1, is just on the first step. As mentioned before, heuristic PJ_2, uses PS_2 to construct π. Therefore, Pseudocode of heuristic PJ_2 is not presented due to its similarity with heuristic PJ_1.

Note that if there are ties in the E_h of products, they are broken by taking the first product. Also the same rule is considered for breaking ties on the SPT rule which is used in heuristic PJ_1 to calculate π. As a final note, and to enforce the technological constraint that no machine should be left empty, if after the application of any of the two proposed heuristics, any machine is left empty, we reassign to it the job with the smallest processing time at that machine. The two proposed heuristics are applied to the previous example in the next section for further clarification.

5.4.2 Heuristic application example

The example of Table 5.1 is used to detail heuristic PJ_1 first. Products are first sorted according to shortest processing assembly times so $\pi = \{1, 3, 2\}$. In the second step, following the product order in π, first we assign jobs of product 1, to factories through the job to machine-factory assignment procedure. $N_1 = \{2, 7, 8\}$ so we first take job 2. The earliest completion time of this first job in all machines of all factories is 3. For job 7 is 2 (considering that it can only be assigned to factory 2) and for job 8 is 1 and can only be assigned to factory 1. The minimum is 1, which corresponds to the assignment of job 8 to the second machine of factory 1. Note that if there is a tie in the minimum completion time for the jobs, it is broken by taking the first job. We now have to consider the unscheduled jobs 2 and 7. We now calculate the earliest completion times of these two jobs at all machines of all eligible factories considering that job 8 is already assigned. These minimum completion times are 3 and 2 for jobs 2 and 7, respectively. Therefore job 7 is scheduled at factory 2 (the only eligible for this job) and to machine 2. Lastly, job 2 is scheduled with the earliest completion time of 3 at factory 1, machine 1. Note that we could have assigned this job to machine 1 of factory 2 with the same completion time, so we break ties by assigning jobs to the first machine and factory with equal completion time. After this procedure $\pi_1 = \{2, 8; 0, 7\}$. Following the same process, the jobs in the assembly programs of products 3 and 2 are assigned to factories one after the other, resulting in the final job sequence $\pi_T = \{2 - 12 - 3, 8 - 14 - 1 - 10; 5 - 6 - 4 - 13, 7 - 11 - 9\}$. The completion times of all jobs at the production stage are: $C_1 = 5$, $C_2 = 3$, $C_3 = 10$, $C_4 = 7$, $C_5 = 1$, $C_6 = 4$, $C_7 = 2$, $C_8 = 1$, $C_9 = 12$, $C_{10} = 13$, $C_{11} = 6$, $C_{12} = 6$, $C_{13} = 11$ and $C_{14} = 4$. The earliest assembling time for products 1 to 3, by considering their respective assembly programs are: $E_1 = 3$, $E_2 = 13$ and $E_3 = 12$, respectively. In the third step, the product sequence π on the assembly stage is updated by sorting E_h in ascending order, i.e., $\pi = \{1, 3, 2\}$ and the $C_{\max}$ of the application of PJ_1 to this example is 31.

For the second heuristic PJ_2 we calculate the E_h values for all products one by one with the job to machine-factory assignment procedure, the obtained sequences are $\pi_1 = \{2, 8; 0, 7\}$ with $E_1 = 3$, $\pi_2 = \{12 - 10, 1 - 3; 4, 13\}$ with $E_2 = 10$ and $\pi_3 = \{5, 14; 6, 11 - 9\}$ with $E_3 = 10$, so $\pi = \{1, 2, 3\}$. Note that there is a tie in the E_h of products 2 and 3 so again we break ties by taking the first product. Using this π we apply again the job to machine-factory assignment procedure obtaining $\pi_T = \{2-12-10, 8-1-3; 4-5-6-9, 7-13-14-11\}$ with completion times for the jobs as: $C_1 = 2$, $C_2 = 3$, $C_3 = 7$, $C_4 = 3$, $C_5 = 4$, $C_6 = 7$, $C_7 = 2$, $C_8 = 1$, $C_9 = 16$, $C_{10} = 13$, $C_{11} = 10$, $C_{12} = 6$, $C_{13} = 3$ and $C_{14} = 6$. In the third step, again products are sorted in increasing order of their respective E_h which are $E_1 = 3$, $E_2 = 13$ and $E_3 = 16$. Therefore, the updated product sequence for the assembly stage is $\pi = \{1, 2, 3\}$ with a makespan of 32.

5.5 Computational evaluation

To test the proposed MILP model and constructive heuristics, six complete sets of instances have been generated. We consider different number of problem characteristics to comprehensively evaluate and test the proposed approaches: Number of jobs (n), number of machines (m), number of factories (f) and number of products (t) are four controlled instance factors. Depending on the chosen values we have small, medium and large-sized instances, referred to as GA, GB and GC, respectively. The processing times of the jobs on each machine in the production stage, are generated following a random uniform distribution in the range $[1, 99]$, as it is common in the scheduling literature. The last instance factor we consider is the distribution of the assembly processing times which are fixed as: $U[|N_h|, 49 \times |N_h|]$ and $U[|N_h|, 99 \times |N_h|]$. These two distributions are referred to in short as 50, and 100, respectively. The final sets of instances are then denoted as $GA_{50}, GA_{100}, \ldots, GC_{100}$. For each combination of instance factors we have five replications. The combinations for each instance size are given in Table 5.2.

Instance factor	Symbol	Values		
		GA	GB	GC
Number of jobs	n	10, 12, 14, 16, 18	20, 22, 24	200, 300, 400
Number of machines	m	2, 3	2, 3, 4	5, 10, 15
Number of factories	f	2, 3	2, 3, 4	4, 6, 8
Number of products	t	2, 3, 4	2, 3, 4	20, 30, 40

Table 5.2: Instance and factors for proposed instances.

Therefore, the total number of instances is 300 for GA_{50} and another 300 for GA_{100} and 405 for every set in GB_{50} through GC_{100} resulting in a grand total of 2220 instances.

5.5.1 MILP model evaluation

The proposed MILP model is tested only on sets GA and GB given the impossibility to solve large instances. Two state-of-the-art commercial solvers are used, namely CPLEX 12.6 and GUROBI 5.6.3, which are, at the time of the writing of this thesis, the latest versions available. Two different stopping times are tested with each solver: 900 and 3600 seconds. In total we have obtained 5640 results. All tests are performed in a high performance computing cluster with 30 blades, each one containing 16 GBytes of RAM memory and two Intel XEON E5420 processors running at 2.5 GHz. The 30 blade servers are used only to divide the workload since experiments are performed in virtualized Windows XP machines, each one with a virtualized processor with two cores and 2 GB of RAM memory. Therefore, since both CPLEX and GUROBI are parallel solvers, the two available threads at each virtual machine are used.

After solving the models with CPLEX and GUROBI, three possible outcomes are obtained. The first type is “optimal”, which means that an optimal solution with a given makespan value was obtained in the given maximum CPU time. The second type is “nonoptimal”, meaning that a feasible integer solution was obtained within the time limit but it was not possible to demonstrate its optimality and the gap is reported. The third and last outcome is “out of memory”, by which the solver had an error and ran out of RAM memory, reporting a solution and a gap calculated with respect to the best obtained solution for that instance. In general, the solvers were able to find 294 (98.00%) and 297 (99.00%) optimal solutions in sets GA_{50} and GA_{100}, respectively. For GB_{50} and GB_{100} the numbers are 338 (83.45%) and 363 (89.63%) for the 405 instances, respectively. The summarized results, according to the instance factors, type of solver and time limit, are presented in Table 5.3 for sets GA and GB. The reported values at the tables are the percentage of optimum solutions found ($\%opt$), the percentage of cases with out of memory error ($\%outm$), the average gap for non-optimal solution ($GAP\%$) and the average CPU time in seconds ($AvTime$).

Solver	Time Limit	900s				3600s			
	Instance set	GA_{50}	GA_{100}	GB_{50}	GB_{100}	GA_{50}	GA_{100}	GB_{50}	GB_{100}
CPLEX	% opt	96.67	98.00	79.50	87.40	97.00	98.33	81.72	88.39
	% outm	0.00	0.00	2.46	1.72	0.00	0.00	12.34	6.41
	GAP%	0.18	0.06	0.55	0.23	0.15	0.05	0.32	0.07
	Av Time (sec.)	48.92	28.37	201.02	133.95	133.28	79.04	391.35	286.12
GUROBI	% opt	95.67	98.00	74.56	81.97	97.00	98.67	77.03	84.44
	GAP%	0.29	0.07	1.15	0.51	0.21	0.05	0.82	0.37
	Av Time (sec.)	61.08	36.06	292.75	221.95	159.21	83.35	932.96	658.33

Table 5.3: Performance results for solvers and time limit for instance sets of GA_{50}, GA_{100}, GB_{50} and GB_{100}.

As we can see, the effect of the distribution of the assembly times at the assembly stage is much stronger than either the type of solver or CPU time limit. For group GA, instances with more disperse assembly times are easier to solve and also need less CPU time. As regards the comparison between CPLEX and GUROBI, for set GA we see comparable performance with slightly shorter CPU times for CPLEX. For instance sets GB the differences between solvers are stronger. We see that GUROBI is much slower than CPLEX and has higher gap values. However, CPLEX reports out of memory errors that in some cases average more than 12% (GB_{50}). So it is important to conclude that there is no clear winner for this problem between these two solvers. In total, the largest tested instances in sets GB have 24 jobs and 16 machines distributed in 4 factories so we can attest that the proposed mathematical model has an adequate performance.

5.5.2 Heuristics evaluation

The two proposed heuristics, PJ_1 and PJ_2, are now tested. The response variable is the Relative Percentage Deviation (RPD), measured as:

$$RPD = \frac{Alg_{sol} - Best_{sol}}{Best_{sol}} \times 100$$

Where $Best_{sol}$ is the best makespan obtained after all experimentation in this chapter for any instance and Alg_{sol} is the makespan obtained by the heuristic. The heuristics are coded in C# and are compiled under Visual Studio 2010. The same computing platform used for the MILP evaluation is employed here. The average RPD values for the proposed heuristics are given in Tables 5.4, 5.5 and 5.6 for instances sets GA, GB and GC, respectively. All results are grouped by n and f. The average RPD values of CPLEX and GUROBI are reported as well for reference.

As can be observed, PJ_2 is generally much better than PJ_1 in all groups of instances, although the difference is not very big in the large instances.

	GA_{50}				GA_{100}			
$f \times n$	CPLEX	GUROBI	PJ_1	PJ_2	CPLEX	GUROBI	PJ_1	PJ_2
2×10	0.00	0.00	9.52	3.26	0.00	0.00	2.21	0.46
2×12	0.00	0.00	8.24	4.04	0.00	0.00	3.58	1.76
2×14	0.00	0.00	8.29	3.36	0.00	0.00	4.50	0.62
2×16	0.00	0.00	9.31	4.59	0.00	0.00	4.30	1.16
2×18	0.00	0.21	7.27	3.90	0.00	0.00	3.34	1.23
3×10	0.00	0.00	5.10	2.59	0.00	0.00	1.09	1.24
3×12	0.00	0.00	4.72	1.43	0.00	0.00	1.85	0.98
3×14	0.00	0.00	4.51	1.71	0.00	0.00	1.51	0.28
3×16	0.00	0.00	4.79	1.39	0.00	0.00	2.63	0.72
3×18	0.00	0.00	4.04	1.34	0.00	0.00	2.15	0.78
Average	0.00	0.02	6.58	2.76	0.00	0.00	2.72	0.92

Table 5.4: Average Relative Percentage Deviation (RPD) of CPLEX, GUROBI and the proposed heuristics for instance sets GA_{50} and GA_{100}.

It is important to observe how in the largest instances in set GB of 24 jobs and 4 factories, PJ_2, gives a very small gap of just 0.35% which indicates that PJ_2 is a very capable heuristic with close to optimality performance. On average, PJ_2 is below 1% RPD for instance groups GA and GB. For the large instances in GC it is not possible to calculate the optimum solution so we only have an overall picture were PJ_2 always obtains the best solution. As a matter of fact and although not reported in detail here, among the 810 instances in GC_{50} and GC_{100}, PJ_2 is always better or equal than PJ_1.

We report now on the CPU times of the proposed heuristics in Table 5.7. It has to be noted that CPU times are negligible, on the verge of being below the margin of error in measurements.

As can be seen, the average CPU times are below one tenth of a second for the largest instances in group GC. On average, PJ_2 is relatively slower than PJ_1 but on absolute terms the CPU times are very small. Although not shown

	GB_{50}				GB_{100}			
$f \times n$	CPLEX	GUROBI	PJ_1	PJ_2	CPLEX	GUROBI	PJ_1	PJ_2
2×20	0.20	0.00	7.03	2.33	0.00	0.00	3.33	1.03
2×22	0.29	0.03	5.26	2.36	0.05	0.03	2.36	0.87
2×24	0.13	0.19	4.78	1.89	0.11	0.04	3.33	1.40
3×20	0.00	0.00	3.21	1.42	0.00	0.00	2.48	1.27
3×22	0.01	0.01	2.89	1.69	0.04	0.02	1.54	0.47
3×24	0.10	0.02	3.17	1.25	0.00	0.02	1.38	0.70
4×20	0.00	0.00	2.31	1.29	0.00	0.00	1.83	0.67
4×22	0.00	0.00	2.18	0.77	0.00	0.00	1.78	0.75
4×24	0.00	0.00	2.50	1.12	0.00	0.00	1.82	0.35
Average	0.08	0.03	3.70	1.57	0.02	0.01	2.21	0.84

Table 5.5: Average Relative Percentage Deviation (RPD) of CPLEX, GUROBI and the proposed heuristics for instance sets GB_{50} and GB_{100}.

here, the largest measured CPU time corresponds to heuristic PJ_2 has been 0.41 seconds. From this final evaluation and considering the relative RPD of PJ_2 we can conclude that it is a capable and very fast heuristic.

Even though the observed differences are large in all cases for the proposed heuristics and very small for the two solvers, we carry out some statistical analyzes in order to ascertain if the observed differences are indeed statistically significant. All results are examined with the Analysis of Variance (ANOVA) technique. ANOVA is a powerful parametric tool, which has been used in the last 10 years in the scheduling literature with great success. For the small instances there is no statistically significant difference in the performance of CPLEX and GUROBI and PJ_2 is statistically better than PJ_1. The detailed data is not reported for space reasons. For the medium sized-instances in set GB we observe the interaction between the distribution of the assembly processing times and tested methods in Figure 5.1.

	GC_{50}		GC_{100}	
$f \times n$	PJ_1	PJ_2	PJ_1	PJ_2
4×200	0.13	0.00	0.07	0.00
4×300	0.12	0.00	0.05	0.00
4×400	0.11	0.00	0.06	0.00
6×200	0.09	0.00	0.04	0.00
6×300	0.08	0.00	0.05	0.00
6×400	0.08	0.00	0.04	0.00
8×200	0.06	0.00	0.03	0.00
8×300	0.08	0.00	0.03	0.00
8×400	0.05	0.00	0.04	0.00
Average	0.09	0.00	0.05	0.00

Table 5.6: Average Relative Percentage Deviation (RPD) of the proposed heuristics for instance sets GC_{50} and GC_{100}.

