Scheduling in semi parallel flow shop with a final synchronizing operation

Irce Fernandes Gomes Guimarães

Supervisor: **Prof. Dr. Maurício Cardoso Souza**
Professeur des Universités Farouk Yaloui

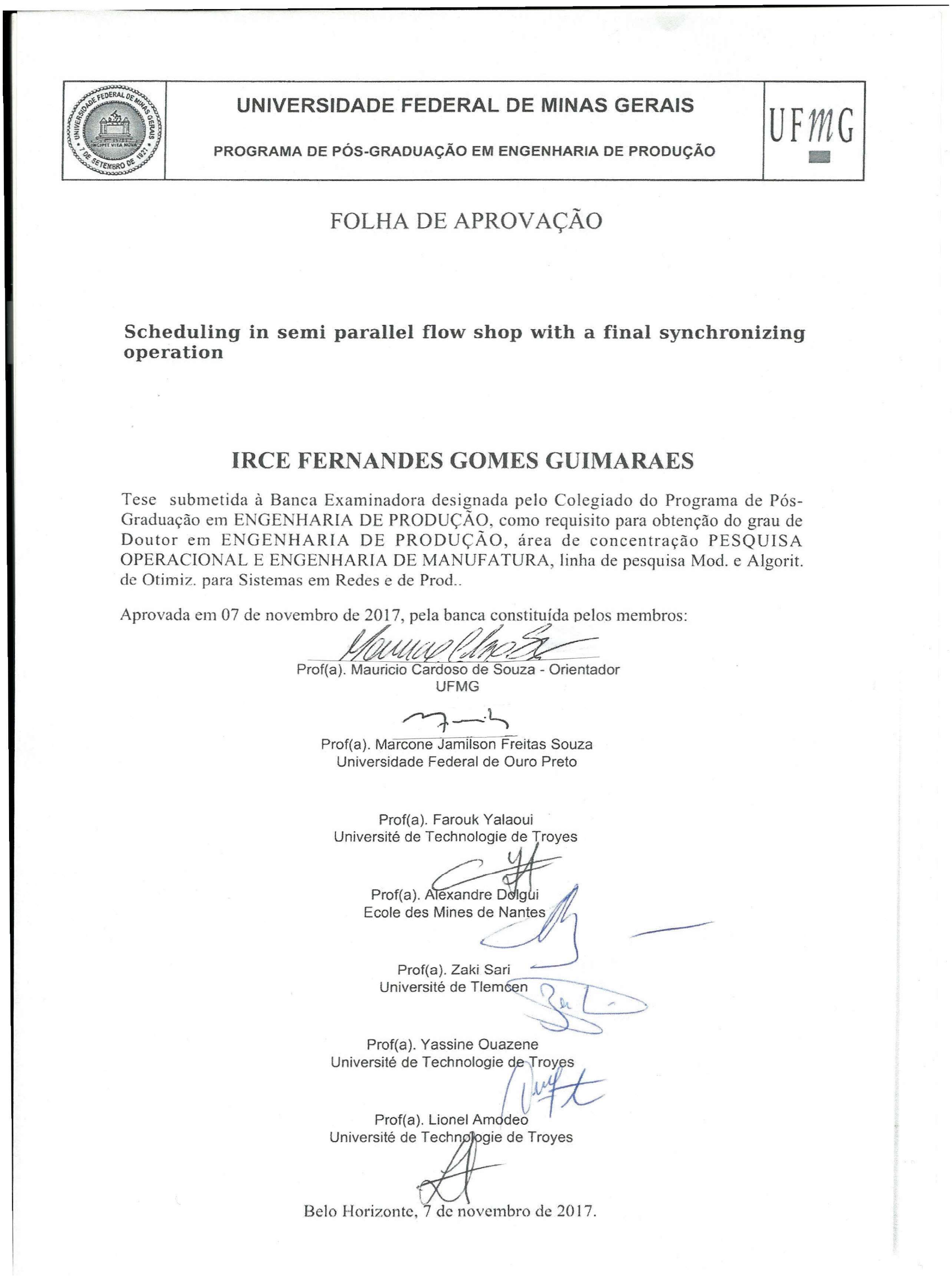
Postgraduate program in Production Engineering,
Ecole Doctorale Sciences et Technologies
Federal University of Minas Gerais - UFMG.
University of Technology of Troyes - UTT.

This dissertation is submitted for the degree of
Doctor in Production Engineering - UFMG and Doctor in Systems Optimization and Dependability - UTT
February 2018

"Our deepest fear is not that we are weak. Our deepest fear is that we are powerful beyond measure. It is our light, not our darkness that most frightens us. We ask ourselves, who am I to be brilliant, gorgeous, talented, fabulous? Actually, who are you not to be? You are a child of God. Your playing small does not serve the world ... As we are liberated from our own fear, our presence automatically liberates others."

Nelson Mandela

I would like to dedicate this thesis to my loving parents.



UNIVERSIDADE FEDERAL DE MINAS GERAIS

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

UFMG

FOLHA DE APROVAÇÃO

Scheduling in semi parallel flow shop with a final synchronizing operation

IRCE FERNANDES GOMES GUIMARAES

Tese submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em ENGENHARIA DE PRODUÇÃO, como requisito para obtenção do grau de Doutor em ENGENHARIA DE PRODUÇÃO, área de concentração PESQUISA OPERACIONAL E ENGENHARIA DE MANUFATURA, linha de pesquisa Mod. e Algorit. de Otimiz. para Sistemas em Redes e de Prod..

Aprovada em 07 de novembro de 2017, pela banca constituída pelos membros:

[illegible]

Prof(a). Mauricio Cardoso de Souza - Orientador
UFMG

[illegible]

Prof(a). Marcone Jamilson Freitas Souza
Universidade Federal de Ouro Preto

Prof(a). Farouk Yalaoui
Université de Technologie de Troyes

[illegible]

Prof(a). Alexandre Dolgui
Ecole des Mines de Nantes

[illegible]

Prof(a). Zaki Sari
Université de Tlemcen

[illegible]

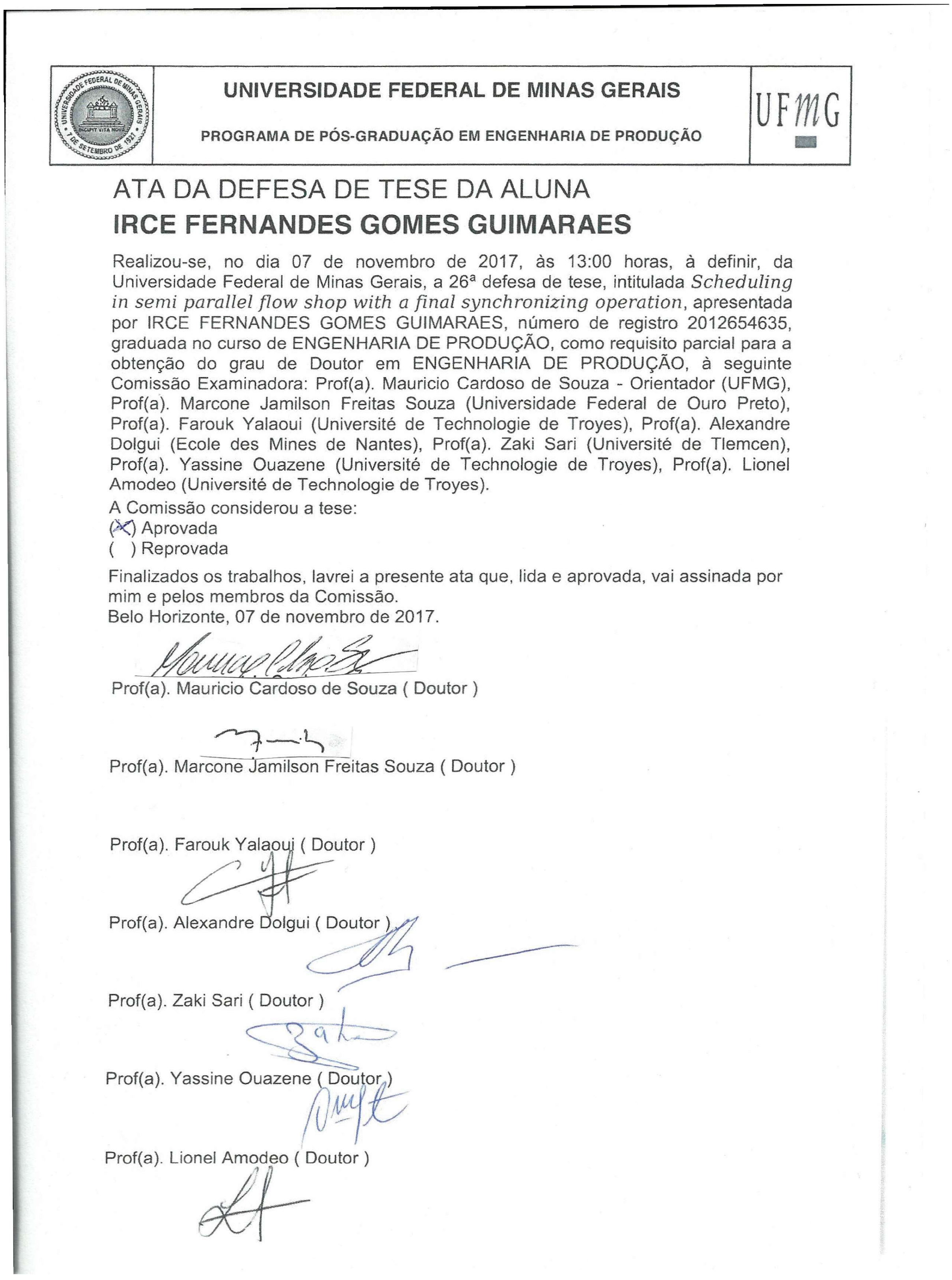
Prof(a). Yassine Ouazene
Université de Technologie de Troyes

[illegible]

Prof(a). Lionel Amodeo
Université de Technologie de Troyes

[illegible]

Belo Horizonte, 7 de novembro de 2017.



UNIVERSIDADE FEDERAL DE MINAS GERAIS

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

UFMG

ATA DA DEFESA DE TESE DA ALUNA
IRCE FERNANDES GOMES GUIMARAES

Realizou-se, no dia 07 de novembro de 2017, às 13:00 horas, à definir, da Universidade Federal de Minas Gerais, a 26ª defesa de tese, intitulada *Scheduling in semi parallel flow shop with a final synchronizing operation*, apresentada por IRCE FERNANDES GOMES GUIMARAES, número de registro 2012654635, graduada no curso de ENGENHARIA DE PRODUÇÃO, como requisito parcial para a obtenção do grau de Doutor em ENGENHARIA DE PRODUÇÃO, à seguinte Comissão Examinadora: Prof(a). Mauricio Cardoso de Souza - Orientador (UFMG), Prof(a). Marcone Jamilson Freitas Souza (Universidade Federal de Ouro Preto), Prof(a). Farouk Yalaoui (Université de Technologie de Troyes), Prof(a). Alexandre Dolgui (Ecole des Mines de Nantes), Prof(a). Zaki Sari (Université de Tlemcen), Prof(a). Yassine Ouazene (Université de Technologie de Troyes), Prof(a). Lionel Amodeo (Université de Technologie de Troyes).

A Comissão considerou a tese:

[✓ Aprovada]

( ) Reprovada

Finalizados os trabalhos, lavrei a presente ata que, lida e aprovada, vai assinada por mim e pelos membros da Comissão.

Belo Horizonte, 07 de novembro de 2017.

[illegible]

Prof(a). Mauricio Cardoso de Souza (Doutor)

[illegible]

Prof(a). Marcone Jamilson Freitas Souza (Doutor)

Prof(a). Farouk Yalaoui (Doutor)

[illegible]

Prof(a). Alexandre Dolgui (Doutor)

[illegible]

Prof(a). Zaki Sari (Doutor)

[illegible]

Prof(a). Yassine Ouazene (Doutor)

[illegible]

Prof(a). Lionel Amodeo (Doutor)

[illegible]

Acknowledgements

Many people have contributed to the elaboration of this thesis. Certainly, it would be impossible to name everyone, because the list is immense, so I am afraid of committing injustice. I would manifest my deep gratitude for their support. Nonetheless, there are those who I want to outstand, to thank them and to express my acknowledgement.