GA_{50}		GA_{100}		GB_{50}		GB_{100}		GC_{50}		GC_{100}	
PJ_1	PJ_2	PJ_1	PJ_2	PJ_1	PJ_2	PJ_1	PJ_2	PJ_1	PJ_2	PJ_1	PJ_2
0.0031	0.0055	0.0049	0.0060	0.0070	0.0069	0.0070	0.0067	0.05	0.09	0.05	0.09

Table 5.7: Heuristic's CPU time (in seconds) for all instance groups.

As can be seen, the results are similar to those of set GA. The differences between the proposed heuristics are large enough so as to be statistically significant whereas the differences in the performance of the solvers are not statistically relevant. As for the large instances in group GC we can only test the significance in the observed differences in the average RPD between the two heuristics. This is given in Figure 5.2.

As can be observed, PJ_2 is statistically better than PJ_1 even though the absolute difference between both proposed methods is practically small.

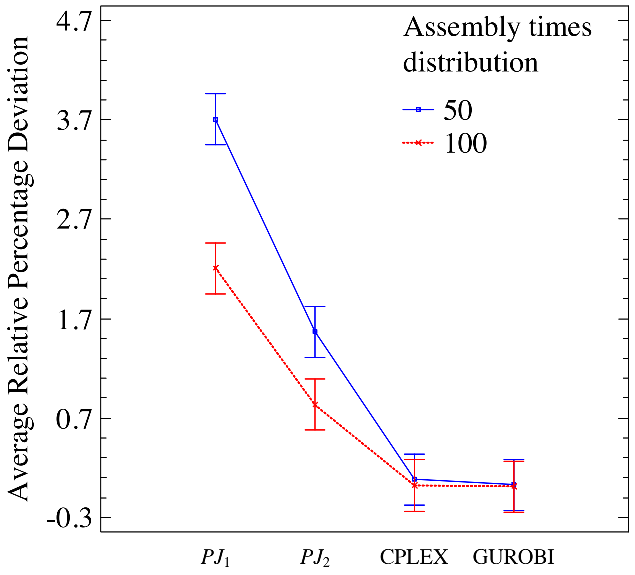


Figure 5.1: Means plot with the interaction between the distribution of the assembly processing times and the tested methods for instances GB. All means have Tukey's Honest Significant Difference (HSD) 95% confidence intervals.

5.6 Conclusions of the chapter

In this chapter we have studied an interesting combination of a distributed manufacturing problem with assembly operations. More specifically, we have presented a distributed unrelated parallel machines problem by which a number of factories, each one containing unrelated parallel machines have to manufacture jobs. All these jobs are later assembled into products in a factory with a single assembly machine. The objective is to minimize the makespan in the assembly stage. Such a problem has been motivated and shown not to have been studied to date. We have presented a mathematical model and two constructive heuristics. The mathematical model has been comprehensively evaluated and tested using two state-of-the-art commercial solvers. Results

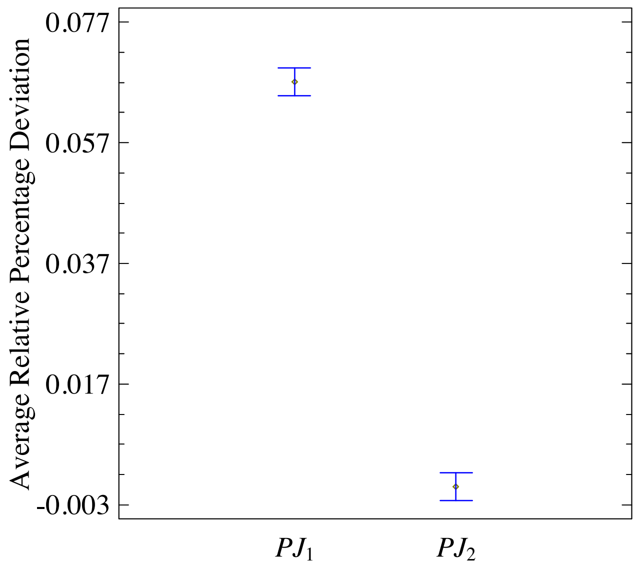


Figure 5.2: Means plot for the two heuristics in large instances (GC). All means have Tukey's Honest Significant Difference (HSD) 95% confidence intervals.

have shown that we are able to solve optimally problems of up to 24 jobs and 16 machines distributed in 4 factories. The two proposed heuristics are inherently simple and at the same time report solutions very close to optimal in the cases for which the optimal solution has been obtained. Furthermore, for large instances, the performance is very good, obtaining solutions in less than half a second.

CHAPTER 6

HEURISTICS FOR A DISTRIBUTED PARALLEL MACHINE AND ASSEMBLY SCHEDULING PROBLEM WITH ELIGIBILITY CONSTRAINTS

“The contents of this chapter are submitted to International Conference on Industrial Engineering and System Management (IESM 2015), Seville, Spain.”

In this chapter we study a production scheduling problem with production and assembly stages. There is a set of distributed identical factories, each one with a set of unrelated parallel machines at the production stage and a single assembly machine in the assembly stage. Jobs have to be assigned to one of the distributed factories and processed by one of the unrelated parallel machines. Processed jobs are assembled into final products through a defined assembly program in the assembly stage. This problem is referred to as the Distributed Parallel Machine and Assembly Scheduling Problem or DPMASP. Minimizing the makespan of the products in the assembly stage is considered as the objective. Because of technological constraints, some factories are bit able to process some jobs and empty machines at factories are permitted. We present a mathematical model, four simple, fast and high performing heuristics to solve the considered problem. CPLEX and GUROBI as two state-of-the-art commercial solvers are used to solve the mathematical model. Comprehensive computational experiments and ANOVA statistical analyses are performed to evaluate the performance of the proposed mathematical model and heuristics. Our results show that the mathematical model is able to solve moderately-sized instances and some of the heuristics report solutions that are very close to optimality in negligible CPU times.

6.1 Introduction

Today, the manufacturing industry faces numerous challenges and is beleaguered by obstacles in the marketplace. Some of the significant challenges are: intense global competition, technological changes, product life cycles reduction, increasing product variety, demand of customized goods instead of mass production, consumer needs diversity, faster delivery, higher products quality, cost pressures, uncertain and dynamic global market, etc. In order to overcome these challenges, the consideration of efficient production strategies is necessary and essential. One of these strategies is to employ distributed manufacturing environments able to produce the variety of products that the customer demands. Presenting diversified product offerings is one of key advantages for companies to compete in the unpredictable global market. On the other hand, scheduling and optimizing such systems is a complex task which directly affects production performance.

Assembly systems have attracted the attention of practitioners and researchers. Different manufacturing systems have adopted them in their production structure to meet a high variety of customer demands through an increase in the flexibility and capability level of the system. Various components are produced through different independent operations in parallel. These components are later assembled into finished products in assembly lines. Assembly systems are capable of producing a high variety of finished products by assembling different combinations of produced components. These types of manufacturing settings are referred to as Assembly Scheduling Problems (ASP). Tozkapan et al. (2003) presented a two-stage assembly scheduling problem with the objective function of minimizing the total weighted flow time. Later, Al-Anzi and Allahverdi (2006) addressed the model presented by Tozkapan et al. (2003) minimizing the total completion time of all the jobs and proposed metaheuristics to solve it. A small extract of the many existing papers in this regard are presented here.

Distributed Manufacturing Systems (DMS ,Peklenik (1992)) consist of several factories with different machines and tools at different geographical locations. It is one of the effective approaches to improve the levels of flexibility, reconfigurability and productivity of the manufacturing systems and to remove the traditional manufacturing systems weakness and bottlenecks in order to meet the aforementioned challenges (Sluga et al. (1998)). Different research results (e.g., Wang (1997); Kahn et al. (2004); Chan et al. (2005b); Mahdavi et al. (2008)) have shown that DMS achieve better product quality, lower production costs and reduced management risks for the system. From the viewpoint of the manager, scheduling in DMS is more complicated than the scheduling of a single manufacturing factory. In DMS, the first decision that has to be made is to select a factory for each job and then determine a job schedule for each factory, while for a single manufacturing system only a job schedule for each set of machines has to be determined.

A Distributed Permutation Flowshop Problem (DPFSP) for the production stage was first proposed by Naderi and Ruiz (2010) and has been thoroughly studied thereafter, with recent results in Naderi and Ruiz (2014) and in Fernandez-Viagas and Framinan (2015). Hatami et al. (2013) studied the Distributed Assembly Permutation Flowshop Scheduling Problem (DAPFSP) for the first time. In the DAPFSP, the first stage is a distributed flowshop and the second stage is a single assembly machine. Later, Hatami et al. (2015) replaced the flowshop figuration in the production stage with unrelated parallel machines and presented the Distributed Parallel Machine and Assembly Scheduling Problem (DPMASP). In this chapter, we extend the DPMASP presented by Hatami et al. (2015) by relaxing the special constraint that no machine at any factory might be empty due to technological or economical constraints. Furthermore, additional heuristics are presented for the studied problem.

In our considered model, a set N of n jobs has to be assigned to a set F of f identical factories. Each factory consists of a set M of m unrelated parallel machines. Each job has to be assigned and processed at exactly one machine at one factory. Some of the jobs have eligibility constraints as regards some factories. LF_j is a subset of set F with factories where job j can be assigned to where $f \geq |LF_j| \geq 1, j = 1, \dots, n$. A set T of t independent products, formed by a defined number of jobs, have to be assembled at the single assembly machine on the assembly stage M_A. Product $h, h = 1, \dots, t$ is made by assembling the subset of jobs $N_h \subseteq N$ which are produced at the distributed factories. Each job belongs to a single product, i.e., $\sum_{h=1}^{t} |N_h| = n$. The earliest time to start the assembly of product h is when all jobs in N_h have been produced and completed on the production stage. For the sake of simplicity, all factories have the same number of unrelated parallel machines. The processing time of job j at machine k of any factory and the assembly time of product t on the single assembly machine are denoted by p_{jk} and p_h respectively. These processing times are known, deterministic and positive. Jobs have to be assigned to factories and then again assigned to and scheduled on the parallel machines of the distributed factories and the products have to be scheduled on the single assembly machine. The objective is to minimize the makespan on the assembly stage. The DPMASP is an extension of the identical parallel machines problem. According to the results of Lenstra et al. (1977), the identical parallel machines problem, even in the case of two identical machines (and a single factory), i.e., the $P2//C_{\max}$ problem, is $\mathcal{NP}$-Hard. Therefore, the computational complexity of the DPMASP when

$n >> f$ is $\mathcal{NP}$-Hard as well.

The chapter is organized as follows: A Mixed Integer Linear Programming (MILP) model is presented for the considered problem in Section 6.2. Four simple and high performance heuristics are proposed to obtain solutions in much shorter CPU times are detailed in Section 6.3. In section 6.4 a comprehensive computational and statistical experiment is carried out to evaluate the proposed MILP and heuristics. Finally, Section 6.5 provides some final conclusion for the problem considered.

6.2 Mixed Integer Linear Programming model

A mathematical model for the proposed DPMASP is presented. To better understand this mathematical model, the definition of the used indexes, pa-

		Description
index	i, j	denotes jobs, $i, j = 0, 1, \ldots, n$, where 0 presents a dummy job
	k	denotes machines, $k = 1, \ldots, m$
	q	denotes factories, $q = 1, \ldots, f$
	l, s	denotes products, $l, s = 0, 1, \ldots, t$, where 0 presents a dummy product
	M	A sufficiently large positive number, $M = 100000$
Parameters	n	number of jobs
	m	number of machines
	f	number of factories
	t	number of products
	p_{jk}	processing time of job j on machine k
	p_s	processing time of product s at the assembly stage
	G_{js}	Binary parameter equal to 1 if job j belongs to product s, and 0 otherwise
Variable	X_{ijkq}	binary variable equal to 1 if job i is an immediate predecessor of job j on machine k in factory q
	Y_{ls}	binary variable equal to 1 if product l is an immediate predecessor of product s at the assembly machine
	C_j	completion time of job j at the production stage
	CA_s	completion time of product s on assembly stage
	$C_{\max}$	makespan

Table 6.1: indices, parameters and variables used in MILP mathematical model.

rameters and variables is given in Table 6.1.

The model minimizes the makespan ($C_{\max}$) as the objective function and the model is subject to the following constraints:

$$\sum_{\substack{i=0, i\neq j}}^{n} \sum_{k=1}^{m} \sum_{\substack{q=1\\ q\in LF_j\\ q\in LF_i}}^{f} X_{ijkq} = 1 \qquad \forall j \tag{6.1}$$

$$\sum_{\substack{j=0, j\neq i}}^{n} \sum_{k=1}^{m} \sum_{\substack{q=1\\ q\in LF_j\\ q\in LF_i}}^{f} X_{ijkq} = 1 \qquad \forall i \tag{6.2}$$

$$\sum_{\substack{j=1\\ q\in LF_j}}^{n} X_{0jkq} \leq 1 \qquad \forall k, q \tag{6.3}$$

$$\sum_{\substack{i=1\\ q\in LF_i}}^{n} X_{i0kq} \leq 1 \qquad \forall k, q \tag{6.4}$$

$$\sum_{\substack{j=1, j\neq i\\ q\in LF_j}}^{n} (X_{ijkq} - X_{jikq}) = 0 \qquad \forall i, k, q, q \in LF_i \tag{6.5}$$

$$\sum_{k=1}^{m} \sum_{\substack{q=1\\ q\in LF_i\\ q\in LF_j}}^{f} X_{ijkq} + \sum_{k=1}^{m} \sum_{\substack{q=1\\ q\in LF_i\\ q\in LF_j}}^{f} X_{jikq} \leq 1 \qquad \forall i \in \{1, \dots, n-1\}, j > i \tag{6.6}$$

$$C_j \geq C_i + p_{jk} + M(1 - X_{ijkq}) \qquad \forall i, j, k, \\ \forall q, q \in LF_i, LF_j \tag{6.7}$$

$$\sum_{l=0, l \neq s}^{t} Y_{ls} = 1 \qquad \forall s \tag{6.8}$$

$$\sum_{s=1, l \neq s}^{t} Y_{ls} \leq 1 \qquad \forall l \tag{6.9}$$

$$Y_{ls} + Y_{sl} \leq 1 \qquad \forall l \in \{1, \ldots, t-1\}, s > l \tag{6.10}$$

$$CA_s \geq (C_j \cdot G_{js}) + p_s \qquad \forall j, s \tag{6.11}$$

$$CA_s \geq CA_l + p_s + M(1 - Y_{ls}) \qquad \forall l, s, l \neq s \tag{6.12}$$

$$C_{\max} \geq CA_s \qquad \forall s \tag{6.13}$$

$$X_{ijkq} \in \{0, 1\} \qquad \forall i, j, k, q, i \neq j, q \in LF_i, q \in LF_j \tag{6.14}$$

$$Y_{ls} \in \{0, 1\} \qquad \forall l, s, l \neq s \tag{6.15}$$

$$C_j \geq 0 \qquad \forall j \tag{6.16}$$

$$CA_s \geq 0 \qquad \forall s \tag{6.17}$$

It should be considered that, variables C_0 and CA_0 are equal to 0. Having exactly one predecessor and successor for each job is ensured by sets (6.1) and (6.2), respectively. Constraint sets (6.3) and (6.4) force that each machine at each factory has a dummy job 0 as predecessor and successor, respectively. Note that it is possible that one or more machines end up with no jobs assigned. Constraint set (6.5) ensures that if a job is sequenced on a machine, then its predecessor and successor must be processed on the same machine. A job cannot be both a predecessor and successor of another job at the same time and this is controlled by constraint set (6.6). Constraint set (6.7) ensures that if job j is placed immediately after job i, job j can start processing at machine k only when job i is finished at machine k. Each product should have one preceding and at most one succeeding product in the assembly machine according to constraint set (6.8) and (6.9), respectively. Constraint set (6.10) controls that is not allowed for a product to be being both a predecessor and a successor of another product at the same time in the assembly machine. Constraint set (6.11) forces that each product h can start its assembly on the assembly stage only after all jobs belonging to it are completed at the production stage. Constraint set (6.12) determines that if product s is placed immediately after product l, it has to wait until the assembly of product l in the assembly machine finishes. The makespan and the domain of the decision variables are defined in constraint set (6.13) and from set (6.14) through (6.17).

6.3 Constructive heuristics

As mentioned in Section 6.1, the DPMASP is an $\mathcal{NP}$-Hard problem if $n >> f$ (which is, by the way, the most usual case as it makes little sense to have more factories than jobs to produce). Therefore, the design and development of heuristic methods to solve large-sized problems is usually unavoidable. To better explain the proposed heuristics an example with ten jobs ($n = 10$), two products ($t = 2$), three factories ($f = 3$) and two unrelated parallel machines in each factory ($m = 2$) is given. The ten jobs have to be assembled into the two products with the following program $N_1 = \{3, 5, 6, 7, 10\}$ and $N_2 = \{1, 2, 4, 8, 9\}$. Therefore, in order to assemble product 1 the processing of jobs 3, 5, 6, 7 and 10 must be finished at the production stage. Table 6.2 details the job processing times on each machine at the production stage and eligibility constraints. The assembly processing times for products 1 and 2 are 6 and 14, respectively.

Some additional notation is now given. A possible sequence of products on the assembly machine is referred to as a *product sequence* and is represented by π, e.g., $\pi = \{2, 1\}$. The jobs belonging to product h have to be assigned to factories and inside each factory, to the unrelated machines in the production

		Job									
		J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
Machine	M_1	1	2	2	1	6	9	1	3	1	5
	M_2	5	3	1	3	3	4	2	1	3	6
	LF_j	2	1,2	1,2	1,3	3	1,2,3	1,2	2	2	1

Table 6.2: Job processing times and factory eligibility constraints for the example.

stage. Furthermore, the job sequence at each unrelated machine is needed. We will propose some job to machine-factory assignment procedures. The sequence of the jobs of product h on machines at a given factory is referred to as a *job to machine-factory sequence for product* h and represented by π_h. For the example one possibility is $\pi_1 = \{0, 3 - 10; 0, 7; 5 - 6, 0\}$ and $\pi_2 = \{0, 0; 2 - 9 - 8, 1; 4, 0\}$. The sequence of the jobs for each factory is separated by ";" and for each factory, the job sequence for each machine is separated by ",". The sequence of the jobs assigned to a factory and machine is separated by "-". Any machine without jobs assigned is indicated by "0". In the given example, π_1 indicates that the first machine of the first factory is empty of any jobs belonging to product 1. Jobs 3 and 10 are assigned, in this sequence, to the second machine of the first factory. The first machine of the second factory is empty and job 7 is scheduled on the second machine of the second factory. Finally, jobs 5 and 6 are scheduled to the first machine of the third factory and the second machine of this last factory is empty. The final job sequence after assigning all jobs of the products to the factories is denoted as the *final job sequence* and referred to as π_T, e.g., $\pi_T = \{0, 3-10; 2-9-8, 7-1; 5-6-4, 0\}$. Note that π_T concatenates the different job to machine-factory sequences of all products (π_h) following the product sequence π. As mentioned before, in the considered problem there is a restriction of job assignment to factories. The number of factories to which job j cannot be assigned is referred to as *number of prohibited factories for job j* and noted by NP_j, e.g., $NP_1 = 2, NP_3 = 1, NP_6 = 0$ in the example. The earliest time at which the assembly of a product in the assembly stage can start, which corresponds to the maximum completion time of all the jobs in the production stage that belong to this product is referred to as the *product h earliest assembling time* and denoted by E_h.

In this chapter, different methods are proposed to construct the *product sequence*, π and the *job to machine-factory sequence for product h*, π_h. In the following sections, these methods are explained in detail.

6.3.1 Product sequence construction

Two different methods are presented to build a product sequence. This sequence determines the sequence of the products on the single assembly machine at the assembly stage.

- The first method, referred to as PS_1 and constructs the product sequence based on the Shortest Processing Time heuristic (SPT). Products are sorted in increasing order of their assembly times. This heuristic reduces the average number of jobs in the system, in-process inventories and average job tardiness (Stafford et al. (2005)).

- The second method is referred to as PS_2 and constructs the product sequence by sorting products in ascending order of their earliest assembling times (E_h).

6.3.2 Job to machine-factory sequence for products construction

To construct the job to machine-factory sequence for product h, all jobs belonging to product h need to be considered. Two construction procedures are considered.

- The first procedure is referred to as JA_1. We denote by U_h to the set of unscheduled jobs of product h, i.e., those jobs not yet assigned to factories. The set of scheduled jobs at machine k inside factory q is referred by S_{kq}. Among all unscheduled jobs belonging to product h, the job with the earliest completion time at any machine in every eligible factory is assigned first. More specifically, we assign job $j^* \in U_h$ to machine k^* at factory q^* satisfying:

$$\{j^*, k^*, q^*\} = \underset{k \in m, q \in LF_j, j \in U_h}{\arg\min} \left\{ \sum_{i \in S_{kq}} p_{ik} + p_{jk} \right\}$$

The process is repeated until all jobs in U_h are scheduled.

- In the second procedure, jobs belonging to product h are divided into two sets. The first set, referred to as R_h, contains the unscheduled jobs of product h with one or more prohibited factories ($NP_j > 0$). These jobs are sorted in descending order of NP_j, $j \in N_h, NP_j > 0$. Jobs in set R_h are further divided into x subsets. The first subset contains the jobs with the largest number of prohibited factories ($L_1 R_h$) and the last subset of jobs has the least number of prohibited factories ($L_x N_h$). The second set Z_h contains the unscheduled jobs of product h with NP_j equal to zero. There are two steps in this procedure for job to machine-factory assignment. In the first step, all jobs belonging to the first subset of set R_h, ($L_1 R_h$), are assigned to the factories using the previous JA_1 procedure. After assigning all jobs in $L_1 R_h$, all jobs belonging to the next subset of R_h, $L_2 R_h$, are scheduled. This process is applied to all x subsets until all jobs in set R_h are assigned. In the second step, all jobs belonging to the second set Z_h are assigned to factories. This procedure is referred to as JA_2.

In this chapter we present four simple constructive heuristics. All of them consist of three main steps. The *product sequence* π is determined in the first step. In the second step, according to the order of products in π, the jobs belonging to the different products are assigned to factories and machines, using one of the job to machine-factory assignment methods, presented in section 6.3.2. Finally, in the last step, and once all jobs are assigned and completed at the production stage, the sequence of products on the assembly stage is updated by sorting the products in ascending order of their earliest assembly times E_h.

In all simple constructive heuristics, the third step is the same, but the methods to construct the product π and job to machine-factory assignment are different. These are all described in the following sections.

6.3.3 Heuristic H_1

In heuristic H_1, PS_1 is used to determine the *product sequence* π and JA_1 is the procedure employed to assign jobs to factories and machines. When all jobs are completed at the production stage, E_h is calculated for each product h. Then the product sequence π on the assembly machine is updated by sorting the products in ascending order of E_h. The makespan of the problem is finally calculated.

The previous example is used to detail heuristic H_1. First, π is determined according to the SPT rule, sorting the assembly times of the products in ascending order. The result is $\pi = \{1, 2\}$. The second step starts by assigning jobs of product 1 as the first product on π to the factories following the JA_1 procedure. The jobs of product 1 are 3, 5, 6, 7 and 10. The earliest completion time of job 3 in the first and second factories is 1 if assigned to the second machine (it can not be assigned to factory 3). The earliest completion time for job 5 is 3 in factory 3 as it is the only eligible factory for this job. For job 6, which can assigned to all factories is 4. Job 7 can be assigned to the first and second factories with an earliest completion time of 1. For job 10 which can only be assigned to the first factory is 5. The minimum is 1 which corresponds to the assignment of two jobs: job 3 to the second machine of factories 1 or 2 and job 7 to the first machine of factories 1 or 2. Note that in the cases with ties and more than one possibility, ties are broken by assigning the first job to the first machine and factory with equal completion time. Therefore, job 3 is selected and assigned to the second machine of factory 1. Now we have to assign the remaining unscheduled jobs 5, 6, 7 and 10. The earliest calculated completion time are calculated considering that job 3 is already assigned. These minimum completion times are 3, 4, 1 and 5 for jobs 5, 6, 7 and 10, respectively. The minimum is 1 and corresponds to the assignment of job 7 to first machine of factories 1 and 2. So, we break ties by assigning job 7 to first machine at factory 1. The process continues for all unscheduled jobs 5, 6 and 10. The third scheduled job is 5 which is assigned to the second machine of factory 3 with the earliest completion time of 3. The forth job is job 6 with the minimum earliest time of 4 which is scheduled on the second machine of factory 2. Lastly, job 10 is scheduled with the earliest completion time of 6 at factory 1, machine 1. After applying the JA_1 method for assigning the jobs of product 1, $\pi_1 = \{7-10, 3; 0, 6; 0, 5\}$. Using the same procedure, jobs belonging to product 2 are assigned to factories, resulting in the final job sequence $\pi_T = \{7-10, 3-2; 1-9-8, 6; 4, 5\}$. The completion time of all jobs at the production stage are: $C_1 = 1$, $C_2 = 4$, $C_3 = 1$, $C_4 = 1$, $C_5 = 3$, $C_6 = 4$, $C_7 = 1$, $C_8 = 5$,$C_9 = 2$, $C_{10} = 6$. According to the assembly program of each product, the earliest assembling time for products are: $E_1 = 6$, $E_2 = 5$. In the third step the *product sequence* π on the assembly stage is updated by sorting the obtained earliest assembly time of products E_h in ascending order. This results in $\pi = \{2, 1\}$. The $C_{\max}$ after applying the proposed H_1 heuristic for this example is 25. The Gantt chart of the considered example is shown in Figure 6.1.

6.3.4 Heuristic H_2

In heuristic H_2, the first and third steps are the same as heuristic H_1, and JA_2 is employed as job to machine-factory assignment method in the second step. The obtained product order in the first step is $\pi = \{1, 2\}$. Therefore two sets R_1 and Z_1 are determined for the jobs of product 1, $R_1 = \{5, 10, 3, 7\}$ and $Z_1 = \{6\}$. Set R_1 is sorted in descending order of the prohibited number of factories and has two subsets ($x = 2$), $L_1R_1 = \{5, 10\}$ and

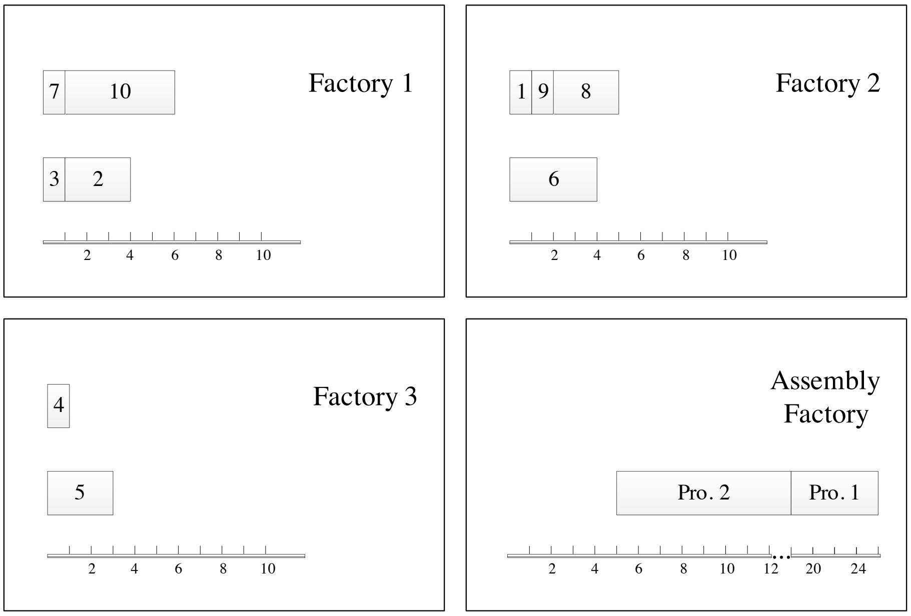


Figure 6.1: Gantt chart of H_1 for the example.

$L_2R_1 = \{3, 7\}$. First, all jobs in subset L_1R_1 with a number of prohibited factories of 2 are assigned to factories according to the JA_1. The result is $\pi_1 = \{10, 0; 0, 0; 0, 5\}$. The process continues with the assignment of the jobs of the next subset and the result is $\pi_1 = \{10, 3; 7, 0; 0, 5\}$. Now the jobs in set $Z_1 = \{6\}$ are assigned to factories and the outcome is $\pi_1 = \{10, 3; 7, 6; 0, 5\}$. Jobs of product 2 are also assigned to factories with the same procedure with two the sets being $R_2 = \{1, 8, 9, 2, 4\}$ and $Z_2 = \{\phi\}$, where R_2 is further divided into two subsets: $L_1R_2 = \{1, 8, 9\}$ and $L_2R_2 = \{2, 4\}$. This results in the *final job sequence*, $\pi_T = \{10, 3-2; 7-1-9, 6-8; 4, 5\}$. The completion times of the jobs on the production stage are: $C_1 = 2$, $C_2 = 4$, $C_3 = 1$, $C_4 = 1$, $C_5 = 3$, $C_6 = 4$, $C_7 = 1$, $C_8 = 5$,$C_9 = 3$ and $C_{10} = 5$. Therefore, the earliest assembling time of the products considering the assembly program are: $E_1 = 5$ and $E_2 = 5$. In the third step the *product sequence* π on the assembly machine is updated by sorting the obtained E_h in ascending order, i.e., we obtain $\pi = \{1, 2\}$. The final $C_{\max}$ for the example using the second proposed heuristic H_2 is also 25.