First and foremost, I wish to express my sincere gratitude to Professor Maurício Cardoso de Souza of the Federal University of Minas Gerais (UFMG) for believing in me, encouraging and giving me this great opportunity and above all for his valuable supervision and patience throughout my PhD study; and to Professor Farouk Yalaoui of the Université de Technologie de Troyes (UTT) for his availability, encouraging, advices, and for welcoming me at the LOSI Laboratory.

I would also like to thank the rapporteurs of these organizations: Pr. Alexandre DOLGUI, Head of Departement, Production and Computer Science Dept, Ecole des Mines de Nantes, Pr. Zaki SARI, DSc Directeur du Laboratoire de Productique de Tlemcen (MELT) Université Aboubekr Belkaïd de Tlemcen Algérie for their attention and attentive reading of my manuscript.

Thanks to the members of the jury who accepted and dedicated their time to the evaluation of this thesis: Pr. Dr. Marcone, JAMILSON FREITAS SOUZA, Professor, Federal University of OURO PRETO, Brazil, Pr. Lionel, AMODEO Head of the Engineering Degree in Industrial Engineering, UTT and Dr. Yassine, OUAZENE Maitre de Conferences of UTT.

I am grateful to the professionals of the Federal University of Minas Gerais (UFMG), the Troyes University of Technology (UTT) and the Federal University of Ouro Preto (UFOP) who have contributed directly or indirectly to this study and headed me beyond the borders of my knowledge.

I cannot forget my colleagues of the Graduate Laboratory of Production Engineering of Federal University of Minas Gerais, of the Industrial Systems Optimization Laboratory of Troyes University of Technology and also of the production engineering department of UFOP.

To the Coordination of Improvement of Higher Level People (Capes) who granted me an abroad research scholarship allowing me to work full time in this thesis.

I also want to thank Gizelia Maria Machado Adeodato for revising my writings.

Finally, I thank God for my life, for guiding and lightening me in most difficult moments. I thank a lot my mother Maria Geralda, my father Sebastião (now in memory), my siblings Maria Teresa, Mírian, Italo, Josiano (in memory), my nephew Franklin, and all my close friends, who have pushed and helped me to overcome my hard moments.

Abstract

This study deals with a variant of the flow shop problem motivated by a practical situation. In this environment, there is an assembly line composed by two parallel semi- lines and independent operation. The first semi-line has a number q_1 of machines and the second has a number q_2 of machines. At the end of the two semi-lines there is a machine in charge of the union of the semi-products from the two semi-lines, each semi-line is devoted to different tasks, since each job requires operations in each machine of the semi-lines with different processing time. The tasks of one semi-line do not depend on the completion of the task of the other semi-line, so that a job is processed in parallel in each semi-line. The sequence of jobs in each parallel semi-line must be the same, although the first task of a job in each semi-line does not have to start at the same time. The final synchronizing operation can only start when the operations within both semi-lines have been finished. The solution to this problem is to determine one possible sequence of jobs that optimizes a given performance measure. In this sense, the objective of this study is to model the flow shop problem in the proposed production environment and to solve it with a mathematical model, heuristic and metaheuristic, considering the makespan as performance measure. In a first approach, a mixed integer linear programming model was defined. Due to the complexity of the problem, it was also solved by Johnson's rule, the *NEH* heuristics and by the Iterated Local Search, Simulated Annealing and *GRASP* metaheuristics. Finally, an extensive computational experiment was performed and a comparison of the proposed methods was made. The hybrid method that used the *NEH* algorithm with *GRASP* metaheuristic presented better quality than the other methods.
keywords: Scheduling (Management), Computer scheduling , Heuristic, Metaheuristics, Synchronization

Extended Abstract

Introduction

Different models of production scheduling, controlling and planning are focus of scientific studies aiming to guarantee better performance of the productive systems. In this sense, it can be said that the production scheduling is an important activity and is considered a broad and diversified field by many researchers. In this activity, it is necessary to observe the priorities of the operations to be processed and to optimize the use of resources to establish the best time to start and finish each activity. This optimization can bring gains such as a reduction of the unproductive time, leading to a better system utilization, reliability beyond reducing the production costs. Given the importance of this activity in many types of manufacturing system, a variant of the flow shop problem was chosen for this study, since this is a common problem in many industrial sectors that requires efficient methods for its resolution.

The model of flow shop scheduling in study

The model is an assembly line consisting of two parallel semi-lines and a final synchronization operation. Each semi-line produces one of the halves of the final product that is assembled into a single product in the final synchronization operation. The order of the halves of the final products in each semi-line must be equal and must be followed in the final synchronization operation as well. The first semi-line has a number q_1 of machines and the second semi-line has a number q_2 of machines. There is a machine in charge of the union of the semi-products at the end of the two semi-lines. Each job requires operations in each machine of the semi-lines with different processing time. An operation in the machine of a semi-line does not need to start at the same time of an operation in the machine of the other semi-line. However, the final synchronization operation for a product can only be started when the operations of its halves in both semi-lines are completed. Given this arrangement, it is possible to classify this environment as a special case of flow shop. The objective of this study is to achieve the optimal or near-optimal solution to minimize the total processing time. A mathematical formulation model, heuristics and metaheuristics were proposed to solve the problem.

This system in a real problem

This research was motivated by a practical situation in the process of manufacturing of a company that works in the electrical and electronic sector of protection system for residential and commercial areas. Among the products manufactured by the company, the counters, the

residential and industrial circuit breakers can be pointed out. The industry has four main manufacturing sectors: plastic injection, stamping, welding and assembly. The processing sequence of the company's production items is similar for all products. The welding system of this product was taken as the starting point for this study since this product has a great impact in the production schedule of the company.

Objective and method

The main objective of this study was to model the flow shop schedule problem in parallel semi-lines with a synchronization operation at the end and to solve it with a mathematical model and algorithms such as heuristics and metaheuristics. In a first approach, a mixed integer linear programming was used based on studies already developed for the flow shop problem in other types of environments. Due to its complexity, the problem has been solved by heuristics and metaheuristics. Methods using Johnson's rule, NEH algorithm and iterated local search (*ILS*), simulated annealing (*SA*) and *GRASP* metaheuristics were also used to solve the problem.

The different methods were tested in 240 different instances. They are composed by combinations of number of jobs x number of machines of the semi-line 1 x number of machines of the semi-line 2. The first 130 instances were created considering that the two semi-lines have the same number of machines. The other 110 instances were created considering different numbers of machines in the semi-lines. The results of the mathematical model were achieved by using the software $CPLEX 12:6:1$ and the algorithms were implemented in $C++$. The statistical analysis of the results was achieved by using Minitab software $17:2:1$. The absolute relative deviation (*GAP*) and the standard deviation (*SD*) of the *GAP*s were used to compare the results.

Approaches

Mathematical approach

The mixed integer linear programming model was defined based on studies by Stafford [100] and by Wagner [104]. The model required adaptations to the problem considering two parallel semi-lines followed by the final synchronization operation. The operations in the parallel machines were grouped and the processing time of the operation corresponding to the set of operations of the slower group was adopted. At the end of each semi-line, a dummy machine was included (considered as a synchronization machine in the model).

These two dummy machines represent the independent processing of the last operation. The existence of two artificially independent semi-lines allowed the determination of the highest value of processing time between them. The model was efficient for smaller instances. It presented difficulties to find out the optimal solutions in a viable computational time for larger instances.

Approach using Johnson's algorithm and the *NEH* heuristic

The second approach used to solve the problem considered two adaptations of Johnson's algorithm (Zhang and Van de Velde [109], Allahverdi et al. [7]), and three adaptations of the *NEH* algorithm (Nawaz et al. [66], Guimarães et al.[38]).

The general principle of the adaptations of Johnson's algorithm considers the studied system as a flow shop of two machines. The processing time in the machines of the semi-lines was considered as the processing time in the first Johnson's machine and the processing time of the synchronization machine as the processing time of the second Johnson's machine and then Johnson's algorithm is applied. The first adaptation of Johnson's algorithm takes the longest processing time of each job in all the machines of the two semi-lines for the first Johnson's machine and the processing time of the synchronization machine for the second Johnson's machine. The second adaptation of Johnson's algorithm considers the highest average processing time between the semi-lines for the first Johnson's machine and the processing time of the synchronization machine for the second Johnson's machine.

Three adaptations were made for the *NEH* heuristics: the NEH_{av} algorithm, considering the average processing time of the jobs between the parallel machines of the lines; the NEH_{hi} algorithm, considering the longest processing time of the jobs between the parallel machines; and the NEH_{sep} algorithm, considering each semi-line separately, including the synchronization machine. The general principle of the adaptations was to reduce the two semi-lines in a single line and to apply the *NEH* algorithm.

A comparison between the results of the mean relative deviation (*GAP*) and standard deviation (*SD*) generated by the adaptations of the *NEH* and Johnson's algorithm was made. The results for the algorithms NEH_{sep} had smaller mean relative deviations for environments with the same number of machines.

In relation to the environment with different number of machines, the NEH_{sep} showed the best results. However, these results could be improved in order to reach or to be closer to the optimal solution. Therefore, metaheuristics were used to solve the problem.

Approach using iterated local search

In this part of the study the iterated local search algorithm (*ILS*) was considered. The neighborhood space was explored in order to improve the objective function. This algorithm was based on studies by Ruiz and Stutzle [88] and Cavalcanti et al.[17] and it starts with the *NEH* algorithms considering the separated semi-lines and Johnson's adaptation considering the mean of the semi-lines. This heuristic yielded better results than those presented by the previously used methods. The method that yielded the lowest mean relative deviation considering the optimal result was the $Jonh_{av}$ method with *ILS*.

Approach using simulated annealing

Two initial methods of solution were considered in this algorithm. The simulated annealing metaheuristic was an adaptation of the Hurkala and Hurkala [42] and Nearchou [67] method and used the following parameters: initial temperature $Ti = 6000$, cooling factor $\alpha = 0,2, 0,5$ and $0,95$ and the final temperature $= 0,0001$. The algorithm generated changing in the sequence of the jobs and provided another neighbor solution. A new changing was made at each iteration and a new solution was generated. The generated solutions underwent an evaluation process to verify if the objective of minimizing the makespan has been reached. The *ILS* algorithm was used to refine the best solution upon reaching the maximum number of iterations.

The tests performed with this algorithm showed better results than those presented only with the *ILS* algorithm. The best mean relative deviation (*GAP*) for the environment with the same number of machines was the method that used NEH_{sep} for the initial solution with $SA - ILS$ with $\alpha = 0.95$. For the environment with different numbers of machines, the best *GAP* was the method starting with Johnson's algorithm with the average time of the semi-lines with SA and *ILS* with $\alpha = 0.95$.

Approach using Greedy Randomized of Adaptive Search-*GRASP*

GRASP algorithm for the the flow shop scheduling problem in parallel semi-line with a final synchronization operation consisted of two phases: the construction phase of the initial solution and the local search phase or the solution improvement. The first phase consists of constructing a feasible solution. After that, the acquired solution in the previous stage underwent a search in the neighborhood to achieve the local minimum. The best solution was stored as a partial result. At the end of all iterations, the best of the partial results was adopted. Arroyo et al. ([10] and Resende and Ribeiro [83]).

The *GRASP* method was an adaptation of the original *GRASP* algorithm of the NEH_{sep} algorithm or $John_{av}$ algorithm. In the construction phase, the restricted list was generated by selecting $\alpha\%$ from the candidate list. In this study, three alpha values (0.2 %;0.3 % and 0.5 %) were evaluated. The results of each α value were the best in relation to all the approached heuristic and metaheuristic methods. For the environment with the same number of machines, the best average relative deviation was for the $GRASP - NEH_{sep}$ with $\alpha = 0.5$ %. For the environment with different numbers of machine in each parallel semi-line, the best average relative deviation was with the $GRASP - NEH_{sep}$ with $\alpha = 0.5$.

Final remarks

This research focused on the development of optimization methods to solve a variant of the flow shop problem with parallel and a final synchronization operation. This is a common problem in many industrial sectors that requires efficient methods for its resolution. Therefore, this study arose from a practical situation in a welding process of an industry that manufactures products for the electrical electronic system. This study aimed to model the flow shop problem with parallel semi-lines with a final synchronization operation at the end of the semi-lines to solve it with a mathematical model, heuristic and metaheuristic.

In a first approach, a model of mixed integer linear programming was defined based on studies already developed for the flow shop problem in other types of environments. The model required adaptations for the problem considering two parallel semi-lines followed by a final synchronization operation. The operations in the parallel machines were grouped, and the processing time of the corresponding operation to the set of operations of the slower group was adopted. A dummy machine, considered as synchronization machine in the model, was included at the end of each semi-line. These two dummy machines represent the independent processing of the last operation. The existence of two artificially independent semi-lines allowed the determination of the highest value of the processing time between them. The model was efficient for smaller instances, but it presented some difficulties to find out the optimal solutions in a feasible computational time for larger instances. Due to its complexity, the problem was also solved by heuristics and metaheuristics. Based on computational results, it was possible to prove that the construction of methods through heuristics and metaheuristics yielded feasible results for the studied problem. The used algorithms showed good options to solve larger instances. The solution of the problem through methods such as Johnson's rule, the heuristic method as the *NEH* algorithm, and the metaheuristics as the *ILS*, the simulated annealing and the *GRASP* showed the behavior of these methods in the studied environment and the possibility of finding optimal solution. In addition, among the methods NEH_{hi}, NEH_{av}, NEH_{sep}, $SA-NEH-ILS$, $SA-John-ILS$; $GRASP-NEH$,$GRASP-Jonh$, the GRASP method with 0.5% achieved the most amount of optimal results and the lowest GAP. The lowest average GAP was 0.35% and the lowest standard deviation was 0.15% for the environment with the same number of machines in the semi-lines and 0.37% and 0.33% for environments with different number of machines in each semi-line.

Further directions of research may involve more complex problems and multi-objective functions, considering the same environment. The performance of the algorithm with the development of other exploration techniques of the search space and the test of other metaheuristics for this problem (genetic algorithm and Tabu Search) are subject matters to be considered in other researches.

Résumé

Cette étude est une variante du problème flow shop motivée par une situation pratique. Dans ce contexte, il y a une ligne d'assemblage composée de deux demi-ligne parallèles avec des activités indépendantes. La premiére demi-ligne a un nombre de machines q_1 et le second demi-ligne a un nombre de machines q_2 . A la fin des deux demi-ligne il y a une machine responsable par l'union des produits des deux demi-lignes. Chaque demi-ligne est dédiée à différentes tâches, en raison chaque travail nécessite des opérations dans chacune des machines des demi-ligne avec temps de transformation différents. Les tâches d'une demi-ligne ne dépendent pas de la réalisation d'une autre tâche dans autre demi-ligne, de sorte qu'une tâche est traitée en parallèle dans chaque demi-ligne. La séquence des travaux dans chaque demi-ligne paralléle devrait être la même, bien qu'une tâche d'une demi-ligne n'ait pas de besoin de commencer en même temps a une autre demi-ligne. La dernière operation de synchronisation ne peut pas être démarrée que lorsque les opérations dans les deux demi-lignes aient ont été complétées. La solution à ce problème est de déterminer une séquence de travail possible pour optimiser le makespan. En ce sens, l'objectif de cette étude est de modéliser le problème flow shop dans l'environnement de production propose et les résoudre avec des algorithmes spécialisés. Une première approche, nous avons défini un modelé de programmation linéaire mixte en nombres entiers et au vu des la complexité du problème, il a également été résolu par la règle Johnson, heuristiques *NEH* et la recherche locale Meta-heuristiques Iterated, Recuit Simule et *GRASP*. Enfin, une importante campagne de tests a été menée et une comparaison des méthodes proposées a été réalisée. Le méthode hybride qui utilise l'algorithme *NEH* avec le *GRASP* a démontré sa supériorité par rapport autres méthodes proposées.

mots-clés: Ordonnancement (Gestion), Ordonnancement (Informatique), Heuristique, Métaheuristiques, Synchronisation

Resumo

Este estudo trata de uma variante do problema do flow shop motivado por uma situação prática. Neste ambiente, existe uma linha de montagem composta por duas semi-linhas paralelas em operações independentes. A primeira semi-linha tem número q_1 de máquinas e a segunda tem número q_2 de máquinas. No final das duas semi-linhas, existe uma máquina encarregada pela união dos semi-produtos manufaturados pelas duas semi-linhas. Cada semi-linha é dedicada a diferentes tarefas, uma vez que cada tarefa exige operações em cada máquina das semi-linhas com diferentes tempos de processamento. As tarefas de uma semi-linha não dependem da conclusão das tarefas da outra semi-linha, de modo que uma tarefa é processada em paralelo em cada semi-linha. A sequência das tarefas em cada semi-linha paralela deve ser a mesma, embora a primeira tarefa em cada semi-linha não precise iniciar ao mesmo tempo. A operação de sincronização final só pode ser iniciada quando as operações nas semi-linhas forem concluídas. A solução para este problema é determinar uma possível sequência de tarefas que otimizem uma dada medida de desempenho. Nesse sentido, o objetivo deste estudo é modelar o flow shop no ambiente de produção proposto e resolvê-lo com algoritmos especializados. Em uma primeira abordagem, definiu-se um modelo de programação linear inteira mista, e em vista da complexidade do problema, ele também foi resolvido pela regra de Johnson, heurísticas de *NEH* e pelas metaheurísticas *Iterated Local Search*-(*ILS*), *Simulated Annealing*-(*SA*) e *Greedy Randomized Adaptive Search Procedure*-(*GRASP*). Finalmente, uma extensa experimentação computacional foi realizada e uma comparação entre os métodos propostos foi feita. O método híbrido que utilizou o algoritmo *NEH* com a metaheurística *GRASP* apresentou qualidade superior aos outros métodos propostos.

Palavras chave: Sequenciamento (Gestão), Sequenciamento (Informática), Heurística, Metaheurística, Sincronização

Table of contents

List of figures

List of tables

List of Abreviations

GA: **G**enetic **A**lgorithm
TS: **T**abu **S**earch
GRASP: **G**reedy **R**andomized **A**daptive **S**earch **P**rocedure
LS: **L**ocal **S**earch
CRL: **C**andidate **R**estrict **L**ist
SA: **S**imulated **A**nnealing
VND: **V**ariable **N**eighborhood **D**escent
SPIRIT: **S**equencing **P**roblem **I**nvolving a **R**esolution by **I**ntegration of the **T**abu search technique
HFS: **H**ybrid **F**low **S**hop
VNS: **V**ariable **N**eighborhood **S**earch
NEH: **H**euristic of **N**awaz, **E**nscore and **H**am
MIP: **M**ixed-**I**nteger **P**rogramming
FIFO: **F**irst **I**n **F**irst **O**ut
NSGA: **N**ot **D**ominated **G**enetic **A**lgorithm
SPEA: **S**trength **P**areto and **E**volutionary **A**lgorithm
ACO: **A**nt **C**olony **O**ptimization
GALS: **G**enetic **A**lgorithm with **L**ocal **S**earch
DPFSP: **D**istributed **P**ermutation **F**low **S**hop **P**roblem
ILS: **I**terated **L**ocal **S**earch
VND: **V**ariable **N**eighborhood **D**escent
NEH$_{av}$: **Heuristic NEH** with the average processing time in the machines in the semi-lines
NEH$_{hi}$: **Heuristic NEH** with the longest processing time in the machines in the semi-lines
NEH$_{sep}$: **Heuristic NEH** considering the semi-lines separately
John$_{av}$: **Johnson 's Rule** considering the average processing time in the semi-lines
John$_{hi}$: **Johnson 's Rule** considering the longest processing time in the semi-lines

Chapter 1

Introduction

In this chapter, it will be presented an overview on planning and scheduling of the production, the welding process of a company that works in the sector of electrical-electronic systems that motivated this research. It will also be presented the flow shop and the system under study. After, some characteristics of the problem, the objective, the importance and the structure of this study will be discussed.

1.1 General overview of production sequencing

Different models of production planning and scheduling are focus of scientific studies, often to ensure a good performance of the production system. In this sense, it can be stated that the production schedule is one of the main activities of a production system and is considered a very broad and diversified field by many researchers.

According to Cheng et al. [21], this activity can be generically defined as allocation of the available resources for the jobs processing in a time horizon. It consists in developing models to achieve the best programming solutions, in order to optimize the work flow through the system. These models aim to eliminate the bottlenecks and to adjust the step priorities by observing the losses and the overloads among the production centers and the occupation of the available workforce.

For this activity, it is necessary to observe the operations priorities to be carried out in order to establish the best time to start and finish each activity and to assess the best use of the resources that generally look for optimizing a given measure of performance. Onwubolu [71] presents different measures of performance to determine the best sequencing. Minimizing the makespan, the delay of the activities, the equipment and workforce idle time, among others are the main measures. However, such measures of performance often come into conflicts. The comprehension and the implications of such measures are extremely important to establish the suitable criteria for the problems of sequencing. It is possible to get better results related to the due date and the lower cycle time when there are clear and no ambiguity goals.

Methods of the operational research aid the decision making to minimize such conflicts in the production scheduling. According to Aarts and Lenstra [1], the production scheduling has a broad class of problems. This research field encompasses many problems of decision making that are implemented by using mathematical models. For the production scheduling there are models at the literature that aim to answer how much and in which order (sequence) that each product must be produced to minimize a given measure of performance. Although the optimal solution for a finite problem can sometimes be found by simple enumeration, in practice this task is often impossible, especially for big problems in that the number of possible solutions can be extremely high. Therefore, different models of production planning and scheduling have been developed, see Pinedo [78].

In this work, we are interested in the production scheduling in environment flow shop, which is characterized by a flow of the jobs that are processed by several machines in series. The jobs follow a similar technological path through the machines. According to Widmer and Hertz [106], many heuristics have been proposed since Johnson [45] introduced the resolution for the flow shop problem considering two and three machines. Most of the studies founded in the literature address the classical flow shop problem with makespan criterion. The classic problem of flow shop scheduling consists of n jobs to be processed among a set of m machines arranged in series. The main feature of this problem is that the jobs must have the same technological sequence of which each job has a specific processing time in each machine Gupta and Chauhan [39]. The objective is to determine the one that optimizes a certain performance measure among the possible sequences. The most commonly used is the total completion time minimization (makespan). The major difficulty for this problem is to find an optimal solution in polynomial time, since this problem is known as NP-hard (RUIZ and MAROTO [87]).

Thus, some studies have been developed in the flow shop scheduling literature considering exact techniques such as mathematical programming. Examples that can be pointed out are the studies by Naderi et al. [64], who proposed a model of mixed integer linear programming to minimize the makespan, and the total delay in a flow shop environment. Frasch et al. [30] also presented an approach of MIP modelling for flow shop problems with a limited number of intermediate buffers. They considered three objective functions to minimize: the makespan, the sum of completion time and the number of strands. Ronconi and Birgin [86] proposed a mixed integer linear programming (MIP) to minimize the number of early arrival and the total delay of jobs through an assessment in terms of computational cost, the flow shop problem with unlimited and zero buffer. Hnaien et al. [41] developed two models of mixed-integer programming (*MIP*) to the problem of flow shop scheduling of two-machines with unavailability constraint in the first machine in order a minimize the makespan. The authors proposed a branch-and-bound algorithm based on a set of new lower bounds and heuristics. Computational results showed the impact of the unavailability of the initial time period and the difficulty to solve the problem. The branch-and-bound algorithm had better results than the two *MIP* models.

Other studies also suggested exact methods for the flow shop. However, the exact methods generally take too much time of the *CPU* to solve a problem with a large number of jobs due to the computational complexity. Heuristic and metaheuristic approaches are considered in many studies in the literature to solve larger instances. Examples of heuristics that are used in some studies are those by Johnson [45] and Nawaz et al. [66]. These methods have the advantage of finding viable solutions in reasonable computational time.

Johnson's algorithm has been adopted by Allahverdi et al. [7] to minimize the makespan in flow shop context with two-machines. The authors proposed five different algorithms. Johnson's algorithm was used based on the average of the lower and upper time limits of the job processing. Computational experiments indicated that the modified Johnson's algorithm can be applied with success in programming environments with random and limited processing time. Pan et al. [74] introduced the scheduling resolution problem in a flow shop environment with zero-buffer. In this study, they analyzed the heuristics by Nawaz et al.[66] that exploited specific characteristics of the problem such as NEH. They found good solutions with lower computational effort.