6.3.5 Heuristic H_3

Different from the first and second proposed heuristic, the third heuristic H_3 uses the PS_2 method in the first step to construct the *product sequence* π. The earliest assembling time of the products are sorted in ascending order to build π. In order to do so, the earliest assembling times of each product (E_h) are calculated after assigning the jobs of each product to factories using the JA_1 method. This calculation is carried out for each product, separately. In the second step, the jobs of the products according to the product order π are assigned to factories, using again the JA_1 method. The third updates the *product sequence* in the same way as in the previous two heuristics.

Following the example, for the first step, and to determine the *product sequence* π, the jobs of the products 1 and 2 are assigned to factories according to JA_1 method, separately. The details of this assignment are explained in Section 6.3.3. The obtained job to machine-factory sequence for product 1 is $\pi_1 = \{7 - 10, 3; 0, 6; 0, 5\}$. Now the jobs of product 2 are assigned to the factories with the same procedure which result in $\pi_2 = \{4-2, 0; 1-9, 8; 0, 0\}$. The earliest assembling times of products 1 and 2 are 6 and 3, respectively. Therefore, the *product sequence* is $\pi = \{2, 1\}$. In the second step, the jobs of the products are assigned to factories through the JA_1 method following the product order in π, which results in the *final job sequence* $\pi_T = \{4 - 2 - 10, 3 - 7; 1 - 9, 8 - 6; 0, 5\}$. The completion times of the jobs are: $C_1 = 1$, $C_2 = 3$, $C_3 = 1$, $C_4 = 1$, $C_5 = 3$, $C_6 = 5$, $C_7 = 3$, $C_8 = 1$,$C_9 = 2$ and $C_{10} = 8$. In the last step, the earliest assembling times of the products are obtained by considering the completion times of the jobs as $E_1 = 8$ and $E_2 = 3$. The *product sequence* is updated by sorting E_h in ascending order, giving $\pi = \{2, 1\}$ and the $C_{\max}$ for the example in the case of heuristic H_3 is equal to 23.

6.3.6 Heuristic H_4

In the last proposed heuristic H_4, the PS_2 and JA_2 methods are employed to construct the *product sequence* and to carry out the job to machine-factory assignment in the first and second steps, respectively. Third step is the same as in the other heuristics.

In the first step and to construct π, the earliest assembling times of all products (E_h) have to be calculated. Each product is considered separately and its jobs are assigned to factories using the JA_2 method. First, jobs of product 1 are assigned to factories. The details of this assignments have been already given in Section 6.3.4. The result was $\pi_1 = \{10, 3; 7, 6; 0, 5\}$. Jobs of product 2 are split into the sets $R_2 = \{1, 8, 9, 2, 4\}$ (divided into subsets $L_1R_2 = \{1, 8, 9\}$ and $L_2R_2 = \{2, 4\}$) and $Z_2 = \{\varnothing\}$. Jobs are then assigned to factories using the JA_2 method which results in $\pi_2 = \{4 - 2, 0; 1 - 9, 8; 0, 0\}$. The earliest assembling times of the products are obtained: $E_1 = 5$ and $E_2 = 3$. After this, π is constructed, resulting in $\pi = \{2, 1\}$. In the next step, according to the product order in π, jobs are assigned and scheduled to factories using the JA_2 procedure, which results in $\pi_T = \{4 - 2, 10; 1 - 9 - 7, 8 - 3 - 6; 0, 5\}$. The earliest assembling times are $E_1 = 6$ and $E_2 = 3$ and π on the assembly stage is updated by sorting E_h in ascending order: $\pi_1 = \{2, 1\}$. The $C_{\max}$ for the example after heuristic H_4 is 23.

6.4 Computational evaluation

To test and to evaluate the proposed MILP model and constructive heuristics, six complete sets of instances with different problem characteristics are generated. For each instance, there are four controlled instance factors: number of jobs (n), number of machines (m), number of factories (f) and number of products (t). Depending on the values chosen for some of theses factors, we generate three benchmarks referred to as small (GA), medium (GB) and large

(GC). For GA, n is set at five levels (10, 12, 14, 16, 18), and both m and f have two levels each with the same values (2,3) and t is set at three levels (2, 3, 4). For GB all controlled instance factors are set at three levels each; the values for n are (20, 22, 24) and m, f and t are fixed to the same values of (2,3,4). For GC the controlled instance factors n, m, f and t are set to three levels each, (200, 300, 400), (5, 10, 15), (4, 6, 8) and (20, 30, 40), respectively. The job processing times on each parallel machine in the production stage are generated from a random uniform distribution in the range $[1, 99]$, as it is common in the scheduling literature. The last considered instance factor is the distribution of product assembly times in the single machine on the assembly stage which depends on the number of jobs that to each product h and are fixed to $U[|N_h|, 49 \times |N_h|]$ and $U[|N_h|, 99 \times |N_h|]$. For simplicity, these two distributions are briefly referred to as 50 and 100. The three small, medium and large-sized sets of instances in combination with these distribution intervals are denoted as $GA_{50}, GA_{100}, \ldots, GC_{100}$. Five replications are considered for each combination of instance factors. Therefore, the total number of instances is 300 for GA_{50} and GA_{100} and 405 for every set in GB and GC which results in a grand total of 2220 instances considering all six instance sets.

6.4.1 MILP model evaluation

Initial tests proved that the MILP is unable to solve the large instances in GC. Therefore only the instance sets GA and GB are used to test the proposed MILP model. Two state-of-the-art commercial solvers, CPLEX 12.6 and GUROBI 5.6.3 are used to solved the MILPs. Two different stopping time criteria of 900 and 3600 seconds are considered with each solver. Totally, there are 5640 obtained results. All tests are performed in a high performance computing cluster with 30 blade servers, each one containing 16 GBytes of RAM memory and two Intel XEON E5420 processors running at 2.5 GHz. The 30 blade servers are used only to divide the workload since experiments are performed in virtualized Windows XP machines, each one with a single virtualized processor with two cores and 2 GBytes of RAM memory. Since both CPLEX and GUROBI are parallel solvers, the two available cores at each virtual machine are used.

Three types of possible outcomes are obtained after solving the MILPs with the solvers. In the first outcome type, named "optimal", an optimal solution is obtained within the maximum CPU time with a given makespan value. In the second type, "non-optimal", a feasible integer solution with a given makespan value within the time limit is obtained. However, this solution is not proven to be optimal and the gap with respect to the best non-integer is reported by the solver. In the last outcome "out of memory", the solver ran out of RAM memory. In this type of outcome the best solution and gap up to the error is reported. Altogether, the solvers were able to find 294 (98.00%) and 298 (99.33%) optimal solutions for the 300 instances in sets GA_{50} and GA_{100}, respectively. The number of optimal solutions for sets GB_{50} and GB_{100} (405 instances) are 335 (82.72%) and 360 (88.89%), respectively. The summarized performance results according to the type of solver and time limit for sets GA and GB, are reported in Table 6.3. The percentage of optimal solutions is denoted by $\%opt$, the percentage out of memory cases by $\%outm$, the average reported gap for non-optimal solutions as $GAP\%$ and the average CPU time in seconds as $Av.\ Time$.

The effect of the distribution of the product assembly times at the assembly stage is stronger than the type of solver or CPU time limit. The problems with more disperse assembly times appear to be easier to solve and need less CPU times. CPLEX always uses shorter CPU times in comparison with GUROBI. The average CPU time difference between the solvers is stronger for instance sets GB. As it is shown in the table, GUROBI finds less optimal solutions, results in higher gap values and in larger CPU times. However, GUROBI does not have memory errors. As a result each solver has its own benefits in this problem and there is no clear preference.

Solver	Time Limit	900s				3600s			
	Instance set	GA_{50}	GA_{100}	GB_{50}	GB_{100}	GA_{50}	GA_{100}	GB_{50}	GB_{100}
CPLEX	% opt	95.33	98.33	79.26	83.70	97.00	98.67	80.74	85.18
	% outm	0.00	0.00	1.97	4.44	1.67	0.00	14.57	7.90
	GAP%	0.24	0.05	0.57	0.37	0.03	0.05	0.20	0.29
	Av. Time (sec.)	50.98	24.68	214.14	137.50	97.77	58.05	345.40	326.98
GUROBI	% opt	95.67	97.33	72.34	80.74	96.67	98.33	76.79	83.21
	GAP%	0.27	0.08	1.13	0.85	0.17	0.07	0.75	0.38
	Av. Time (sec.)	57.78	45.79	391.83	245.84	156.51	99.62	990.47	720.05

Table 6.3: Results of the MILP solution for the tested solvers and time limits for instance sets of GA_{50}, GA_{100}, GB_{50} and GB_{100}.

6.4.2 Heuristics evaluation

The four proposed heuristics, H_1, H_2, H_3 and H_4 are now tested. We measure the Relative Percentage Deviation (RPD) as:

$$RPD = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \times 100$$

Where $Best_{sol}$ is the best found solution through all heuristics and the MILP model for any instance and Heu_{sol} is the makespan value obtained by a given heuristic for a given instance.

Tables 6.4 and 6.5 show the summarized results of the RPD values for the MILP and heuristics for instance sets GA and GB, respectively. Table 6.6 shows the average RPD values for the proposed heuristics for instance set GC as there are not solutions for the MILP models in this set. All results in the tables are categorized by n and f.

As can be observed, H_4 provides better results than the other heuristics in instance sets GA and GB, although this advantage is not as strong in the large

	GA_{50}						GA_{100}					
$f \times n$	CP	GU	H_1	H_2	H_3	H_4	CP	GU	H_1	H_2	H_3	H_4
2 × 10	0.00	0.00	9.50	8.43	3.24	2.60	0.00	0.00	2.21	1.74	0.46	0.15
2 × 12	0.00	0.00	8.24	7.37	4.04	3.03	0.00	0.00	3.58	3.45	1.76	0.99
2 × 14	0.00	0.00	8.29	6.63	3.36	2.15	0.00	0.00	4.50	4.38	0.62	0.63
2 × 16	0.08	0.00	9.31	8.51	4.59	4.12	0.00	0.00	4.30	4.08	1.16	1.02
2 × 18	0.02	0.00	7.27	6.49	3.90	2.77	0.00	0.72	4.10	3.73	1.97	1.53
3 × 10	0.00	0.00	5.55	4.63	2.87	1.88	0.00	0.00	2.31	2.45	1.54	1.34
3 × 12	0.00	0.00	4.70	4.89	1.43	1.67	0.00	0.00	1.94	1.32	0.98	0.32
3 × 14	0.00	0.00	4.61	3.44	1.81	0.97	0.00	0.00	1.51	1.15	0.28	0.25
3 × 16	0.00	0.00	4.79	4.13	1.39	0.59	0.00	0.00	2.63	1.96	0.72	0.31
3 × 18	0.00	0.00	4.09	4.19	1.35	0.97	0.00	0.00	2.15	1.68	0.78	0.54
Average	0.01	0.00	6.64	5.87	2.80	2.08	0.00	0.07	2.92	2.59	1.03	0.77

Table 6.4: Average Relative Percentage Deviation (RPD) of CPLEX (CP), GUROBI (GU) and the proposed heuristics for instance sets GA_{50} and GA_{100}.

instances. Heuristic H_4 can solve the largest instances in set GB of 24 jobs and 4 factories with a very small optimality gap of only 0.07%, which means this heuristic is very effective. The results show that the average RPD of H_4 is 1.1% for instance groups GA and GB. For the large instance set GC the best solution is always reported by H_4.

The CPU times of the proposed heuristics are reported in Table 6.7. They are negligible for all instance sets GA, GB and GC. For instance sets GA and GB, heuristics report solutions with low RPD in very short CPU times, while the presented MILP needs much more time in comparison.

On average, heuristic H_4 is slower than heuristics H_1 and H_2 but on absolute terms the CPU times are very small. The largest measured CPU time corresponds to H_4 is 0.18 seconds for one of the largest instances. From this final evaluation and considering the low RPD of H_4 in comparison with the other heuristics, we can conclude that it is a efficient and effective heuristic.

The observed differences in the performance for the two solvers and the

	GB_{50}						GB_{100}					
$f \times n$	CP	GU	H_1	H_2	H_3	H_4	CP	GU	H_1	H_2	H_3	H_4
2×20	0.22	0.01	6.95	6.81	2.26	2.27	0.00	0.01	3.32	3.32	1.03	0.58
2×22	0.18	0.04	5.27	5.28	2.37	1.41	0.06	0.05	2.38	2.16	0.89	0.58
2×24	0.21	0.19	4.79	4.76	1.90	1.75	0.69	0.00	3.32	2.27	1.39	1.14
3×20	0.00	0.01	3.21	2.58	1.33	1.13	0.03	0.00	2.48	2.21	1.27	0.50
3×22	0.00	0.00	2.89	2.34	1.68	0.98	0.00	0.00	1.54	1.37	0.47	0.33
3×24	0.18	0.04	3.16	2.12	1.25	0.61	0.31	0.00	1.38	1.15	0.72	0.25
4×20	0.00	0.00	2.26	1.90	1.22	0.44	0.00	0.00	1.58	1.60	0.66	0.50
4×22	0.00	0.00	2.00	1.36	0.55	0.31	0.00	0.08	1.73	1.41	0.72	0.34
4×24	0.00	0.00	2.50	2.04	1.12	0.77	0.33	0.05	1.87	1.35	0.35	0.07
Average	0.09	0.03	3.67	3.27	1.52	1.08	0.16	0.02	2.21	1.92	0.83	0.48

Table 6.5: Average Relative Percentage Deviation (RPD) of CPLEX (CP), GUROBI (GU) and the proposed heuristics for instance sets GB_{50} and GB_{100}.

	GC_{50}				GC_{100}			
$f \times n$	H_1	H_2	H_3	H_4	H_1	H_2	H_3	H_4
4×200	0.135	0.133	0.004	0.000	0.065	0.001	0.000	0.000
4×300	0.119	0.114	0.004	0.000	0.053	0.001	0.000	0.000
4×400	0.115	0.104	0.006	0.000	0.059	0.001	0.002	0.000
6×200	0.090	0.084	0.000	0.000	0.036	0.000	0.000	0.000
6×300	0.083	0.083	0.001	0.000	0.054	0.001	0.000	0.000
6×400	0.080	0.076	0.001	0.000	0.041	0.000	0.002	0.000
8×200	0.064	0.059	0.000	0.000	0.033	0.000	0.000	0.000
8×300	0.077	0.079	0.001	0.000	0.031	0.000	0.000	0.000
8×400	0.054	0.050	0.002	0.000	0.044	0.000	0.000	0.000
Average	0.091	0.087	0.002	0.000	0.046	0.000	0.001	0.000

Table 6.6: Average Relative Percentage Deviation (RPD) of the proposed heuristics for instance sets GC_{50} and GC_{100}.

tested heuristics are small. Therefore, some statistical tests are performed in order to ascertain if the observed differences are indeed statistically significant.

	H_1	H_2	H_3	H_4
GA_{50}	0.0024	0.0004	0.0043	0.0032
GA_{100}	0.0027	0.0004	0.0042	0.0031
GB_{50}	0.0052	0.0006	0.0034	0.0038
GB_{100}	0.0057	0.0006	0.0032	0.0174
GC_{50}	0.0459	0.0152	0.0766	0.0401
GC_{100}	0.0422	0.0147	0.0789	0.0422
Average	0.0174	0.0053	0.0284	0.0183

Table 6.7: Heuristic's CPU time (in seconds) for all instance groups.

For this reason, the Analysis of Variance (ANOVA) technique, as a powerful parametric tool, is used to examine the results. For the instance set GA, the tested methods can be categorized into three groups that have no statistical significant differences between them: solvers, H_1 and H_2 in one group, and H_3 and H_4 in another group. H_4 has the smaller RPD among the other heuristics and with a very little overlap with H_3. The observed differences between the first and the third group are small. The performance of the tested methods in set GB are similar to those of set GA. The interaction between the tested methods and assembly times distribution is interesting. As shown Figure 6.2, for instance set GB, instances with more disperse assembly times are easier to solve. The interaction between solvers and assembly times distribution has no statistically significant effect on the response variable. Less disperse assembly times always complicate problems when the heuristics are used and the interaction between them has a significant effect on RPD. With more disperse assembly times, H_4 has no statistical significant differences with solvers in both instance sets GA and GB.

Figure 6.3 shows a means plot with 99% confidence level Tukey's HSD intervals for the heuristics on the large instances set GC. As can be observed, H_3 and H_4 are statistically better than the other ones even though the absolute difference between all proposed methods is practically small.

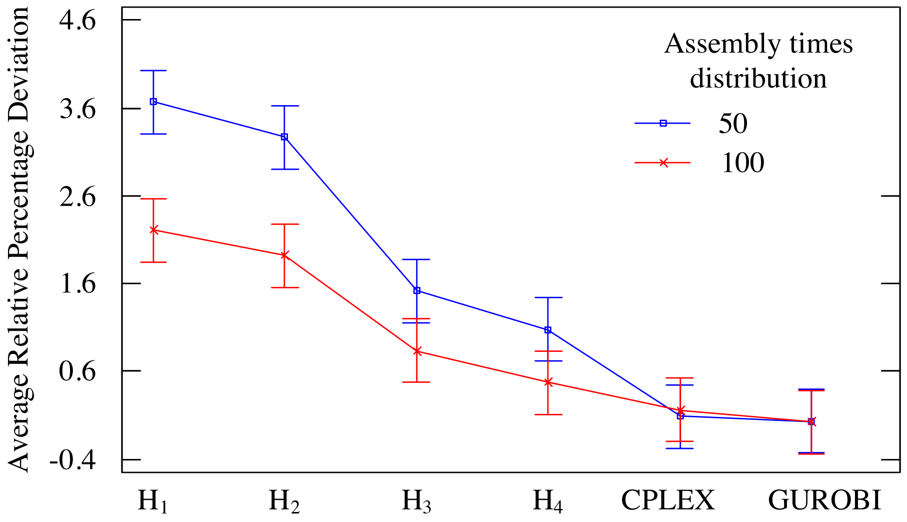


Figure 6.2: Means plot with the interaction between the distribution of the assembly processing times and the tested methods for instance set G_B. All means have Tukey's Honest Significant Difference (HSD) 99% confidence intervals.

6.5 Conclusions of this chapter

To the best of our knowledge, the studied problem in this chapter is the first attempt to solve the Distributed Parallel Machine and Assembly Scheduling Problem or DPMASP with eligibility constraints of factories and allowing empty machines on factories. In this problem, there is more than one production center with unrelated parallel machines to process jobs and a single assembly center to make final products from produced jobs following a defined assembly program. The objective is to minimize the makespan of products in the second assembly stage.

A mathematical model and four simple constructive heuristics are presented to solve the model. CPLEX and GUROBI are used to solve the mathematical model. Three sets of small, medium and large instances are considered

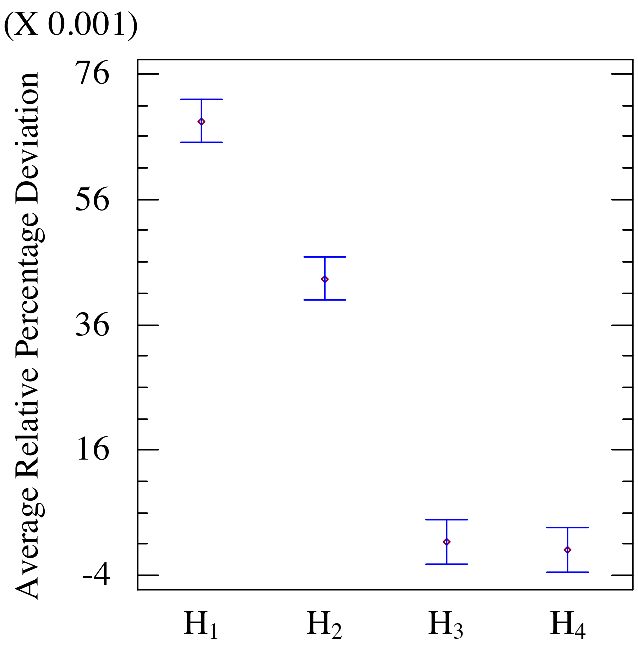


Figure 6.3: Means plot and 99% confidence level Tukey'S HSD intervals for the heuristics in the large instance set GC.

to test and to evaluate the mathematical model and heuristics. As it is shown by the obtained results, the mathematical model in conjunction with the solvers is able to to solve problems of up to 24 jobs and 16 machines distributed in 4 factories. ANOVAs were used to statistically analyze the results. In small and medium instances, both solvers perform better than the proposed heuristics. H_4 reports a smaller RPD among the presented heuristics, but statistical analysis shows that there are no statistical significant differences between H_4 and H_3.

Heuristics H_3 and H_4 use negligible CPU times and at the same produce good solutions that are very close to optimality in the cases for which the optimal solution has been obtained. The largest CPU time corresponds to H_4

and yet it is a very small 0.18 seconds. Still, the deviation from optimality barely exceeds 1.1%.

CHAPTER 7

GENERAL DISCUSSION OF THE RESULTS

Manufacturing systems have faced a number of significant obstacles recently. These obstacles are not without solutions. The industry needs to recreate competitive manufacturing systems to meet these recent challenges. Distributed manufacturing and assembly systems are two useful alternatives which are used to face these obstacles. In this thesis, both alternatives are considered in one production problem to increase the ability of such manufacturing systems to face challenges. The studied problems consist of two stages, production and assembly. The first stage is dedicated to the production of different jobs via various production centers (factories) comprising the concept of distributed manufacturing systems. The second stage is an assembly stage where jobs are assembled through a defined assembly program on a single assembly machine to make final products. Two different problems are studied by considering two different shop configurations for the production stage. Some extensions are added to both studied problems so as to bring them as close as possible to real world problems.

In the literature, there is a strong lack of consideration to both distributed manufacturing and assembly systems together in the same production problem. To the best of our knowledge, the only research which considers distributed manufacturing and assembly system together is presented by Xiong et al. (2014). The authors studied a distributed two-stage assembly system with setup times. They considered f distributed factories where each factory has the same m processing parallel machines at the first stage and an assembly machine at the second stage. Therefore, totally there are $f.m$ parallel machines and f assembly machines. Each product is made of m components produced by parallel machines. They developed heuristic methods and three hybrid meta-heuristics to minimize the total completion time. The problem studied by Xiong et al. (2014) is different from the problems studied in this thesis. First, in the problems presented in this thesis a separated assembly stage is considered, not an assembly operation at each factory. Second, the different jobs composing a product are allowed to be produced in different factories. Third, each product might have a number of jobs (components) different from m.

In the next section a general discussion of the obtained results of the studied problems in this thesis is provided.

7.1 The Distributed Assembly Permutation Flowshop Scheduling Problem

The first studied problem in this thesis is the Distributed Assembly Permutation Flowshop Scheduling Problem (DAPFSP). This model is a generalization of the Distributed Permutation Flowshop Scheduling Problem (DPFSP) which was presented for the first time by Naderi and Ruiz (2010). As mentioned before, all proposed problems in this thesis are composed of two stages, production and assembly. The first stage of the DAPFSP is composed of f identical production factories with flowshop configuration that produces jobs which they have been assembled into final products at the second assembly stage through a defined assembly program. The objective function for this problem is minimization of the makespan of the produced products on the assembly stage.

A Mixed Integer Linear Programming model (MILP), three heuristics and a Variable Neighborhood Descent (VND) are presented to solve the problem. To test the MILP, two commercial solver packages (CPLEX and GUROBI) are used. These types of problems are $\mathcal{NP}$-Complete and certainly the MILP solvers cannot solve large-sized problems. Therefore, it is necessary to develop a heuristic approach to solve them. In this thesis, three simple constructive heuristics are proposed. These heuristics are the SPT rule, Framinan and Leisten (2003) and two job assignment rules proposed by Naderi and Ruiz (2010). In total six proposed simple constructive heuristics are obtained by using two job assignment rules for each of the three heuristics.

In addition to simple heuristics, a VND method is also proposed to solve the problem. The VND approach needs an initial solution, therefore the six proposed constructive heuristics are used to obtain the initial solution instead of random solutions. Two neighborhood structures are considered for VND. The first one is a product local search that attempts to improve the objective function by examining different product sequences. The second neighborhood tries to find different partial job sequences for each product to improve the objective function.

To test the MILP model and the heuristics, two complete sets of instances based on different levels of four instance factors (number of jobs, machines, products and factories) have been generated. The MILP model is tested in a set of small instances with all the combinations of the test factors, using CPLEX and GUROBI solvers (commercial solver packages), serial and parallel computing (the number of CPU threads) and two time limits of 900 and 3600 seconds (time limitation for the stopping criterion). The heuristics are also tested in a set of larger instances.

In total there are 900 small instances and in the allowed CPU time, the LP model is able to find 516 optimum solutions (57.33 %).

The results show that GUROBI performs better than CPLEX. It is able to find more optimal solutions than CPLEX and reports smaller average gap and smaller average CPU time consumption than CPLEX. Overall, the time limit of 3600 seconds and parallel computing (2 threads) result in a larger number of optimal solutions. GUROBI with parallel computing results in a less average gap in comparison with serial computing, while this trend is reversed with CPLEX. GUROBI with two threads and 3600 seconds time limitation finds more optimum solutions.

The MILP results are also analyzed by the exhaustive CHAID method which is used to draw a decision tree to analyze the effect and interactions of the factors. The obtained decision tree shows that the most significant factor is the number of jobs. When this factor increases, the number of cases for which an optimal solution is found decreases. In the next level, nodes are split into the number of factories factor, except for the number of jobs equal to 8.

In the problem with larger amount of factories, jobs have more options for allocations. Therefore, the completion of the jobs is reduced. Consequently, the earliest possible time to start product assembly also is reduced, and the possibility of finding a better solution increases. The next significant factor is the number of products, except for the node with the number of jobs equal to 12 and the number of factories equal to 3, where in this node number of machines is a significant factor. The stopping criterion for branching is met for nodes with the number of jobs equal to 12 and the number of factories equal to 4 and for the node with the number of jobs equal to 24 and the number of factories equal to 2. The number of products factor interacts with the number of factories factor; that is, a higher percentage of optimal solutions is found when there is a larger number of products. No further statistically significant divisions are found. And also, the effect of type of solver, thread and time limit are not statistically significant.

The twelve proposed methods (six simple constructive heuristics and six

VND which start with six heuristics) are tested with both set of instances. The Relative percentage deviation (RPD) is measured for comparisons of the proposed heuristics.

The results show that the mathematical model is unable to find an optimum or best solution for all the small instances. The problem becomes harder for the MILP by increasing the number of jobs and by decreasing the number of factories. All VND algorithms perform better than the constructive algorithms and significantly improve the initial solutions. The second job assignment rule works better than the first one. This rule examines a job in all factories to find the best option for the final assignment.

A multi-factor ANOVA is carried out for the VND results to asses the statistical significant of the observations. The average RPD value for all the simple constructive heuristics is 6.75%, and this amount lowers to 0.63% for the VND methods. The difference in RPD between simple constructive heuristics and VND is very high. Therefore, two separated ANOVAs for the simple heuristics and the VND methods are considered. All controlled factors (number of jobs, machines, products and factories) in the ANOVA analysis, with some exceptions, result in strong statistically significant differences in the RPD response variable.

Among all simple constructive heuristics, the second one performs better than the others and there are no significant differences between the rules used to assign jobs to factories. In the VND algorithm, the job assignment rules are important, and the second one statistically outperforms the first. The VND algorithm improves all the initial solutions. There are no significant differences between the three VND using the second job assignment rule. The CPU times used to solve the small instances with the considered algorithms are negligible.

For the large instances, algorithms are separated into two groups: the VND algorithms and the second simple constructive heuristics with both job assignment rules with better performance in one group and the rest in another group. The algorithms which use the second job assignment rule perform better than those with the first. Since the second group does not report good results in comparison with the first one, they are eliminated from the statistical analysis. The type of initial solution for the VND algorithms does not play an important role and there is no significant differences between the VND algorithms that use the same job assignment rule. The number of jobs and products has no significant effect on the response variable of the algorithms. An increase in the number of machines always complicates problems, while this trend for the number of factories is the reverse.

All RPDs of the VND using the second simple constructive heuristic with the second job assignment rule for initialization are consistently lower than those of the other algorithms.

The VND algorithms use more CPU time compared to simple constructive methods. The maximum average CPU time reported by the VND is 23.20 seconds. Although the VND methods consume more CPU time to solve the problem, they report smaller (almost nine times lower) RPD values than simple constructive algorithms.

To have better solution quality, VND algorithms are the best options. Otherwise, if the CPU time consumption is more important, a simple constructive algorithm can be a good choice.

7.2 The Distributed Assembly Permutation Flowshop Scheduling Problem with Sequence Dependent Setup Times

In Chapter 4, sequence dependent setup times (SDST) are added to the studied problem in Chapter 3 so as to bring it as close as possible to the reality of production shops. The objective of this problem is also makespan minimization of the products on the last stage. This problem with an extension of SDST for all machines is certainly $\mathcal{NP}$-Complete and is more complicated than DAPFSP. Therefore, two simple heuristics and two metaheuristics (VND and Iterated

Greedy) are proposed to solve it. A complete parameter calibration through a Design Of Experiments (DOE) approach is carried out for the metaheuristics. A series of studies are done to justify the elimination of a parameter in the Iterated Greedy. Finally, the performance of all proposed methods is compared through extensive computational and statistical experiments.

Two heuristics are proposed to solve the problem. In total, four heuristics are obtained after considering the two job assignment rules presented by Naderi and Ruiz (2010) for each one of the two heuristics.

In addition to heuristics, two metaheuristics are also proposed to solve the problem. The first suggested metaheuristic is a VND that employs two different solution representations and two neighborhood structures. The VND which uses the first representation solution (Pr1), applies LS_1 and LS_2 as neighborhoods. LS_1 extracts each job at each factory and reinserts it in all possible positions of the PFSP at that factory. The second neighborhood, LS_2, takes all jobs assigned at each factory and inserts them into all possible positions in all other factories looking for a makespan improvement at the involved factories. The VND that employs the second solution representation (Pr2) also uses two neighborhood structures, LS_P and LS_J, respectively. These neighborhoods are same as the presented neighborhoods in Chapter 3.