Another example is the study by Allaoui and Artiba [8] who investigated the problem of hybrid flow shop scheduling with two stages, comprising a single machine in the first stage and m machines in the second stage to minimize the makespan. Fernandez-Viagas and Framinan [29] proposed tie-breaking mechanisms with NEH heuristic to minimize total tardiness of the permutation flow shop scheduling problem. They tested the NEH_{edd} heuristic observing the optimization, the improvement and the overall performance. Other articles present metaheuristics to solve the problem of flow shop scheduling. Santosa and Rofiq [91] studied the problem of hybrid flow shop (HFS) with m machines at each stage. The authors developed a modified simulated annealing algorithm to minimize the makespan and the total delay. Among the computational tests, the modified simulated annealing performed better compared to the regular simulated annealing, especially in bigger problems.

Low et al. [54] proposed a robust simulated annealing heuristic for flow shop scheduling problems with the objective of minimizing the makespan. A mechanism that recorded the characteristics of good solutions was designed and introduced into simulated annealing to make the searching procedure more robust. The performance of the proposed algorithm was compared with some existing searching algorithms in two benchmark problem sets. The results showed that the algorithm achieves solutions near to optimal. Another study was by Prabhaharan et al. [79] who researched the problem of permutation flow shop to minimize the makespan. They presented results of the jobs sequencing with the the greedy randomized adaptive search procedure metaheuristic (*GRASP*) and then the *NEH* algorithm. The computational results showed that the *GRASP* algorithm overcame the traditional *NEH* algorithm. Shahul et al. [96] dealt with the flow shop problem with m machines in order to minimize the weighted sum of the makespan and the maximum delay. One of the used techniques was the *GRASP* Bi-criteria algorithm. The proposed algorithm was evaluated using reference problem based on Taillard [101] and compared with the results of an existing model using Simulated Annealing. The $Bi-GRASP$ algorithm was effective for small problems.

Based on the studies mentioned above, this thesis presents an analysis of a variant of flow shop problem motivated by a practical application of production sequence in a welding line in an electrical-electronics industry. The main objective is to model the flow shop scheduling problem in parallel semi-line with a final synchronization operation and to solve it using mixed integer programming model, Johnson's algorithm, the *NEH* heuristic and the metaheuristics of local search, simulated annealing and Grasp.

1.2 Problem description

This research was motivated by a practical situation in a manufacturing process of a company that works at the sector of electrical-electronic system protection for residential and commercial fields introduced by Vaz and Araki [103].

1.2.1 The welding process of the electrical-electronic system in study

The company under study is in the state of Minas Gerais, Brazil. Among its manufactured products, it can be pointed out the meters and residential and industrial circuit breakers. The company has four main manufacturing sectors: plastic injection, stamping, welding and assembly.

The processing sequence of the company 's production items is similar for all products. Among its many products, there is a manufacturing process for residential circuit breaker. This product was taken as an example for this study since it has a great impact on the

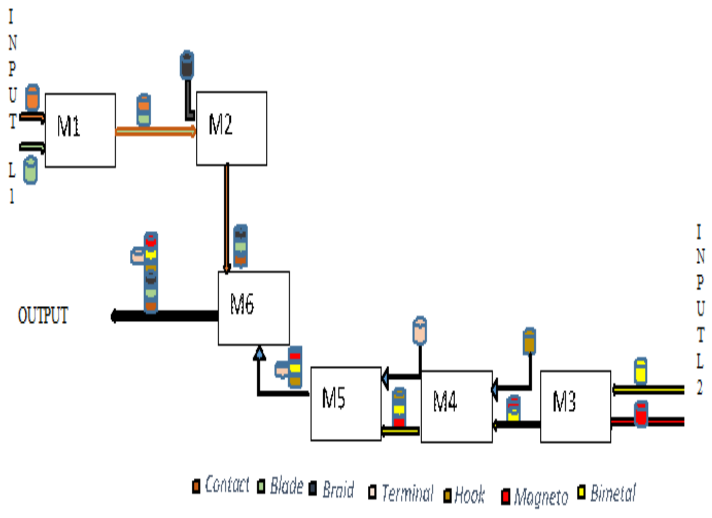


Fig. 1.1 General welding process of circuit breakers

company's production scheduling. It is a product of high amperage so its components can only be welded in one of the four lines of the welding area. It must be taken into account that the higher the amperage the longer the time of welding operation processing. The production of this welding line is a critical factor at the circuit breaker manufacturing since the high amperage line has an installed capacity strongly occupied by the production. It consists of six welding machines by resistance spot welding, each performing a specific activity. It is possible to observe two semi-lines in parallel operation, the first with two machines and the second with three machines. The union of the semi-product arose from the two semi-lines is made by the sixth machine. Figure 1.1 shows this processing in which the machine $1(M1)$ makes the connection between the contact and the blade; the machine $2(M2)$ links the blade and the braid; the machine $3(M3)$ joins the magneto and the bimetal; the machine $4(M4)$ links the bimetal and the hook; the machine $5(M5)$ makes the union between the terminal and the bimetal and the machine $6(M6)$ ends the processing by combining the braid and the hook. All the manufactured circuit breaker in this company go through the same processes, following the same sequence. Given the high competition of the high amperage products by the welding resources, there is a need for better utilizing those resources. Therefore, it is important to have an in-depth study of production scheduling of the items. For a first analysis of the described environment, it was made a process design by using graph that is shown in Figure 1.2. In this case, J_{111} is the job processing 1 in the machine 1 of the semi-line 1, J_{121} is the job1 processing on the machine 2 of the semi-line 1, and so on. In addition, the processing order is the same for all the machines and there is a dependency relationship

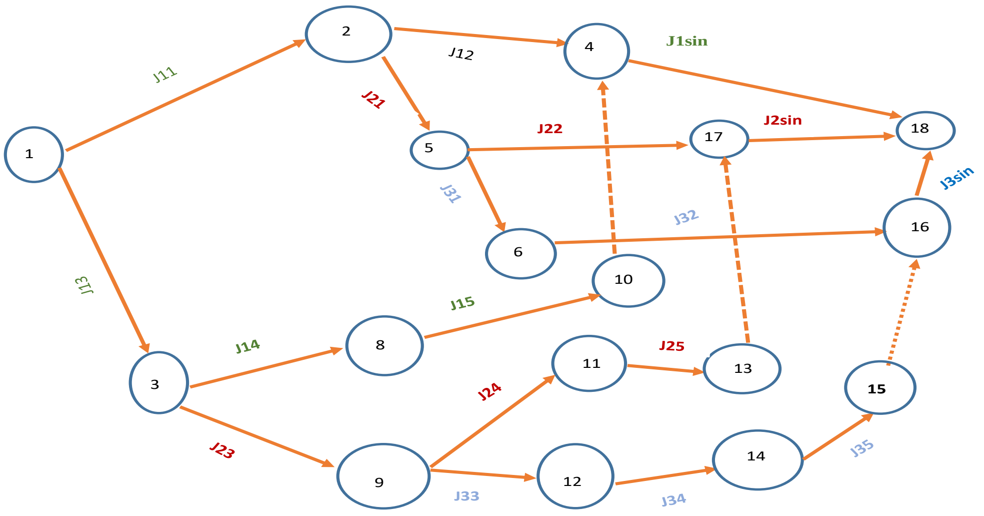


Fig. 1.2 The process graph

to carry out the activities. For example, J_{211} can only be done after the J_{121} has finished its processing. From the second job, it is noticed the same dependence pattern on all jobs. Thus, the following generalizations were considered: J_{iml} as job processing $(i = 1,...,I)$ in the machine $m(m = 1,...,M)$ of the semi-line $l(l = 1,2)$; and J_{isinl} as the processing in the synchronous machine $(sin = 1,...,S)$. Table 1.1 shows the dependency relationship.

Parts of the production line	Dependence Relationship		
	Job1	Job2	Job3
Semi-line1	$J_{111}-------\text{Not}$	$J_{211}----J_{111}$	$J_{311}----J_{211}$
	$J_{121}-------J_{111}$	$J_{221}----J_{211}$	$J_{321}----J_{311}$
Semi-line2	$J_{132}-------\text{Not}$	$J_{232}----J_{132}$	$J_{332}----J_{232}$
	$J_{142}-------J_{132}$	$J_{242}----J_{142}$	$J_{342}----J_{242}$
	$J_{152}-------J_{142}$	$J_{252}----J_{152}$	$J_{352}----J_{252}$
Synchronization	$J_{1sin}----J_{121},J_{152}$	$J_{2sin}--J_{221},J_{252}$	$J_{3sin}--J_{321},J_{352}$

Table 1.1 Dependence relationship between the jobs

1.2.2 Model description

The model is an assembly line consisted of two parallel semi-lines and a final synchronization operation. Each semi-line produces one of the halves of the final product that is assembled into a single product in the final synchronization operation. The order of the halves of the final products in each semi-line must be equal as well as it must be followed in the final synchronization operation.

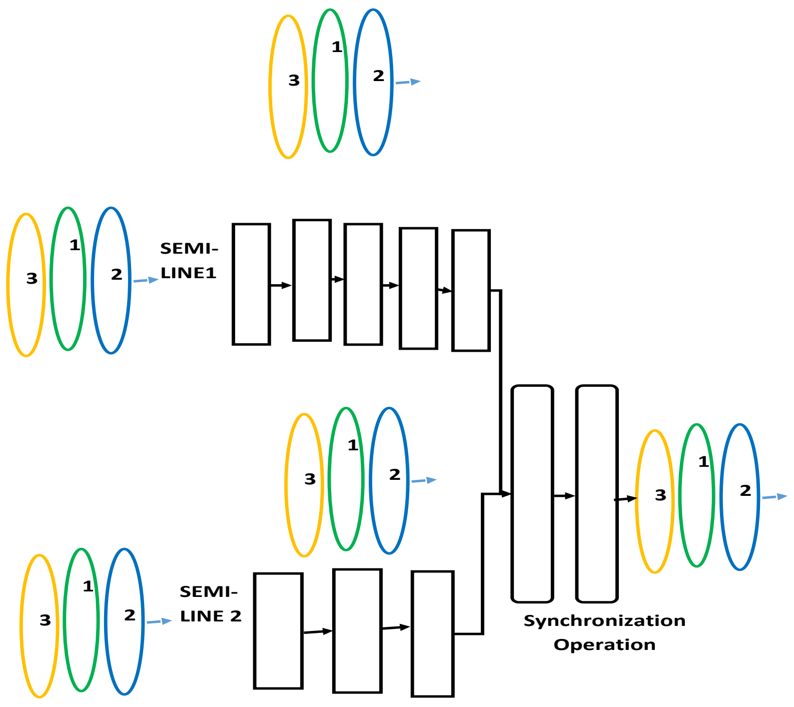


Fig. 1.3 System under study

The first semi-line has a number q_1 of machines and the second semi-line has a number q_2 of machines. At the end of the two semi-lines there is a machine in charge of the union of the semi-products from the two semi-lines. Each job requires operations in each machine of semi-lines with different processing time. An operation at the machine of a semi-line does not need to start at the same time of an operation at the machine of the other semi-line. However, the final synchronization operation for a product must only be started when the operations of their halves in both semi-lines are completed. Given this arrangement, it is possible to classify this environment as a special flow shop case since the problem has not been previously considered in the literature. The objective of this study is to achieve the optimal or nearly optimal solution in order to minimize the total processing time. Figure 1.3

shows a scheme of the studied environment.

1.3 Importance and purpose of the study

Determining the best production sequence within any sector of a company implies in a considerable cost reduction. This can be confirmed in the study by Gao and Chen [31], Gao et al. [32], Zhang and Van de Velde [109] and Shao et al. [98]. In an electrical-electronic material company, where the products compete for the same resources and the difference of the productivity of the machines is high, a well-defined sequence may reduce the waiting and delaying time, as well as optimize the time used in each resource. Taillard [101] asserts that the scheduling problem in a flow shop production environment has been the object of research in the last 50 years for its great importance in many types of manufacturing system. Although there are many studies on the classical flow shop problem, there are not meaningful studies on the flow shop problem with parallel semi-lines and synchronization operation at the end of the semi-lines yet. In this sense, the contribution of this research for this field is the development of a mathematical model and some computational models that use heuristics and metaheuristics for this flow shop environment. It also contributes with the analysis of those models by means of computational tests that search for new ways of reducing loss as well as the costs related to the environment in which it was believed that the optimized methods could not be applied.