The next proposed metaheuristic is the Iterated Greedy (IG) which applies iteratively four phases (destruction, construction, local search and acceptance criterion) to the incumbent solution until a termination criterion is reached. A method which constructs the initial solution for IG is selected among all four proposed heuristics after testing them. The four phases of the proposed IG are different depending on the solution representation Pr1 or Pr2.

In most existing IG, once the first three phases are carried out over the incumbent solution, a possibly different schedule is obtained and must be determined if it replaces the incumbent one. Usually, a simulated annealing-like type of acceptance criterion (AC_1) with a constant temperature (T) is used in most IG algorithms, where T needs to be calibrated. Two additional acceptance criteria are proposed. In the first (AC_2), instead of the factor T in the simulated annealing-like type acceptance criterion, the Relative Percentage Difference (RPD) is used. In the second proposed acceptance criterion (AC_3), the factor T is eliminated and it is further simplified.

Statistical techniques are applied to verify the quality of the results of the two new proposed acceptance criteria for the DAPFSP-SDST problem. The Design Of Experiments (DOE) approach is used to calibrate the proposed methods and a set of instances is generated for this experiment.

Three factors are considered for VND to be calibrated, the type of solution representation, job assignment rules and simple constructive heuristic used for initialization. All studied factors are statistically significant according to the analysis and ANOVA results. The most significant are the representation (Pr), job assignment rules and the initial solution (INI). The second solution representation and the second job assignment rule result in statistically better performance. The second solution representation, second job assignment rule and the second constructive heuristics which applies the first job assignment rule (CH_{21}) are selected as running parameters for VND.

For the IG calibration there are three additional factors: percentage of jobs to destruct in the destruction phase, type of acceptance criterion and the value of T used in the calculation. The first two acceptance criterion (AC_1 and AC_2) depend on the aforementioned parameter T, whereas the third (AC_3) does not have a T factor. Therefore, two different experiments are carried out.

The results of the first experiment show that the only non-significant factor is T with a p-value very close to 1. For AC_1, increasing the value of T results in better solutions and this situation is just the opposite for AC_2. T is not significant and also the previous studies on the IG methodology show that T is statistically insignificant. These observations reinforce our idea to remove T from the acceptance criterion. For the second experiment, T is set at 2.5 and 0.5 for AC_1 and AC_2, respectively.

In the second experiment all previous factors and all three acceptance criteria with fixed T are considered. Results of the ANOVA indicate that the interaction between the solution representation and job assignment rule factors have the most significant effect. Similar to the VND, the second solution representation and the second job assignment rule results in better performance. The reported p-values of more than 0.85 for the type of acceptance criterion factor indicate that there are very little (if any) differences between the three proposed acceptance criteria. It is preferable to employ the third proposed criterion with equivalent performance and at the same time with one less parameter. For the final experiments in addition to (AC_3), the original Ruiz and Stützle (2007) acceptance criterion (AC_1) are considered to attest if our new proposed acceptance criterion is actually equivalent or not.

After, the parameters of VND and IG are calibrated, the proposed approaches are tested. The four proposed simple constructive heuristics are fast and take very little CPU time, where the maximum reported CPU time is just 0.079 seconds and the average observed CPU time in all results is only 0.008 seconds. These heuristics are almost instantaneous even for the largest tested instances. All four heuristics provide similar results and with a average RPD between a little more than 16% and below 18%.

According to the ANOVA statistical test on the obtained results, three non-controllable factors, number of job, products and factories are very significant, in this order. The first simple constructive heuristic using the second job assignment rule, CH_{12}, is statistically better than the others.

Finally, we test VND and IG . The IG method is tested with two different acceptance criteria (AC_1 and AC_3). VND results are relatively good with an average of 5.33% RPD in all tests with almost 37 seconds of average CPU time. Two termination criteria are considered for the proposed Iterated Greedy methods. An interesting conclusion is that AC_3, albeit simpler and with one less parameter, gives better results when compared with the regular acceptance criterion. ANOVA results indicate that VND is clearly not statistically better than the IG methods. The results obtained by ANOVA show that, IG$_3$ is statistically better than IG_1, this is our proposed simpler acceptance criteria is better.

7.3 The Distributed Parallel Machine and Assembly Scheduling Problem with Eligibility Constraints

As mentioned before, all considered problems in this thesis consist of two stages, production and assembly. Chapter 5 presented a new problem where there is a set of identical factories with parallel machines in the production stage. This problem is referred to as the Distributed Parallel Machine and Assembly Scheduling Problem or DPMASP. The objective function of this problem is still the same as the previous problems. Due to some technological constraints, at least one job should assigned to each machine and any of them cannot be idle. Also, some jobs can be processed only in certain factories. To solve this problem, a mathematical model and two heuristics are proposed. CPLEX and GUROBI are used to test the mathematical model. Comprehensive computational experiences are carried out to evaluate the performance of the proposed solution methods.

Two different heuristics based on two different product sequence construction methods and one job to machine-factory assignments for products ar proposed. To test and evaluate the proposed solution methods, six complete sets of instances considering different number of problem characteristics such as different number of jobs, machines, factories and products have been generated. These sets of instances are divided into three size-based groups: small, medium and large-sized instances, referred to as GA, GB and GC, respectively. Two distributions of the assembly processing times (referred to as 50, and 100) are considered for each set of instances. The final sets of instances are then denoted as $GA_{50}, GA_{100}, \ldots, GC_{100}$. The proposed MILP model is tested only on small and medium-sized instances sets (GA and GB) given the impossibility to solve large instances. As the obtained results show, generally the solvers are able to find 294 (98.00%) and 297 (99.00%) optimal solutions in sets GA_{50} and GA_{100}, respectively. According to the results it is clear that the effect of the distribution of the assembly times at the assembly stage is much stronger than either the type of solver or CPU time limit.

The second proposed algorithm is generally much better than the first one in all groups of instances, although the difference is not very big in the large instances. It is important to indicate that the second heuristic reports a very small gap for the instances set of GB which indicates that it is a very capable heuristic close to optimality in performance. The reported RPD average by the second heuristic for both instance groups GA and GB is below 1%. Since it is not possible to calculate the optimum solution for the large instances in GC, we only have an overall picture were the second heuristic always obtains the best solution. The consumed CPU times by the heuristics are negligible. For example, the average CPU times are below one tenth of a second for the largest instances in group GC. The largest measured CPU time corresponds to the second heuristic has been 0.41 seconds. From this final evaluation and considering the relative RPD of the second heuristic we can conclude that it is a capable and very fast method.

The observed differences are large in all cases for the proposed heuristics and very small for the two solvers. Therefore, some statistical analyses are carried out in order to ascertain if the observed differences are indeed statistically significant. For the small instances there are no statistically significant difference in the performance of CPLEX and GUROBI and the second heuristic is statistically better than the first one.

The results for the medium sized-instances in set GB are similar to those of set GA. For the large instances in group GC, the observed differences in the average RPD between the two heuristics show that the second heuristic is statistically better than the first one.

In general, results show that the proposed model is able to solve moderately-sized instances, and the second heuristic is fast, giving optimal solutions close to optimum in less than half a second in the worst case.

7.4 Heuristics for a Distributed Parallel Machine and Assembly Scheduling Problem with Eligibility Constraints

The difference between the studied model in Chapters 5 and 6 is to remove the constraint that forces parallel machine to have at least one job. Therefore, in the studied problem presented in Chapter 6, empty machines at factories are permitted. A mathematical model, four simple, fast and high-performing heuristics are proposed to solve the considered problem. Four heuristics based on two different product sequence construction and two different job to machine-factory sequence for products construction are proposed.

To test and to evaluate the proposed MILP model and constructive heuristics, six complete sets of instances with different problem characteristics are generated. Three size-based groups of instances have been generated: small (GA), medium (GB) and large (GC). Two different distributions of product assembly times referred to as 50 and 100 are considered for each group of instances. The three small, medium and large-sized sets of instances in combination with these distribution intervals are denoted as $GA_{50}, GA_{100}, \ldots, GC_{100}$.

Our results show that the mathematical model is able to solve moderately-sized instances and is unable to solve the large instances in GC. Therefore, only the instance sets GA and GB are used to test the proposed MILP model. Two different stopping time criteria of 900 and 3600 seconds are considered with each solver. In total, the solvers were able to find 294 (98.00%) and 298 (99.33%) optimal solutions for the 300 instances in sets GA_{50} and GA_{100}, respectively. The number of optimal solutions for sets GB_{50} and GB_{100} (405 instances) decrease to 335 (82.72%) and 360 (88.89%), respectively.

Like in the previous studied problem with the constraint, the effect of the distribution of the product assembly times at the assembly stage is stronger than the type of solver or CPU time limit. The problems with more disperse assembly times appear to be easier to solve and need less CPU times. CPLEX always uses less CPU times than GUROBI and this difference between the solvers is stronger for instance sets GB. GUROBI finds less optimal solutions, results with higher gap values and in larger CPU times. However, GUROBI does not have memory errors. As a result each solver has its own benefits in this problem and there is no clear preference.

The fourth heuristic provides better results than the other ones in both instance sets GA and GB, although this advantage is not as strong in the large instances. The fourth heuristic can solve the largest instances in set GB of 24 jobs and 4 factories with a very small optimality gap of only 0.07%, which means this heuristic is very effective. The results show that the average RPD of the forth heuristic is 1.1% for instance groups GA and GB.

The CPU times of the proposed heuristics are negligible for all instance sets. For instance sets GA and GB, heuristics report solutions with low RPD in very short CPU times, while the presented MILP needs much more time in comparison. The largest measured CPU time corresponds to the fourth heuristic is 0.18 seconds for one of the largest instances. From this final evaluation and considering the low RPD of the fourth heuristic in comparison with the other heuristics, we can conclude that it is an efficient and effective method.

The observed differences in the performance for the two solvers and the tested heuristics are small. Therefore, the ANOVA technique is used to examine the results in order to ascertain if the observed differences are indeed statistically significant. For instance set GA, the fourth heuristic has the smaller RPD among the other heuristics. The performance of the tested methods in set GB are similar to those of set GA.

CHAPTER 8

CONCLUSIONS AND FUTURE RESEARCH

In this final chapter we draw the general conclusions about the research lines pursued in this thesis. The thesis focused on solving distributed manufacturing and single assembly scheduling problems. In the following we outline some general discussions about the different topics of this work and summarize the main contributions of this thesis. The contributions are manyfold, presenting two new methods to manage a manufacturing system to tackle the recent global challenges and solving them with solution techniques such as mathematical models, heuristics and metaheuristics. In Section 8.1, the main achievements of the thesis are recalled. In Section 8.2, the main limitations of the work are outlined and several possible directions for future research are presented.

8.1 Results

The main contributions of the thesis are combining distributed manufacturing and assembly systems in a production problem that is able to tackle the recent global challenges. The problem consists of two stages, distributed manufacturing and a single assembly centers, respectively. In this new production setting products composed of many jobs are produced. These jobs are manufactured on the first stage and according to a defined assembly program constitute the final products after assembly. In two different production problems, two different process structures, flowshop and parallel machines are considered for the distributed manufacturing centers. Also several extensions are studied so as to bring the problem as close as possible to the reality of production shops. Different methods such as mathematical models, simple constructive heuristics and metaheuristics are presented to solve these problems. The objective of all throughout the thesis is the minimization of the makespan of the products which are assembled on the last stage.

As presented in Chapter 3, all distributed manufacturing factories in the first stage of the problem have a flowshop structure. To simplify the problem, the same job permutation is considered for all machines on each manufacturing center. This problem is referred to as the Distributed Assembly Permutation Flowshop Scheduling Problem (DAPFSP). A Mixed Integer Linear Programming model (MILP) is presented for the DAPFSP and two solvers, CPLEX and GUROBI are used to solve it. Three constructive algorithms and a Variable Neighborhood Descent (VND) algorithm have been designed for the problem. The constructive heuristics and VND are tested with two groups of small and large instances. A comprehensive ANOVA statistical analysis was used to analyze results. The results show that the VND algorithm performs better than constructive heuristics. On the other hand, the simple constructive heuristics consume little CPU time and still produce reasonable solutions.

The results of this work have been presented at national and international conferences:

- Hatami, S., Ruiz, R., and Andrés-Romano, C. (2013). Two simple constructive algorithms for the distributed assembly permutation flowshop scheduling problem. In *Book of Proceedings of the* 17th *International Conference on Industrial Engineering and Industrial Management, XVII*

Congreso de Ingeniería de Organizacíon (CIO), pages 245-252, Valladolid, Spain. (This paper received the best paper award at the conference)

- Hatami, S., Ruiz, R., and Andrés-Romano, C. (2013). A mixed integer linear programming model for distributed assembly permutation flowshop scheduling problem. In *IFORS ELAVIO*, Valencia, Spain.

Also the results of this chapter have been published as a chapter in a book:

- Hatami, S., Ruiz, R., and Andrés-Romano, C. (2014). Two Simple Constructive algorithms for the Distributed Assembly Permutation Flowshop Scheduling Problem. *In* Iglesias, C. H., López-Paredes, A., and Pérez Ríos, J. M., editors. *Managing Complexity: Challenges for Industrial Engineering and Operations Management*, chapter 2, pages 139-145. Springer International Publishing.

Finally we have published the results of this chapter as a printed article in an international journal listed in the Journal Citation Reports (JCR):

- Hatami, S., Ruiz, R., and Andrés-Romano, C. (2013). The distributed assembly permutation flowshop scheduling problem. *International Journal of Production Research*, 51(17):5292–5308.

During the years of the publication, the article has received 4 citations according to Google scholar and 2 according to Web of Knowledge.

To bring the DPFSP as close as possible to the reality of production shops, sequence dependent setup times (SDST) are added to all machines in the production stage and to the single assembly machine on the second stage (see Chapter 4). This problem represents a solid step forward in solving more realistic distributed scheduling problems. Two simple heuristics are combined with two existing job to factory assignment rules to solve the problem. Furthermore, two metaheuristics, Variable Neighborhood Descent

(VND) and Iterated Greedy (IG) are designed for the problem. Two different solution representations are considered. A new acceptance criterion that does not consider a simulated annealing-like temperature as it is common in the IG literature is proposed. According to the results, the new parameter-less acceptance criterion is better which simplifies the already simple IG even further. The performance of all presented methods is analyzed by statistical techniques. The constructive heuristics produce reasonably good results in short (almost instantaneous) CPU times, while results close to optimality are reached by the more time consuming methods like VND or IG.

We have presented the results of this chapter in the following international conference:

- Hatami, S., Ruiz, R., and Andrés-Romano, C. (2014). Simple constructive heuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times. In *Second International Conference on Control, Decision and Information Technologies*, CoDIT14, Metz, France.

Also, the results of this chapter are accepted in another international journal cited in the Journal Citation Reports (JCR):

- Hatami, S., Ruiz, R., and Andrés-Romano, C. (2015). Heuristics and Metaheuristics for the Distributed Assembly Permutation Flowshop Scheduling Problem with Sequence Dependent Setup Times. *International Journal of Production Economics*.

In chapter 5, the first stage of the problem is changed to a set of identical factories with unrelated parallel machines and the second stage is a single assembly machine as in the previous problems. The problem is referred as the Distributed Parallel Machine and Assembly Scheduling Problem or DPMASP. Due to technological constraints, the vacancy of the machines is not allowed and some jobs can might be processed only in certain factories. A mathematical model and two high-performing heuristics are proposed for the model. Comprehensive computational experiments are carried out to evaluate the performance of the mathematical model and heuristics. Both proposed simple heuristics are fast but one of them gives close to optimal solutions in less than half a second in the worst case. Furthermore, for large instances the heuristics perform very good and give solutions in almost a negligible time.

We published the results of Chapter 5 as a journal article in *International Journal of Production Management and Engineering* in year 2015:

- Hatami, S., Ruiz, R., and Andrés-Romano, C. (2015). The distributed assembly parallel machine scheduling problem with eligibility constraints. *International Journal of Production Management and Engineering*, 3(1):13–23.

As mentioned, the presented model in Chapter 5 has an especial constraint that no machine at any factory might be empty due to technological or economical constraints. This constraint is relaxed in the model considered in Chapter 6. Four simple, fast and high-performing heuristics are designed. Comprehensive computational experiments on three different sized sets of instances are carried out to evaluate the performance of the mathematical model and heuristics. All heuristics use negligible CPU times in comparison with the solvers.

The results of this work are submitted to an international conference:

- Hatami, S., Ruiz, R., and Andrés-Romano, C. Heuristics for a Distributed Parallel Machine Assembly Scheduling Problem with Eligibility Constraints submitted to *the* 6^{th} *International Conference on Industrial Engineering and System Management* (IESM 2015) Conference, Seville, Spain.

8.2 Future work

In this section we report the limitations of our contributions and open the doors to future research possibilities.

- The joint consideration of distributed manufacturing, an assembly stage and sequence-dependent setup times both in the production stage and in the assembly stage result in a more realistic problem. However, it is not yet a fully practical industrial case. The literature on scheduling problems starts in the 1950s and yet today there is a widely recognized gap between the theory and practice of scheduling systems. This has been recently pointed out in the book of Framinan et al. (2014). Many authors during the last 30 years or more have been working towards closing this research gap. This thesis is another step in this direction. Closing this research gap in a thesis is not possible and large amount of work is needed to close it to practical problems. Real production shops have even more constraints and real situations than those considered in this thesis. Therefore, considering more constraints and additional characteristics in the problem setting will make it even more realistic and close to practical problems. These constraints and additional characteristics could be for example:
  - Transportation times or a transportation stage between the production and assembly stages.
  - Buffer between the stages.
  - Other shop configuration for production stage.
  - Considering maintenance operations.
  - Heterogeneous distributed factories could account for more complex scenarios.
  - The assembly stage could be made more complex with parallel machines or ever with shop problems.

- Other objective functions such as tardiness or production costs instead of makespan is an interesting research path.
- Precedence constraints among the jobs of a given product could be considered as well.
- Other heuristics or metaheuristics may report better solutions if compared to the proposed ones in this thesis.

BIBLIOGRAPHY

(2014). *IBM ILOG AMPL version 12.6.2 User's Guide, Standard (Command-line) Version Including CPLEX Directives*. IBM Corp.

(2015). *GUROBI Optimizer reference manual, Version 6.0*. Gurobi Optimization, Inc.

Al-Anzi, F. S. and Allahverdi, A. (2006). A hybrid tabu search heuristic for the two-stage assembly scheduling problem. *The International Journal of Operational Research*, 3(2):109–119.

Al-Anzi, F. S. and Allahverdi, A. (2007). A self-adaptive differential evolution heuristic for two-stage assembly scheduling problem to minimize maximum lateness with setup times. *European Journal of Operational Research*, 182(1):80–94.

Al-Anzi, F. S. and Allahverdi, A. (2009). Heuristics for a two-stage assembly flowshop with bicriteria of maximum lateness and makespan. *Computers & Operations Research*, 36(9):2682–2689.

Allahverdi, A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, 246(2):345–378.

Allahverdi, A. and Al-Anzi, F. S. (2009). The two-stage assembly scheduling problem to minimize total completion time with setup times. *Computers & Operations Research*, 36(10):2740–2747.

Allahverdi, A., Gupta, J. N. D., and Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *OMEGA, The International Journal of Management Science*, 27(2):219–239.

Allahverdi, A., Ng, C. T., Cheng, T. C. E., and Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, 187(3):985–1032.

Atighehchian, A., Bijari, M., and Tarkesh, H. (2009). A novel hybrid algorithm for scheduling steel-making continuous casting production. *Computers & Operations Research*, 36(8):2450–2461.

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. Wiley, New York.

Bartz-Beielstein, T., Chiarandini, M., Paquete, L., and Preuss, M., editors (2010). *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, New York.

Basso, D., Chiarandini, M., and Salmaso, L. (2007). Synchronized permutation tests inreplicated $i \times j$ designs. *Journal of Statistical Planning and Inference*, 137(8):2564–2578.

Biggs, D., Ville, B. D., and Suen, E. (1991). A method of choosing multiway partitions for classification and decision trees. *Journal of Applied Statistics*, 18(1):49–62.

Boardman, M. and Trappenberg, T. (2006). A heuristic for free parameter optimization with support vector machines. In *Neural Networks, 2006. IJCNN '06. International Joint Conference on*, pages 610–617.

Boussaïd, I., Lepagnot, J., and Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences*, 237:82–117.

Brussel, H. V. (2014). Holonic manufacturing systems. In *CIRP Encyclopedia of Production Engineering*. Springer Berlin Heidelberg.

Brussel, H. V., Bongaerts, L., Wyns, J., Valckenaers, P., and Ginderachter, T. V. (1999). A conceptual framework for holonic manufacturing: identification of manufacturing holons. *Journal of Manufacturing Systems*, 18(1):35–52.

Brussel, H. V., Wyns, J., Valckenaers, P., Bongaerts, L., and Peeters, P. (1998). Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, 37(3):255–274.

Butala, P. and Sluga, A. (2002). Dynamic structuring of distributed manufacturing systems. *Advanced Engineering Informatics*, 16(2):127–133.

Campbell, H. G., Dudek, R. A., and Smith, M. L. (1970). A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, 16(10):B630–B637.

Chan, F. T. S., Chung, S. H., and Chan, P. L. Y. (2005a). An adaptive genetic algorithm with dominated genes for distribute scheduling problems. *Expert Systems with Applications*, 29(2):364–371.

Chan, F. T. S., Chung, S. H., and Chan, P. L. Y. (2005b). An adaptive genetic algorithm with dominated genes for distributed scheduling problems. *Expert Systems with Applications*, 29(2):364–371.

Chan, F. T. S., Chung, S. H., and Chan, P. L. Y. (2006a). Application of genetic algorithms with dominant genes in a distributed scheduling problem in flexible manufacturing systems. *International Journal of Production Research*, 44(3):523–543.

Chan, F. T. S., Chung, S. H., Chan, P. L. Y., Finke, G., and Tiwari, M. K. (2006b). Solving distributed FMS scheduling problems subject to maintenance: genetic algorithms approach. *Robotics and Computer-Integrated Manufacturing*, 22(5-6):493–504.

Chan, H. K. and Chan, F. T. S. (2010). Comparative study of adaptability and flexibility in distributed manufacturing supply chains. *Decision Support Systems*, 48(2):331–341.

Chen, S. H., Chang, P. C., Cheng, T. C. E., and Zhang, Q. (2012). A self-guided genetic algorithm for permutation flowshop scheduling problems. *Computers & Operations Research*, 39(7):1450–1457.

Cheng, T. C. E., Gupta, J. N. D., and Wang, G. (2000). A review of flowshop scheduling research with setup times. *Production and Operations Management*, 9(3):262–282.

Ching-Jong, L. and Li-Man, L. (2008). Improved MILP models for two-machine flowshop with batch processing machines. *Mathematical and Computer Modelling*, 48(7-8):1254–1264.

Chutima, P. and Chimklai, P. (2012). Multi-objective two-sided mixed-model assembly line balancing using particle swarm optimisation with negative knowledge. *Computers & Industrial Engineering*, 62(1):39–55.

Ciavotta, M., Minella, G., and Ruiz, R. (2013). Multi-objective sequence dependent setup times flowshop scheduling: a new algorithm and a comprehensive study. *European Journal of Operational Research*, 227(2):301–313.

Ciuprina, G., Ioan, D., and Munteanu, I. (2002). Use of intelligent-particle swarm optimization in electromagnetics. *Magnetics, IEEE Transactions on*, 38(2):1037–1040.

Coffman, E. G. (1976). *Computer and Job Shop Scheduling Theory*. John Wiley & Sons Inc, New York.

Conway, R. W., Maxwell, W. L., and Miller, L. W. (1968). *Theory of Schedulig*. Addison-Wesley.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Third Annual ACM Symposium on the Theory of Computing*, pages 151–158, New York. ACM.

Dong, X., Huang, H., and Chen, P. (2009). An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Computers & Operations Research*, 36(5):1664–1669.

Dorigo, M. (1992). *Optimization, Learning and Natural Algorithms*. Ph.D. thesis, Politecnico di Milano, Italy.

Dresbach, S. (1994). A new heuristic layout algorithm for directed acyclic graphs. In *Operations Research Proceedings*, pages 121–126.

Dréo, J., Pétrowski, A., Siarry, P., and Taillard, E. (2006). *Metaheuristics for Hard Optimization*. Springer-Verlag Berlin Heidelberg.

ElMaraghy, H., Schuh, G., ElMaraghy, W., Piller, F., Schönsleben, P., Tseng, M., and Bernard, A. (2013). Product variety management. *CIRP Annals - Manufacturing Technology*, 62(2):629–652.

Engels, D. W., Feldman, J., Karger, D. R., and Ruhl, M. (2001). Parallel processor scheduling with delay constraints. In *Proceedings of the* 12^{th} *annual ACM-SIAM symposium on Discrete algorithms*, pages 577–585, PA, USA. Society for Industrial and Applied Mathematics Philadelphia.

Farmer, J. D., Packard, N. H., and Perelson, A. S. (1986). The immune system, adaptation, and machine learning. *Physica D 2*, pages 187–204.

Fernandez-Viagas, V. and Framinan, J. M. (2015). A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 53(4):1111–1123.

Ferreira, P., Doltsinis, S., and Lohse, N. (2014). Symbiotic assembly systems-A new paradigm. In ElMaraghy, H., editor, *Procedia CIRP, Variety Management in Manufacturing Proceedings of the 47th CIRP Conference on Manufacturing Systems*, volume 17, pages 26–31.

Framinan, J. M., Gupta, J. N. D., and Leisten, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55(1):1243–1255.

Framinan, J. M. and Leisten, R. (2003). An efficient constructive heuristic for flowtime minimisation in permutation flow shops. *OMEGA, The International Journal of Management Science*, 31(4):311–317.

Framinan, J. M. and Leisten, R. (2008). Total tardiness minimization in permutation flow shops: a simple approach based on a variable greedy algorithm. *International Journal of Production Research*, 46(22):6479–6498.

Framinan, J. M., Leisten, R., and Ruiz, R. (2014). *Manufacturing Scheduling Systems. An Integrated View on Models, Methods and Tools*. Springer, New York.

Framinan, J. M., Leisten, R., and Ruiz-Usano, R. (2002). Efficient heuristics for flowshop sequencing with the objectives of makespan and flowtime minimization. *European Journal of Operational Research*, 141(3):559–569.

Gao, J. and Chen, R. (2011a). A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Computational Intelligence Systems*, 4(4):497–508.

Gao, J. and Chen, R. (2011b). An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems. *Scientific Research and Essays*, 6(14):3094–3100.

Gao, J., Chen, R., Deng, W., and Liu, Y. (2012a). Solving multi-factory flowshop problems with a novel variable neighbourhood descent algorithm. *Journal of Computational Information Systems*, 8(5):2025–2032.

Gao, J., Chen, R., and Liu, Y. (2012b). A knowledge-based genetic algorithm for permutation flowshop scheduling problems with multiple factories. *International Journal of Advancements in Computing Technology*, 4(7):121–129.

Gao, J., Chen, R., and W., D. (2013). An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 51(3):641–651.

Gendreau, M. and Potvin, J. Y., editors (2010). *Handbook of Metaheuristics (International Series in Operations Research & Management Science)*. Springer US, Second edition.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5):533–549.

Glover, F. and Kochenberger, G. A. (2003). *Handbook of Metaheuristics*. Springer Science & Business Media.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine learning*. Addison-Wesley Longman Publishing Co., Inc.

Goldman, S. L., Nagel, R. N., and Preiss, K. (1994). *Agile Competitors and Virtual Organizations: Strategies for Enriching the Customer*. Van Nostrand Reinhold.

Gupta, J. N. D. (1971). A functional heuristic algorithm for the flowshop scheduling problem. *Operational Research Quarterly(1970-1977)*, 22(1):39–47.

Gupta, J. N. D. (1972). Heuristic algorithms for multistage flowshop scheduling problem. *A I I E Transactions*, 4(1):11–18.

Gupta, J. N. D. and Stafford, E. F. (2006). Flowshop scheduling research after five decades. *European Journal of Operational Research*, 169(3):699–711.

Hansen, P. and Mladenović, N. (2001). Variable neighborhood search: principles and applications. *European Journal of Operational Research*, 130(3):449–467.

Hariri, A. M. A. and Potts, C. N. (1997). A branch and bound algorithm for the two-stage assembly scheduling problem. *European Journal of Operational Research*, 103(2):547–556.

Hatami, S., Ebrahimnejad, S., Tavakkoli-Moghaddam, R., and Maboudian, Y. (2010). Two meta-heuristics for the three-stage assembly flowshop scheduling with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, 50(9-12):1153–1164.

Hatami, S., Ruiz, R., and Andrés-Romano, C. (2013). The distributed assembly permutation flowshop scheduling problem. *International Journal of Production Research*, 51(17):5292–5308.

Hatami, S., Ruiz, R., and Andrés-Romano, C. (2015). The distributed assembly parallel machine scheduling problem with eligibility constraints. *International Journal of Production Management and Engineering*, 3(1):13–23.

Hejazi, S. R. and Saghafian, S. (2005). Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research*, 43(14):2895–2929.

Hu, S. J., Ko, J., Weyand, L., ElMaraghy, H. A., Lien, T. K., Koren, Y., Bley, H., Chryssolouris, G., Nasr, N., and Shpitalni, M. (2011). Assembly system design and operations for product variety. *CIRP Annals - Manufacturing Technology*, 60:715–733.

Jackson, J. R. (1956). An extension of johnson's result on job lot scheduling. *Naval Research Logistics Quarterly*, 3:201–203.

Jacobs, L. W. and Brusco, M. J. (1995). A local-search heuristic for large set-covering problems. *Naval Research Logistics*, 42:1129–1140.