1.3.1 Objective

The main objective of this study was to model the flow shop scheduling problem in parallel semi-lines with a final synchronization operation and to solve it with a mathematical model and algorithms using heuristics and metaheuristics. It also intends to:

1. Present a mathematical optimization model for the problem mentioned;
2. Study one variant of the flow shop problem motivated by a practical situation;
3. Develop new models and methods of solution for the flow shop problem with parallel semi-line and final synchronization operations;
4. Develop and test heuristic methods for the problem studied;
5. Develop metaheuristics in order to obtain a better quality of the solution for the problem.

1.4 Thesis structure

This thesis is organized in five chapters as described below.

- Chapter 1 presents an overview of the production scheduling problem, the description of the problem, the context in which the problem is solved as well as the importance of the study and its objectives.

- Chapter 2 presents some of the most relevant studies on the problem in the current literature. This chapter contains a brief description of the main concepts that guide this study. The problem of production sequencing, the environment of single machine, the environment of multiple stage machine, the variants of the flow shop and the methods of resolution for the flow shop problem are conceptualized.

- Chapter 3 presents methods for solving the flow shop sequencing problem with parallel semi-lines and final synchronization operation. In this chapter, the mathematical model to represent the problem as well as the methodologies of Johnson's rule, the *NEH* heuristics and the methods using local search *ILS*, Simulated Annealing and *GRASP* are described in detail.

- Chapter 4 reports how the test problems were generated as well as it presents and compares the achieved results through mathematical model and through the developed heuristic methodologies.

- Chapter 5 concludes the study and points out some suggestions for future works on the problem studied.

Chapter 2

Literature review

This chapter presents concepts and methodologies showed in the literature about flow shop scheduling problem. It contains a brief description of the main concepts that guide this study. Section 2.1 presents the problem of scheduling. In section 2.1.1, the problem classification is described. Section 2.1.2 discusses the simple machine environment. Section 2.1.3 presents the multiple stage machine environment. Section 2.1.4 introduces the flow shop variants and resolution methods for the flow shop problem. Finally, the main concepts of the resolution methods used in this study are presented in section 2.2.

2.1 Scheduling problem

The theory of scheduling in production systems deals with real problems that consist in obtaining a sequence of the orders of production (jobs) that optimize a given performance measure. In addition to minimizing the completion time of the production activities in factories, it includes the service deadlines or delivery dates, the minimizing of the flow time of the intermediate stocks and the maximizing of the use of available capacity, or even the combination of these objectives. According to Gupta and Chauhan [39], the programming consists of organizing the jobs for the processing of products in single machine or multiple machines. In other words, it is the process of organizing, choosing, temporizing the use of resources to carry out all the required activities in order to produce outputs at the desired time.

2.1.1 Problem classification

Taillard [101] states that, in general a scheduling problem of production is seen as a problem where n jobs $\{j_1, j_2, ...j_j, ..., j_n\}$ are processed on m machines $\{m_1, m_2, ..., m_i, ..., m_m\}$ that are available. The processing of a job j_j on machine m_i is called operation designated op_{ij}. For each operation op_{ij} there is an associated processing time p_{ij}. It consists in organizing jobs or required activities by the processing of a product in one or several machines. The feasible scale is the one that does not have overlapping jobs for a given time interval i.e., a job cannot be processed at the same time interval in two machines or at the time intervals corresponding to the same machine. A machine cannot process two jobs at the same time, and must satisfy the constraints regarding to each particular type of problem. Based on this definition, some assumptions are considered for this study:

Assumptions concerning the jobs

1. Each job is available for the processing from the beginning of the programming period;
2. There is no release date;
3. Each operation is independent of the other;
4. Each job (operation) has finite processing time in each machine. The processing time includes the time of transport and setup, if any, and it is independent of the previous and subsequent jobs;
5. Each job cannot be processed more than once in each machine;
6. Each job must have to wait next to each machine and thus generate an intermediate stock that will be allocated by the post processing.

Assumptions concerning the machines

- Each machine is inactive before starting the programming period;
- Each machine in a phase operates independently of the other machines and thefore is able to operate with its own maximum output rate;
- Each machine cannot process more than one job at a given time since it may eliminate. This eliminates some machines that are designed to handle many jobs simultaneously;
- Each machine will always be available to process the jobs during the programming period. No interruption is allowed either for breaks, maintenance or other causes.

Assumptions concerning the schedule policy

- Each job will be processed only when the machine is freed;

- Each job will be considered an inseparable entity although it may be composed of a number of individual units;
- Each job will be fully processed, once started, i.e. the cancellation of the job is not allowed (neither spliting nor preemption);
- Each job must be completed, once started in a machine. Only after carrying out the activity in this machine, the activity in another machine is started; therefore, the preemption is not allowed;
- Each job is processed in one machine at a time;
- Each machine has enough space to allocate the jobs that must wait the processing;
- The storage capacity will be considered unlimited;
- Each machine is not used for any other purpose during the programming period;
- Each machine processes the jobs in a similar sequence. Therefore, it will not be possible to skip the sequence.

Pinedo [78] reports that different machine configurations are possible for single-stage configuration (single machine, parallel machines) and for a multi-stage environment (machine in series and machine shop). In machine shops, three classifications are made: the flow shop, the job shop and the open shop. These environments are given below.

2.1.2 Single-stage machine environment

Single machine shop

The models of single machine consider the existence of n orders to be processed in a machine. These models are analyzed by researchers who study different types of conditions for different objective functions (Feldmann and Biskup [28], Agnetis et al. [4]). These objective functions can be related to the maximum completion time (Makespan), the waiting time, the setup time, the production time, the earliest due date, the release date of jobs, among others. These objectives may be solved by a set of rules that provide optimal solutions in single machine environment.

According to Pinedo [78], the single machine is any type of the machine environment. It is a special case among the other types of machine environment. Many productive systems are originated by models of single machines. Thus, this type of model is important to the decomposition method of more complicated sequencing problems. A way to solve these problems is by dividing them in a certain number of smaller problems of single machine. Studies by Gavett [34], Koulamas and Kyparisis [48], Magazine et al. [56] and Abdekhodaee et al. [2], [3] show some methods to solve this types of problem for different objective functions.

Parallel machine shop

The programming problem of parallel machines can be described as a set of *n* independent jobs (jobs, production orders) with known processing time that must be processed in one of the *m* parallel machines. In this context, the number of *m* machines is at least two and the number of jobs is greater than the number of machines. In this problem, the jobs are sequenced in order to present a distribution of jobs in the machines according to some criterion (makespan, delays, less load in the machines).

According to Nikabadi and Naderi [68], a set of parallel machines can be considered as a generalization of the single machine. The parallel machine programming is significant, since many algorithms can be reduced to solve individual machine problems and, from a practical point of view, it is important because in a real production environment most of the workshops have more than a machine. Moreover, the techniques for parallel machines are often used in multi-step decomposition process systems.

Cheng et al. [22] and Glass et al. [35] classify parallel machine for the type of used machines as it follows: identical parallel machines, uniform parallel machines and unrelated parallel machines.

- Identical Parallel Machines: production environment where there are *m* identical machines. The time of processing and preparation are identical for all the jobs. There is a unique set with time of execution of the jobs which remain independent of the machine whose job is assigned. All the machines have the same speed.

- Uniform Parallel Machines: In this case, the programming is done in a group of parallel machines in which each machine has different characteristics expressed in terms of processing speeds. The processing time of a job and the preparation time of the most modern machines are proportional to the time of the oldest machines.

- Unrelated Parallel Machines: In this case, there is no relationship between the processing and the preparation time of different machines. The speed of the processing depend directly on the job and the machines to be performed. (Nowicki and Smutnicki [70], Coelho et al. [23], Maschietto et al. [59])

In this environment, it must be determined which orders must be allocated for each machine as well as the processing sequence for each one. Parallel machine models are important for some reasons:

- If a work center has some constraints for the processing (the bottleneck), then the result of this work center determines the performance of the entire system.
- The bottleneck can be studied by analyzing a set of machines that can be in parallel or separately. In some cases, the machines in parallel cannot be exactly identical, some machines may be old and slow, and others may have a better maintenance and be able to perform accurately. In this case, while some jobs can be processed in any machine, others can be processed in only a specific subset of machines.
- When operations are performed by people (considering them as machines), then the processing time may depend on an operation in the job or on who is processing them. (Pinedo [78])

2.1.3 Multi-stage machine environment

Flow shop

In this type of environment, there are m machines in series. Each job has to be processed in each of m machines. Since all the jobs have the same processing sequence in the set of machines, that is, they follow the same route and must be processed first in the machine 1, then in the machine 2 and so on. Each job has a specific operating time in each machine. After its completion in one machine, the job joins the queue at the next machine. According to González-Neira et al. [37], initially all the jobs are assessed and each machine is restricted to the processing of only one job at any particular time. The generalization of the flow shop and parallel-machines environments is that in this case there are n stages. At least one stage has two or more machines in parallel that process the same kind of operation. Thus, the decision to be made is which parallel machines each job must be allocated to each stage. It can be seen that when there is only one machine in all stages the problem is a standard flow shop. The permutational flow shop is the most well known that may appear in the flow shop environment. In this environment, the queues in front of a machine operate according to the first in first out (FIFO) discipline. This implies that the order in which the jobs are processed by the first machine is maintained throughout the system. See (Nowicki and Smutnicki [69], Osman and Potts [72], Sayadi et al. [92], Wang et al.[105]). There is a single available machine for processing at each stage. In this environment it is possible to distinguish two situations. In Figure 2.1, a pure flow-shop in which jobs must be processed in each machine in exactly the same order is shown. However, according to Parveen and Ullah [75], a general flow shop is somewhat different, i.e., the jobs do not need to be processed in all the machines of the sequence. In this case, the processing time in the machine that does not have processing will be zero.

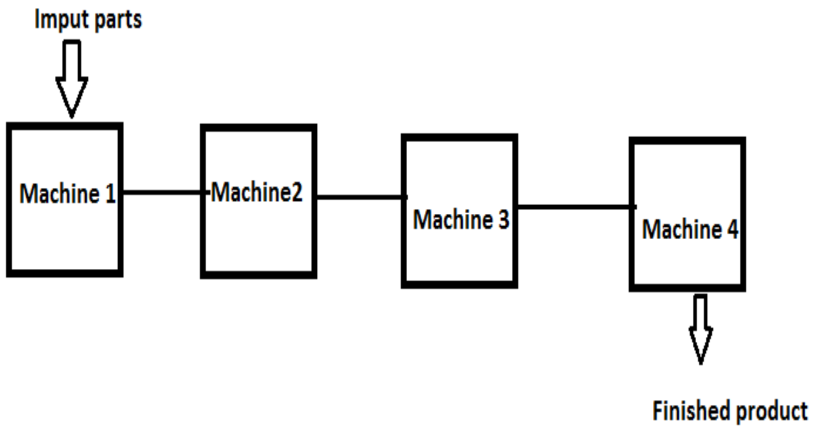


Fig. 2.1 Pure flow shop by Parveen and Ullah [75]

Job shop

In this general configuration, there are n jobs to be processed in m machines in variable routes. Each job has its own processing sequence in the set of machines, including the possibility of returning to a machine more than once. Due to the lack of uniformity of the production routes, in general the machines are grouped by similarity. See (Parveen and Ullah [75], Binato et al. [11], Lourenço [53])

Open shop

In the open shop environment, each product is manufactured by a number of operations, but the technological sequence is not given. Each job j has to be processed in each one of the m machines. However, some of this processing time may be equal to zero. In this process, it is possible to have the maximum in terms of flexibility, since the job route can be defined according to the optimization criteria of an objective function. See (Ciro Campos et al. [16], Schuurman and Woeginger [93] and Chen et al.[20]).

2.1.4 Sequencing of jobs in a flow shop production environment

This study aimed to explore the sequence of jobs in a flow shop production environment . In this environment, the jobs must undergo multiple operations in a number of different machines. All the jobs are processed in the same machines in the same technological sequence. Morton and Pentico [63] discuss this type of manufacturing environment according to some simple variations that can be called skip shop cells (skip operation), reentrant flow shop, compound flow shop, finite queue flow shop.

- Skip shop cells -

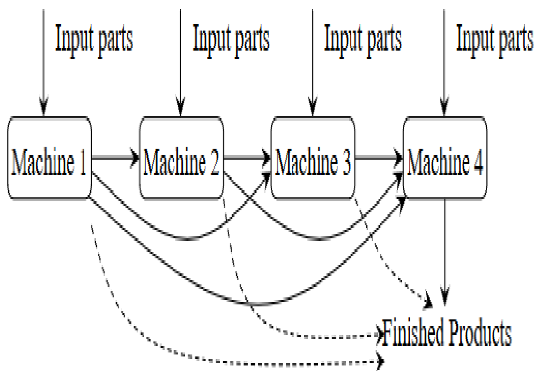


Fig. 2.2 Skip flow shop (Morton and Pentico [63])

  It considers the existence of a single machine in each stage, some jobs may skip some machines. There may be jobs that do not have all of the m operations. In this case, the operation that is not required to be processed in the job receives zero value as processing time. See Figure 2.2.

- Flow shop reentrant - In this environment, the operations for a particular job routing is such that a job may return one or more times to any of the machines. Dugardin et al. [27] state that the main difference with the other scheduling problems is that a product can be processed several times in the same machine. The performance criteria can be the makespan, the jobs of total flow time, the throughput rate, the total tardiness, among many others. Due to the repetitive use of the same machines by the same job, these processes can not be treated as a simple flow-shop problem. See Figure 2.3.

- Hybrid flow shop - It is a linear sequence of grouped m machines. Each group k of m machines is a set of parallel machines. It is considered that there is more than one

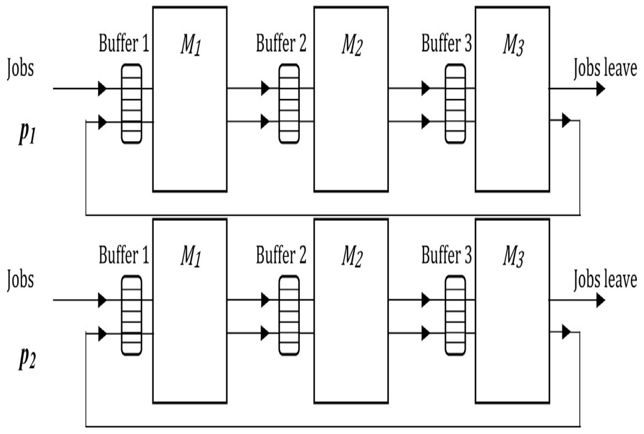


Fig. 2.3 Reentrant flow shop (Rifai et al. [85])

machine at each stage. According to Ribas et al. [84], the hybrid flow shop scheduling problem may be seen as a generalization of two particular types of scheduling problems: the parallel machine scheduling (PMS) problem and the flow shop scheduling (FSS) problem. The main decision-making in the operation of this type of configuration is to determine the order in which the jobs are processed in the different machines of each stage according to one or several given criteria. See Figure 2.4.

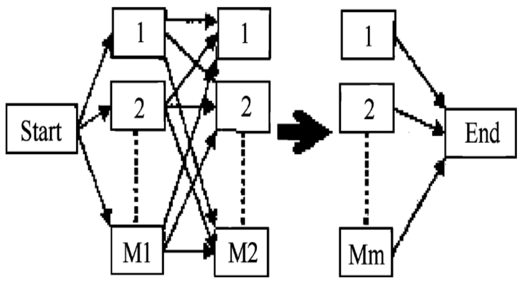


Fig. 2.4 Compound flow shop (Marichelvam et al. [58])

- Flow shop with finite queue - A flow shop with finite queue of the job in waiting in each machine. There is a limit of stock before each intermediate machine of the process. If the queue before a machine k is complete, then the machine $k-1$ will have to wait until the processing in the machine k is finished. This is called blocking. Li and Pan [49] say that the majority of the literature makes the assumption that there are sufficient intermediate buffers between consecutive stages. However, in real scheduling problems, a finite intermediate buffer (limited buffer) always exists, which blocks and delays the operation in the previous machine or the intermediate buffer. Some simple cases of overloading machine can cause blockage to many previous jobs. A special case is when no stock is allowed, except in the first machine. See Figure 2.5.

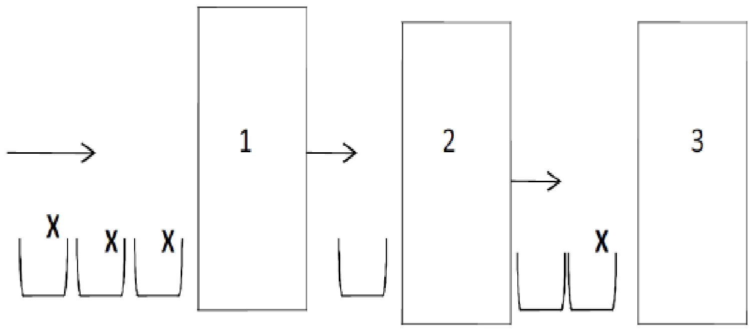


Fig. 2.5 Finite queue flow shop (Morton and Pentico [63])

2.1.5 Flow shop variants

Hybrid Flow Shop

The problem of programming in parallel machine can be described as it follows: n jobs are processed by a system of m-stages of production. Each stage comprises one or more identical parallel machines and has enough storage capacity to leave the job in progress ($WIP's$). Each part can be processed in a machine at each stage at the most. All jobs visit the m stages in the same order, but some parts can skip some stage if necessary (Li and Pan [49], Shahvari and Logendran [97]).

The processing time of jobs are known and do not depend on the machines. The setup time is independent of the sequence and is included in the processing time. No machine can process more than one job at once and the preemption is not allowed. The jobs are continuously transported from one stage to another (for example, by belts), and the time of transport from one machine to another is determined by the preview stages, so that all the machines of the stage are used. Each part has a known processing time that cannot started before.

Some methods for solving this problem are found in Ruiz and Vazques-Rodriguez [89] who presented an analysis of more than 200 scientific articles. They dealt with hybrid flow shop problem (HFS) and related variants. A comprehensive and extensive review of the sequencing flow shop with multiple machines in parallel per phase was made, classifying all the articles according to many parameters, including the constraints, the functions and the methodologies in order to support the researchers and professionals. This problem is found in many real-world applications. Given its importance and complexity, the problem has been HFS intensively studied.

Li et al. [51] studied a combinatorial optimization problem with multi objectives in a hybrid flow shop environment with parallel machines. The problem considers an environment with independent parallel machines with identical release dates, due dates and dependent configuration of the sequence. Preemption was considered prohibited. The objective was to minimize two different points: the makespan and the tardiness. The contribution of this study was to propose a new mathematical model for this particular problem and two approximate the methods: the Non Strong genetic algorithm ($NSGA-II$) and the Strength Pareto Evolutionary Algorithm ($SPEA-II$). Through the experimental results it was possible to verify the advantages of the methods by comparison with the results found in the mathematical model. It was also observed that in all tested cases the $NSGA-II$ algorithm was able to get the best solutions.

Rashidi et al. [81] studied the hybrid flow shop with unrelated parallel machines considering the possibility of preemption and the objective of minimizing the makespan and the maximum delay. A method that uses the principles of genetic algorithm was proposed for the problem resolution. This algorithm divided the population into multiple subpopulation. Each subpopulation received different weights to assist in the exploration of different regions. The algorithm was combined with a step of local search procedure called useful redirection. The proposed redirection procedure tries to help the algorithm to overcome the local optimum and to continue to look for another space of solution. In this step, the algorithm tries to achieve a better solution based on good quality chromosomes. Computational experiments indicated that the proposed method achieved good solutions.

Flow shop reentrant

The basic characteristic of a reentrant flow shop is that a job passes through processing of certain machines more than once. The reentrant flow shop (RFS) means that there are n jobs to be processed in m machines in the flow shop and each job must be processed in machines in the order of $\{m1, m2, ..., mm, m1, m2, ..., mm, m1, m2, ..., mm, ..., m1, m2, ..., mm\}$.

Chen et al. [19] claim that these processes cannot be treated as a simple flow shop problem. The repetitive use of the same machines by the same job means that there may be conflicts among the jobs in some machines in different levels of the processing. Previews processing to be done in a job by some particular machine may interfere in the other operations of this machine in a job. The application of techniques to solve this type of flow shop problem can be found in Jing et al. [44]. The authors studied a re-entrant flow shop scheduling problem in two machines in order to minimize the makespan. They also suggested a number of heuristics and then evaluated their performance through extensive computational experiments. The algorithms were modified from the existing algorithms (weighted profile fitting heuristic for cyclic scheduling of a line flow and Johnson's traditional rule for two machines) and some have recently been developed. Extensive computational experiments were made to evaluate the heuristics performance. The experimented results showed that the heuristics performance was meaningfully affected by the distribution of the job workloads in the machines and some of them were excellent.

Dugardin et al. [27] studied three methods based on a genetic algorithm and a basic exact method in order to solve the reentrant hybrid flow shop scheduling problem: the minimization of the cycle time and the maximization of the utilization rate of the bottleneck. The first method is based on the genetic algorithm ($NSGA2$), the second is based on the Strength Pareto Evolutionary Algorithm (SPEA2) and the final one $L - NSGA$ based on genetic algorithm and the Lorenz dominance. The methods were compared using several measures. The best solutions were provided by the $L - NSGA$.

Flow shop with limited intermediate storage

The resolution of the scheduling problem in a flow shop environment with limited intermediate stock has important application in the industrial system. For this case, the concern is not only looking for an optimal scale for the flow shop problem. In this activity, it was also observed the amount of jobs that waits in the queue, checking whether theses quantities are in accordance with the storage capacity between the stages. When the products are physically large (e.g., television, printer, etc.), the storage space between two successive machines may have limited capacity, which may cause the blockage. Blocking occurs when the storage space is completely filled and the machine upstream is not allowed to allocate this job in the storage space.

Example of application is the study by Almeder and Hartl [9] that showed the problem of productive process sequencing in a metallurgical company. The production process can be considered as stochastic problem of flexible flow shop with limited buffer. The main objective was to use metaheuristic to know what the appropriate measures are for the production schedule in order to avoid the productive process blocking. The authors chose to use the search in the variable neighborhood for stochastic problems. The first step of the proposed algorithm did a search in the neighborhood considering a multi objectives function. The second step consisted of a discrete event simulator to evaluate the production plan. The methodology used in this study allowed the company to reach better performance in the productive system in comparison to the development of the preview production planning.

2.2 Methods of resolution

2.2.1 A mathematical programming for the flow shop problem

The mathematical programming was initially used to solve the problem of production scheduling in the fifties and sixties decades. In this period, Wagner [104] proposed the mixed integer linear programming model.

$$Minimize \qquad c_{max} \tag{2.1}$$

subject to

$$\textstyle\sum_{j=1}^{n} z_{ij} = 1 \qquad i = 1,\dots,n \tag{2.2}$$

$$\textstyle\sum_{i=1}^{n} z_{ij} = 1 \qquad j = 1,\dots,n \tag{2.3}$$

$$\begin{gathered}\textstyle\sum_{i=1}^{n} p_{ri} z_{ij+1} + y_{j+1r} + x_{j+1r} = y_{jr} + \sum_{i=1}^{n} p_{r+1i} z_{ij} + x_{j+1r+1} \\ j = 1,\dots,n-1; j = 1,\dots,n; r = 1,\dots,m-1\end{gathered} \tag{2.4}$$

$$\textstyle\sum_{j=1}^{n}\sum_{i=1}^{n} p_{mi} z_{ij} + \sum_{j=1}^{n} x_{jm} = c_{max} \qquad j = 1,\dots,n \tag{2.5}$$

$$\textstyle\sum_{r=1}^{k-1}\sum_{i=1}^{n} p_{ri} z_{i1} = x_{1k} \qquad k = 2,\dots,m \tag{2.6}$$

$$y_{1k} = 0 \qquad k = 2,\dots,m-1 \tag{2.7}$$

The model deals with the problem of allocation of jobs in the production sequence positions (see Blazewicz et al. [12]) The variables and parameters are listed below: z_{ij}- Binary variable takes the value 1 if job i is in the j^{th} position of the permutation and 0 otherwise; x_{jk}- idle time (waiting time) in the machine k before the starting of the job in the position j in the jobs permutation; y_{jk}- idle time (waiting time) of the job in the j^{th} position of the permutation, after finishing the processing in the k machine, while waiting for the release of the machine $k+1$; p_{ri} - processing time of the job i in the machine r; and

C_{max} the makespan.