Jha, N. K. (1991). In *Handbook of Flexible Manufacturing Systems*. Academic Press.

Jia, H. Z., Fuh, J. Y. H., Nee, A. Y. C., and Zhang, Y. F. (2002). Web-based multi-functional scheduling system for a distributed manufacturing environment. *Concurrent Engineering: Research and Applications*, 10(1):27–39.

Jia, H. Z., Fuh, J. Y. H., Nee, A. Y. C., and Zhang, Y. F. (2007). Integration of genetic algorithm and Gantt chart for job shop scheduling in distributed manufacturing systems. *Computers & Industrial Engineering*, 53(2):313–320.

Jia, H. Z., Nee, A. Y. C., Fuh, J. Y. H., and Zhang, Y. F. (2003). A modified genetic algorithm for distributed scheduling problems. *Journal of Intelligent Manufacturing*, 14(3-4):351–362.

Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68.

Kahn, K. B., Castellion, G. A., and Griffin, A. (2004). *The PDMA handbook of new product development*. Wiley, New York, Second edition.

Karp, R. M. (1972). *Complexity of Computer Computations*, chapter Reducibility among Combinatorial Problems, pages 85–103. The IBM Research Symposia. Springer.

Kass, G. V. (1980). An exploratory technique for investigating large quantities of categorical data. *Journal of Applied Statistics*, 29(2):119–127.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. *IEEE International Conference on Neural Networks*, 4:1942–1948.

Kirkpatrick, S., Gelatt, C., and Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G., and Brussel, H. V. (1999). Reconfigurable manufacturing systems. *Annals of the CIRP*, 48(2):527–540.

Koulamas, C. and Kyparisis, G. J. (2001). The three stage assembly flowshop scheduling problem. *Computers & Operations Research*, 28(7):689–704.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems)*. The MIT Press.

Laha, D. and Sarin, S. C. (2009). A heuristic to minimize total flow time in permutation flow shop. *OMEGA, The International Journal of Management Science*, 37(3):734–739.

Lee, C. Y., Cheng, T. C. E., and Lin, B. M. T. (1993). Minimizing the makespan in the 3-machine assembly-type flowshop scheduling problem. *Management Science*, 39(5):616–625.

Lee, K., Leung, J. Y. T., Jia, Z., Li, W., Pinedo, M. L., and Lin, B. M. T. (2014). Fast approximation algorithms for bi-criteria scheduling with machine assignment costs. *European Journal of Operational Research*, 238(1):54–64.

Lenstra, J. K., Rinnooy Kan, A. H. G., and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362.

Lin, S. W., Lee, Z. J., Ying, K. C., and Lu, C. C. (2011a). Minimization of maximum lateness on parallel machines with sequence-dependent setup times and job release dates. *Computers & Operations Research*, 38(5):809–815.

Lin, S. W., Ying, K. C., and Huang, C. Y. (2013). Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. *International Journal of Production Research*, 51(16):5029–5038.

Lin, Y. and Li, W. (2004). Parallel machine scheduling of machine-dependent jobs with unit-length. *European Journal of Operational Research*, 156(1):261–266.

Lin, Y., Pfund, M., and Fowler, J. (2011b). Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems. *Computers & Operations Research*, 38(6):901–916.

Liu, H. and Gao, L. (2010). A discrete electromagnetism-like mechanism algorithm for solving distributed permutation flowshop scheduling problem. In *Manufacturing Automation (ICMA), 2010 International Conference on*, pages 156–163, Hong Kong. IEEE.

Low, C., Ji, M., Hsu, C., and Su, C. T. (2010). Minimizing the makespan in a single machine scheduling problems with flexible and periodic maintenance. *Applied Mathematical Modelling*, 34(2):334–342.

Mahdavi, I., Shirazi, B., Cho, N., Sahebjamnia, N., and Ghobadi, S. (2008). Modeling an e-based real-time quality control information system in distributed manufacturing shops. *Computers in Industry*, 59(8):759–766.

Manzini, R., Gamberi, M., Regattieri, A., and Persona, A. (2004). Framework for designing a flexible cellular assembly system. *International Journal of Production Research*, 42(17):3505–3528.

Mehrabi, M. G., Ulsoy, A. G., and Koren, Y. (2000). Reconfigurable manufacturing systems: Key to future manufacturing. *Journal of Intelligent Manufacturing*, 11(4):403–419.

Mikos, W. L., Ferreira, J. C. E., Botura, P. E. A., and Freitas, L. S. (2011). A system for distributed sharing and reuse of design and manufacturing knowledge in the pfmea domain using a description logics-based ontology. *Journal of Manufacturing Systems*, 30(3):133–143.

Minella, G., Ruiz, R., and Ciavotta, M. (2011). Restarted iterated pareto greedy algorithm for multi-objective flowshop scheduling problems. *Computers & Operations Research*, 38(11):1521–1533.

Mishra, A. A. and Shah, R. (2009). In union lies strength: collaborative competence in new product development and its performance effects. *Journal of Operations Management*, 27(4):324–338.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Montgomery, D. C. (2012). *Design and Analysis of Experiments*. John Wiley & Sons, Eight edition.

Moon, C., Kim, J., and Hur, S. (2002). Integrated process planning and scheduling with minimizing total tardiness in multi-plants supply chain. *Computers & Industrial Engineering*, 43(1-2):331–349.

Morgan, J. N. and Sonquist, J. A. (1963). Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, 58(302):415–434.

Mozdgir, A., Fatemi Ghomi, S., Jolai, F., and Navaei, J. (2013). Two-stage assembly flow-shop scheduling problem with non-identical assembly machines considering setup times. *International Journal of Production Research*, 51(12):3625–3642.

Naderi, B. and Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, 37(4):754–768.

Naderi, B. and Ruiz, R. (2014). A scatter search algorithm for the distributed permutation flowshop scheduling problem. *European Journal of Operational Research*, 239(2):323–334.

Nawaz, M., Enscore, E. E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. *OMEGA, International Journal of Management Science*, 11(1):91–95.

Ng, C. T., Barketau, M. S., Cheng, T. C. E., and Kovalyov, M. Y. (2010). "Product Partition" and related problems of scheduling and systems reliability: Computational complexity and approximation. *European Journal of Operational Research*, 207(2):601–604.

Osman, I. and Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *OMEGA, The International Journal of Management Science*, 17(6):551–557.

Palmer, D. S. (1965). Sequencing jobs through a multi-stage process in the minimum total time-a quick method of obtaining a near optimum. *Journal of the Operational Research Society*, 16:101–107.

Pan, Q. K. and Ruiz, R. (2012). Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, 222(1):31–43.

Pan, Q. K. and Ruiz, R. (2013). A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research*, 40(1):117–128.

Pan, Q. K. and Ruiz, R. (2014). An effective iterated greedy algorithm for the mixed no-idle flowshop scheduling problem. *OMEGA, The International Journal of Management Science*, 44(1):41–50.

Pan, Q. K., Wang, L., and Zhao, B. H. (2008). An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. *International Journal of Advanced Manufacturing Technology*, 38(7-8):778–786.

Peklenik, J. (1992). Fms: A complex object of control,. In *The Eight International IFIP WG5.3 PROLAMAT Conference*, pages 1–25, Tokyo, Japan.

Peklenik, J. (1997). Structural and operational complexity of future manufacturing systems. In *Proceedings of the* 29^{th} *CIRP International Seminar on Manufacturing Systems*, pages 1–8, Osaka, Japan.

Pinedo, M. (2009). *Planning and Scheduling in Manufacturing and Services*. Springer-Verlag New York, Second edition.

Pinedo, M. (2012). *Scheduling: theory, algorithms, and systems*. Prentice-Hall, Fourth edition.

Potts, C. N., Janov, S. V. S., Strusevich, V. A., Wassenhove, L. N. V., and Zwaneveld, C. M. (1995). The two-stage assembly scheduling problem: Complexity and approximation. *Operations Research*, 43(2):346–355.

Prosser, P. (1993). Hybrid algorithms for the constraint satisfacation problem. *Computational Intelligence*, 9(3):268–299.

Rasch, D. and Guiard, V. (2004). The robustness of parametric statistical methods. *Psychology Science*, 46(2):175–208.

Ribas, I., Companys, R., and Tort-Martorell, X. (2011). An iterated greedy algorithm for the flowshop scheduling problem with blocking. *OMEGA, The International Journal of Management Science*, 39(3):293–301.

Ridge, E. and Kudenko, D. (2010). Tuning an algorithm using design of experiments. In Bartz-Beielstein, T., Chiarandini, M., Paquete, L., and Preuss, M., editors, *Experimental Methods for the Analysis of Optimization Algorithms*, chapter 11, pages 265–286. Springer, New York.

Rosenau, M. D. (1996). *The PDMA handbook of new product development*. Wiley, New York.

Rudek, R. (2011). Computational complexity and solution algorithms for flowshop scheduling problems with the learning effect. *Computers & Industrial Engineering*, 61(1):20–31.

Ruiz, R. and Andrés-Romano, C. (2011). Scheduling unrelated parallel machines with resource-assignable sequence dependent setup times. *International Journal of Advanced Manufacturing Technology*, 57(5-8):777–794.

Ruiz, R., Şerifoğlu, F. S., and Urlings, T. (2008). Modeling realistic hybrid flexible flowshop scheduling problems. *Computers & Operations Research*, 35(4):1151–1175.

Ruiz, R. and Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Ruiz, R. and Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143–1159.

Siarry, P. and Michalewicz, Z., editors (2008). *Advances in Metaheuristics for Hard Optimization*. Springer-Verlag Berlin Heidelberg.

Sluga, A., Butala, P., and Bervar, G. (1998). A multi-agent approach to process planning and fabrication in distributed manufacturing. *Computers & Industrial Engineering*, 35(3-4):455–458.

Smith, W. E. (1956). Various optimizers for single stage production. *Naval Research Logistics Quarterly*, 3(1-2):59–66.

Stafford, E. F., Tseng, F. T., and Gupta, J. N. D. (2005). Comparative evaluation of MILP flowshop models. *Journal of the Operational Research Society*, 56(1):88–101.

Sun, X., Morizawa, K., and Nagasawa, H. (2003). Powerful heuristics to minimize makespan in fixed, 3-machine, assembly-type flowshop scheduling. *European Journal of Operational Research*, 146(3):498–516.

Sung, C. S. and Kim, H. A. (2008). A two-stage multiple-machine assembly scheduling problem for minimizing sum of completion times. *International Journal of Production Economics*, 113(2):1038–1048.

Thomopoulos, N. T. (2014). *Assembly Line Planning and Control*. Springer International Publishing, New York.

T'Kindt, V. and Billaut, J. (2006). *Multicriteria Scheduling*. Springer-Verlag, Berlin Heidelberg, Second edition.

Tozkapan, A., Kirca, O., and Chung, C. S. (2003). A branch and bound algorithm to minimize the total weighted flowtime for the two-stage assembly scheduling problem. *Computers & Operations Research*, 30(2):309–320.

Tseng, F. T. and Stafford, E. F. (2008). New MILP models for the permutation flowshop problem. *Journal of the Operational Research Society*, 59(10):1373–1386.

Ueda, K. (1992). A concept for bionic manufacturing systems based on DNA-type information. In Olling, G. and Kimura, F., editors, IFIP Transactions *Vol. B-3, Proc. PROMALAT Conference on Human Aspects in Computer Integrated Manufacturing*, pages 853–863, Tokyo, Japan. North-Holland Publishing Co.

Vallada, E. and Ruiz, R. (2010). Genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *OMEGA, The International Journal of Management Science*, 38(1-2):57–67.

Vallada, E., Ruiz, R., and Minella, G. (2008). Minimising total tardiness in the m-machine flowshop problem: A review and evaluation of heuristics and metaheuristics. *Computers & Operations Research*, 35(4):1350–1373.

Vollmann, T. E., Berry, W. L., Whybark, D. C., and Jacobs, F. R. (2005). *Manufacturing Planning and Control for Supply Chain Management*. McGraw-Hill Irwin, Boston, Fifth edition.

Walker, A., Hallam, J., and Willshaw, D. (1993). Bee-havior in amobile robot: the construction of aself-organized cognitive map and its use in robot navigation within a complex, natural environment. In *Proceedings of 1993 IEEE International Conference on Neural Networks (ICNN'93)*, pages 1451–1456, San Francisco, California. IEEE Service Center.

Wang, B. (1997). *Integrated product, process and enterprise design*. Chapman & Hall, London.

Wang, H., Wang, H., and Hu, S. J. (2013a). Utilizing variant differentiation to mitigate manufacturing complexity in mixed-model assembly systems. *Journal of Manufacturing Systems*, 32(4):731–740.

Wang, S. Y., Wang, L., Liu, M., and Xu, Y. (2013b). An effective estimation of distribution algorithm for solving the distributed permutation flowshop scheduling problem. *International Journal of Production Economics*, 145(1):387–396.

Warnecke, H. J. (1993). *The Fractal Company, A Revolution in Corporate Culture*, chapter The Fractal Factory: an Integrating Approach, pages 137–217. Springer Berlin Heidelberg, Germany.

Westkaemper, E. (1997). Manufacturing on demand in production networks. *CIRP Annals - Manufacturing Technology*, 46(1):329–334.

Williams, J. F. (1983). A hybrid algorithm for simultaneous scheduling of production and distribution in multi-echelon structures. *Management Science*, 29(1):77–92.

Xiong, F., Xing, K., Wang, F., Lei, H., and Han, L. (2014). Minimizing the total completion time in a distributed two stage assembly system with setup times. *Computers & Operations Research*, 47:92–105.

Xu, X., Xu, Z., and Gu, X. (2011). An asynchronous genetic local search algorithm for the permutation flowshop scheduling problem with total flowtime minimization. *Expert Systems with Applications*, 38(7):7970–7979.

Yang, W. H. and Liao, C. J. (1999). Survey of scheduling research involving setup times. *International Journal of Systems Science*, 30(2):143–155.

Ying, K. C. (2012). Scheduling identical wafer sorting parallel machines with sequence-dependent setup times using an iterated greedy heuristic. *International Journal of Production Research*, 50(10):2710–2719.

Yokoyama, M. (2004). Scheduling for two-stage production system with setup and assembly operations. *Computers & Operations Research*, 31(12):2063–2078.

Yokoyama, M. (2008). Flow-shop scheduling with setup and assembly operations. *European Journal of Operational Research*, 187(3):1184–1195.

Zachariadis, E. E. and Kiranoudis, C. T. (2010). A strategy for reducing the computational complexity of local search-based methods for the vehicle routing problem. *Computers & Operations Research*, 37(12):2089–2105.

Zhang, Y. and Li, X. (2011). Estimation of distribution algorithm for permutation flow shops with total flowtime minimization. *Computers & Industrial Engineering*, 60(4):706–718.

Zobolas, G. I., Tarantilis, C. D., and Ioannou, G. (2009). Minimizing makespan in permutation flowshop scheduling problems using a hybrid metaheuristic algorithm. *Computers & Operations Research*, 36(4):1249–1267.