The set constraints 2.1 is related to the maximum time to minimize the objective function; the constraints 2.2 and 2.3 ensure that each job occupies only one position in the permutation sequence and that in each position of the permutation sequence be allocated by a single job; the constraints 2.4 provide equal processing time plus waiting time for every pair of adjacent machines in the m-machine flow shop; the constraints 2.5 determine the makespan; the constraints 2.6 consider the idle time of the second machine and of the following machines while waiting for the arrival of the first job; the constraints 2.7 ensure that the first job of the permutation always passes immediately to each successive machine.

Other studies have been developed for the flow shop scheduling considering exact techniques such as the mathematical programming. Naderi et al. [64] proposed a model of mixed integer linear programming to minimize the makespan and total delay in a flow shop environment. Frasch et al. [30] studied a MIP modelling approach for flow shop problems with a limited number of intermediate buffers. The authors considered three objective functions to minimize the makespan, the sum of completion time and the number of strands. Ronconi and Birgin [86] proposed a mixed integer linear programming formulation to minimize the advance and the total delay of the jobs delivery through an evaluation in terms of computational cost for the flow shop problem with unlimited buffer and zero buffer. Hnaien et al. [41] developed two models of mixed-integer programming (*MIP*) for the scheduling flow shop problem in two-machines in order to minimize the makespan.

2.2.2 Johnson's algorithm

Johnson's Algorithm [45] is suitable for problems with two machines and can be extended to three machines, if the strategy of dividing the number of machines in two pseudo-machines is used with processing time equal to the sum of the processing time in the current machine.

Given a set of jobs that must go through a sequence of two machines, Johnson's algorithm allocates the jobs from the first and from the last position of the schedule considering them in ascending order of production time as it is shown in Figure 2.6. This algorithm is suitable for problems with two machines and can be extended to three machines, if they are transformed in two pseudo-machines. In this last case, the processing time of one of the machines will be equal to the sum of the processing time of the grouped machines. The generalization of

Johnson's algorithm was proposed by Campbell et al. [15] by solving general problems of n-jobs in m-machines, where $m-1$ problems of 2-machines are solved and the sequence that has the smallest makespan is selected.

```
Procedure Johnson's Algorithm

1  Determine $\min\{P_{jk}\}$;
2      If $Job_j$ with $min\{P_{jk}\}$ came from maq1:
3          S1 $\leftarrow$ Job$_j$;
4      If $Job_j$ with $min\{P_{jk}\}$ came from maq2:
5          S2 $\leftarrow$ Job$_j$;
6      If $P_{j1} = P_{j2}$, may be put in either Set ($S1$ or $S2$);
7  Repeat this procedure until the lists are exhausted;
8  Sort $S1$ in increasing order of $P_{jk}$ ($SPT$);
9  Sort $S2$ in decreasing order of $P_{jk}(LPT)$ ;
10 Sequence $\leftarrow S1 + S2$.
end-Procedure Johnson's Algorithm
```

Fig. 2.6 Pseudo-code by Johnson's Algorithm

The variables P_{jk} is the processing time in the machine k. The basic principle of Johnson's algorithm is the analysis of the shorter processing time in the machines. If the shorter processing time is in the machine 1, the corresponding job will be allocated at the beginning of the sequence. If the shorter processing time is in the machine 2, the corresponding job will be allocated at the end of the sequence. The job i precedes job j in a optimal sequence that is $\min\{a_i, bj\} \leq$ min $\{aj, bi\}$.

Example of application of this method is shown by Zang and Van de Velde [109]. The authors considered the flow shop scheduling problem for n jobs in m machines in two stages with parallel machine in order to minimize the makespan. The problem was divided into two sub-problems: the first is related to the assignment of the jobs to the parallel machines. The second is related to the scheduling of the jobs in flow shop environment using Johnson's rule.

This rule also has been adopted by Allahverdi et al. [7] to minimize the makespan in flow shop context with two-machines. The authors proposed five different algorithms, considering the lower and upper time limits of the job processing. One of the used algorithms was Johnson's algorithm, which is based on the average of the lower and upper time limits of the job processing. Computational experiments indicated that the modified Johnson's algorithm can be applied with success in programming environments with random and limited processing time. Yang [107] considered the problem of minimizing the makespan in a hybrid flow shop with two stages with dedicated machines in the first stage and one machine in the second stage. Four heuristics based on Johnson's algorithm and greedy-type scheduling were proposed. In all the cases, the two-stage hybrid flow shop problem with dedicated machines is assumed in order to minimize the makespan since there is only one machine in the second stage. The results indicated that the heuristic based on Johnson's algorithm provided the best performance. A heuristic that has been widely used to solve the flow shop problem is the NEH heuristic proposed by Nawaz et al. [66]).

2.2.3 Heuristic methods

Constructive heuristics

Many exact methods are developed to solve the flow shop problem, but there is great difficulty to find optimal solutions within feasible computational time using these methods, when the machine number and jobs are high. Therefore, other methods such as heuristic and metaheuristic are used to bring new solutions to this problem. Many researchers use the constructive heuristics because they generate good solutions with low computational effort.

The constructive methods may generate partial solutions that can be used as a starting point to achieve better final solutions. They can be used in isolation, but they are often combined with other more elaborate methods (exact methods, metaheuristics, methods of local search, etc). This solution can be created by successive generations of partial sequences to achieve a complete solution by inserting the operations. One example is the heuristic proposed by Nawaz et al. [66].

Rad et al. [80] remark that the NEH algorithm is commonly used to solve problems of sequencing n jobs in m machines. The sequencing outputs are represented by Gantt's graphs using the processing time of the machines for each job, ensuring that the delay time is taken into consideration. These processes are repeated for different sequences until reaching a minimum makespan, this is the shortest processing time considering all jobs. The storage capacity between successive machines is generally considered unlimited. This really fits to the physically small product processes (integrated circuit), which can be stored in large amount next to the machines. Therefore, this problem is treated considering the intermediate stock limited or unlimited.

Heuristics of Nawaz, Enscore and Ham (NEH)

NEH procedure is based on the assumption that a job with a large total processing time in all machines must receive higher priority than those with shorter total processing time. It builds the sequence by adding a new job to each step finding the best partial solution. The NEH algorithm can be presented in some simple steps as shown in Figure 2.7.

```
Procedure by NEH's Algorithm
1  Sort the tasks in descending order of STP, getting the list: $LC := \{\Pi_1,...,\Pi_n\}$;
2  Select the first and second job of the sequence from $LC$;
3  Calculate the processing time of the two jobs by allocating them in all possible positions;
4  $\Pi := \Pi_j$ a partial sequence formed by the selected job;
5  Remove the jobs $\Pi(1)$ and $\Pi(2)$ from $LC$;
6      For $i = 3$ to $n$ do
7          Select job $\Pi(j)$ randomly from $LCR$;
8          Enter the job $\Pi(j)$ in all possible positions of $\Pi$, generating $i$
9          partial sequences with $i$ jobs;
10         $\Pi'$ : = Select the best generated sequence;
11         Remove the $\Pi_(j)$ from $LC$;
12     end-for
end-procedureNEH's Algorithm
```

Fig. 2.7 Pseudo-code by *NEH*

This heuristic has been widely used to solve the flow shop problem. Kalczynski and Kamburowski [46] consider this heuristic as one of the best to solve the flow shop permutation problem. According to Samuel and Venkumar [90], this heuristic assumes that a job with a high total processing time in all the machines should receive higher priority over those jobs which have low total processing time. This heuristic (*NEH*) builds the final sequence by adding to each step a new job, finding the best partial solution. An example is shown by Pan et al. [74] who solved the scheduling problem in a flow shop environment with buffer-zero, with the reducing criterion of the processing time of all jobs (makespan). The author proposed heuristics to explore specific characteristics of the problem and to find out good solutions with shorter computational efforts.

Simple constructive heuristics were also analyzed based on the approximation of the study by Mccormick and Rao [60] (scheduling on the production line with the blocking to minimize the cycle time). Computational simulations and comparisons have been built based on the number of instance for flow shop with the blocking proposed by Taillard benchmarks. The results showed that the performance of the constructive heuristic brought significant results.

Another example is the study by Allaoui and Artiba [8] who investigated the hybrid flow shop scheduling problem with two stages composed of a single machine in the first stage and m machines in the second stage in order to minimize the makespan. Recently, Fernandez-Viagas and Framinan [29] proposed tie-breaking mechanisms with the NEH heuristic to solve the Fm/prmu/$\sum$ T$_j$ problem.

Improvement heuristics

The local search method in optimization problems is the refinement of a viable starting solution. This heuristic explores the search space in order to find the most promising regions using neighborhood notions. Local search looks for better solution by covering the search that moves iteratively from a nearby solution to another. According to Aarts and Lenstra [1], there is a natural way to represent the solutions for many local search applications for scheduling problem. In a scheduling problem, the representation of the solution is a permutation of the integer $1,...,n$ for an assignment problem. It is in a list of m machines to which the n jobs are assigned. In many cases, there is a definition of the neighborhood structure that attends to improve the constructed solution. Among the best known refinement methods are:

- Descent/Uphill Method

  Local search heuristics start from random initial solution and explore all possible neighbors of this solution in search of better solutions. To construct the neighbors, it is necessary to establish rules that define a certain type of movement (modification that transforms a solution in another in neighborhood). The exchange of a solution to another will be accepted if the neighboring solution represents an improvement in the current value of the evaluation function. The heuristic stopping criteria will be made when the local optimum is reached.

- Random Descent Method/Uphill

  This method is a kind of descent method. In this case, any neighbor is chosen and is only accepted if it improves the current solution. If the chosen neighbor is not better than the current one, other neighbor is chosen and the current solution is maintained until a better solution is generated. The procedure ends when the fixed number of iterations with no improvement over the best solution is reached. Therefore, the final solution will not necessarily be a local optimum.

- First Improvement Method

  This is a kind of descent method that consists in interrupting the descent method in the neighborhood when it reaches the first neighbor with better solution than the current one. In this case, the exhaustive search for the best neighbor is avoided, which only occurs in the worst case.

Mainieri and Ronconi [57] analyzed the problem of minimizing the total delaying in a flexible flow shop environment. In this study, they proposed new rules of order, based on the date of the delivery and future states of the system, which are evaluated with a set of test of 4320 problems. Other example is the study by Nagano and Moccellin [65] who proposed a new heuristic method of two stages. In the first stage, an initial ordering of the tasks was performed. In the second, the solution sequence with the inclusion of tasks at the partial sequences and their improvement with a search method at the neighborhood was built. This method aimed minimizing the processing inventory without interrupting the task performance.

2.2.4 Metaheuristic

The metaheuristics have the ability to escape from local optimum through prior knowledge of the information on the search space to be explored, avoiding the premature stop of the search algorithm. Today, there are several different metaheuristics that serve as reference for the development of search strategies. Those seen as extensions of local search algorithms that seeks to escape from local optimum in order to proceed with the searching in another region (tabu search, variable neighborhood search (VNS), greedy randomized adaptive search (GRASP), simulated annealing, etc), and those based on the philosophy of learning, the combining of the previously given solutions to identify the most promising areas of the search space (genetic algorithms, neural networks, ant colony etc.) can be cited. Some articles that used metaheuristic for the problem of flow shop scheduling are presented below.

Widmer and Hertz [106] presented an heuristic method to solve flow shop scheduling. This method named SPIRIT (Scheduling Problem Involving a Resolution by Integration of the Tabu search technique) was composed of two stages. In the first stage, the initial sequence uses the principles of traveling salesman problem, and in the second stage is the improvement of the initial solution using the heuristic by the tabu search method. The presented heuristics were tested in a total of 500 problems, where problems with $4,5,10,15,20$ jobs and $4,5,10,15,20$ machines were considered. The processing time are randomly generated and uniformly distributed over the interval $(0;10)$. The used distance by the insertion method provides a good initial feasible solution. A list of all tested instances was not exhaustive. The best solution for the initial tabu search method used in this study did not show any meaningful results with respect to the value of makespan.

Alaykỳran et al. [6] used the method of improvement for the ant colony optimization (ACO) to solve the problems of hybrid flow shop (HFS). This algorithm was used in many other problems. The authors have improved and adapted these methods to the HFS problem. The operating parameters have an important role in the quality of the solution. A study of optimization parameters was carried out and evaluated in the HFS problems. The generated algorithm was tested in 63 different reference problems. The results showed that the application of the new method ACO was effective. Today, the inspiration in the nature for the solution of the problem such as the one that has brought good result and it will be adaptable to other problems in the future. Alaykỳran et al. [6] affirm that the best results can be achieved by using hybrid or applications in parallel as well as by setting of the parameters solving problems in these methods.

Buzzo and Moccellin [13] presented a hybrid heuristic with Simulated Annealing and Genetic Algorithms to minimize the makespan. The objective was to evaluate the hybridization efficacy. The hybrid method was compared to the original Genetic Algorithm and Simulated Annealing methods. The method showed that hybrid models can be useful to solve this problem.

Reeves [82] explored the potential of GA to find out the minimum makespan in a permutation flow shop problem of n and m-jobs machines. The algorithm performance was compared to the results acquired by the simple search technique in the neighborhood, the Genetic Algorithm (*GA*) and the Simulated Annealing algorithm (*SA*). By testing a set of instances for this problem, it was observed that the use of sophisticated procedures for certain types of problem may not be worthwhile, since a simple search through the neighborhood can get good quality solutions. The overall implication of these studies was that the *SA* and *GA* algorithms produced results comparable to flow shop scheduling problem for most sizes and types of problem. The *GA* produced better results for large problems. According to the authors, a characteristic of the *GA* is its robustness with respect to the parameter values so that the similar performance was already expected.

Gao et al. [31] considered the Distributed Permutation Flow Shop Problem ($DPFSP$). They proposed a resolution method that uses a hybrid genetic algorithm with local search (*GALS*) to minimize the makespan. In the resolution, the operators of *GA* crossover and mutation were used at one point in order to make it suitable for the $DPFSP$ representing solutions where the set of partial sequences of jobs is the employed method.

In addition, a local search to exploit neighboring solutions has been developed where three local search operations have been proposed for enhancing the VND method. The local search method uses three proposed rules that move jobs within a plant or between two plants. Intense experiences were made based on instances referenced by Taillard. The results indicated that the proposed hybrid genetic algorithm can get better solutions than all the existing algorithms for $DPFSP$ once the relative percentage deviation and the differences in the results were statistically significant.

Gao et al. [32] continued the previous study, considering the problem of flow shop problem Scheduling Distributed Permutation ($DPFSP$), but solving it by tabu search algorithm. The contribution of this study was to treat the problem through a method of exchanging sub-sequences of jobs between two factories that define the neighborhood structure and movements. Furthermore, the method also improves the exchange in the tabu list based on local search method presented in a more effective method of local search hybrid genetic algorithm that is incorporated into the tabu search algorithm. Taillard instances (extended to the problems of distributed permutation flow shop) were used to test the algorithm. As from experiments and comparisons with other algorithms, this study showed that for 472 known instances, the proposed tabu algorithm outperforms all existing performed algorithms, including heuristic algorithms (i.e, $NEH1$, $NEH2$, and VND). In addition, it was shown that the efficiency of the tabu algorithm is better than that of the aforementioned hybrid genetic algorithm.

Simulated annealing procedure

The simulated annealing is based on Monte-Carlo method so that it can be regarded as a special form of iterative improvement. According to Ozdaugouglu [73], the link between this algorithm and mathematical minimization was first observed by Pincus [76]. However, who really proposed the optimization technique for combinatorial problems were Kirkpatrick et al. [47].

Simulated annealing was based on the simulation of the natural process of grain growth in the metal. In this process, metal is melted at a very high temperature, followed by a slow gradual cooling. One of the main points of attention in this process is the phenomenon of the grain growth that constitutes the crystalline metal net. In this phenomenon, the average diameter of the crystalline metal net increases when the metal cooling is carried out slowly. In this regard, Callister et al. [14] say that this phenomenon is a process whose length depends on both time and temperature. To extent that the grains grow in size, the total area of grain boundary decreases, providing a steady reduction in the total energy. This is the driving force which grain growth occurs by the migration of grain boundaries. There is a dependence of the grain size in relation to the time and the temperature. The grain growth proceeds faster as the temperature increases. In this sense, the longest cooling process provides more stable metals if strong structures of lower energy and fast cooling brings about a metastable structure with higher internal energy, what makes the initial temperature and the cooling time important data.

By observing this process, Kirkpatrick et al. [47] developed the algorithm simulated annealing drawing an analogy with metal cooling behavior and the Metropolis algorithm [61]. This algorithm aimed to find the global minimum of a cost function that can contain many local minimum. The combinatorial optimization algorithm performs the process by steps simulating the temperature levels of the metal cooling. According to the Nearchou [67] and Aarts and Lenstra [1], a combinatorial optimization problem generally consists of a set of solutions S and a cost function $C(S)$ which determines a real number for each cost solution $C(S)$. Furthermore, to perform a local search, it is necessary to define the neighborhood structure $N(S)$ in S. $N(S)$ determines S' possible changes for each set of solutions that may be proposed by S. An arbitrary S' solution is obtained from S and continually improved from a small local change until such a change produces a better solution. At each iteration, the algorithm explores the neighborhood generating a finite number of candidate solutions S' randomly of which the value of the cost function $C(S')$ is calculated. The evaluation of the candidate solution is made by the difference between the value of the cost function $C(S)$ and the candidate cost function $C(S')$. S is only replaced by S' if the cost function $C(S')$ does not increase, or $C(S') \leq C(S)$. But the local search algorithms can achieve minimum costs of which they do not offer any improvement in cost and are not necessarily the global minimum. To prevent that the algorithm gets stuck in a local minimum, the simulated annealing accepts "upward movements" with a certain probability, i.e a candidate solution that increases the cost function.

The way to get this probability is made by the drawing of a real value between 0 and 1. If the obtained value is greater than $e^{-\triangle/T}$, the worsening solution is considered, so that the acceptance of this probability decreases with time. For this case, the $\triangle$ is regarded as the difference between the cost of the candidate solution and the cost of the former solution, and T current temperature. The temperature is decreased at each iteration by a cooling factor α until achieves a pre-established minimum value. The classic algorithm is presented in Figure 2.8.

In the current literature, some studies have been using this technique. Nearchou [67] proposed a hybrid algorithm that uses simulated annealing for the classic problem of flow shop scheduling. This study dealt with the problem using some characteristics of the genetic algorithm. The author compared the computational results with two different mechanisms to produce different solutions. The first population was generated using a simple random change. The second population was generated through a recombination system based on the previously acquired knowledge by the individuals that were generated randomly in the population. The experiments were done in benchmark of test problems comparing the proposed method, the genetic algorithm and two other algorithms using standard simulated annealing. The results generated by the hybrid algorithm had low makespans with average results below 1% of the best solution.

Mirsanei et al. [62] also considered the hybrid flow shop problem with parallel identical machines to minimize the makespan. An algorithm that uses the simulated annealing was developed to generate a feasible production schedule with an acceptable computational time. The algorithm combines two movement operations to generate new solutions. The obtained

Procedure *Simulated Annealing*

```
1  Select an initial temperature $T \leftarrow T_0$;
2  Select an initial Solution $S \leftarrow S_0$;
3  $it \leftarrow 0$;
4  $best_{it} \leftarrow 0$;
5  $S_{best} \leftarrow S_0$;
6  While $(\mathtt{it} \leq \mathtt{best_{it}} + \mathtt{max_{it}})$ do
7      $it_{sol} \leftarrow it$;
8      While $(\mathtt{it} \leq \mathtt{it_{sol}} + \mathtt{it_{temp}})$ do
9          $it++$;
10         $S' \leftarrow N(S)$
11         $\triangle \leftarrow cost(S') - cost(S)$;
12         If $\triangle \leq 0$ then :
13             $S \leftarrow S'$
14             $it_{sol} \leftarrow it$
15             If $\mathtt{cost(S')} < \mathtt{cost(S_{best})}$ then :
16                 $S_{best} \leftarrow S'$
17                 $best_{it} \leftarrow it$
18             endif
19         endif
20         Else
21              Generate random number $r \in [0,1]$
22             If ( r < $e^{-\triangle/T}$)Then:
23                 $S \leftarrow S'$;
24             endif
25         end
26     end
27     $T \leftarrow \alpha T$
28 end
29 Return S;
```

end-procedure Simulated Annealing

Fig. 2.8 Pseudo-code of the Simulated Annealing

results were compared with those generated by the genetic algorithm and others previously proposed. The algorithm has brought better results in relation to the algorithms previously reported in the literature.

Senthilkumar [94] developed an algorithm with simulated annealing to minimize the makespan in an environment with single machines and with uniform parallel machines. The scheduling problem in single machine and in uniform parallel machine consists of *n* jobs that were scheduled in *m* parallel machines with different speeds, so that each job has a single operation. To solve the problem, three variations of the simulated annealing were carried out and a statistical analyses (*ANOVA*) was performed to compare the solutions. The three algorithms brought good results for the tested problems.

Sivasankaran [99] compared the simulated annealing algorithm and the *GRASP* (Greedy Randomized Adaptive) algorithm to sequence jobs in *m* unrelated parallel machines and in uniforms parallel machines in order to minimize the makespan. The authors presented two different metaheuristics to the problem and compared them analyzing their solutions. In the first step, the simulated annealing algorithm was developed and then the *GRASP* algorithm. The results showed that the simulated annealing algorithm had better results than *GRASP*. Seyed-Alagheband et al. [95] investigated the problem of jobs scheduling in a flow shop pemutacional environment. To solve the problem, an algorithm with modified classical simulated annealing was developed. The algorithm was effective in terms of solution quality compared to other approaches available in the literature.

Hurkala and Hurkala [42] addressed the problem of programming of *n* jobs in *m* machines with minimal completion time as a performance criterion. The authors developed an algorithm with the simulated annealing to solve the problem. The main difficulty at the algorithm performance was the temperature parameter setting for the shop flow problem since the quality of the solutions is dependent on the choice of the cooling system, the initial temperature, the number of iterations, and the temperature reduction rate. The authors have proposed choices for all this parameter and for the Boltzmann factor for Metropolis scheme. Three disturbance techniques were tested and the impact of these techniques on the solutions quality was assessed. The computational experiments had good results and the quality of the solutions of the algorithm simulated annealing was better compared with two different heuristics.

Lin et al. [52] studied the problem of flow shop permutation schedule in order to minimize the total flow time. A multi-start heuristic with simulated annealing (*MSA*) was developed due to the computational complexity of this problem. This heuristic uses more than an initiation strategy at the simulated annealing (*SA*) to achieve near optimal solutions. A set of computational tests was conducted in a known problem in the literature to examine the performance of the *MSA* algorithm. The test results with the new method were better compared to the traditional methods using other metaheuristics.

Greedy Randomized Adaptive Search procedure (*GRASP*)

```
Procedure GRASP ($Maxiter$), ($Seed$)
1  Read input()
2  $iter \leftarrow 0$;
3      while $iter < maxiter$ do
4          $Solution \leftarrow Construction(seed)$;
5          $Solution \leftarrow LocalSearch(Solution)$;
6          $Update - Solution(Solution, Best - Solution)$;
7      end-while
8      return( Best-Solution)
end-procedure GRASP
```

Fig. 2.9 GRASP pseudo-code (Resende and Ribeiro [83])

The metaheuristic Randomized Adaptive Search Procedure (GRASP) is known as an iterative method proposed by Resende and Ribeiro [83]. The method combines the main characteristics of the greedy algorithms and random methods, generating solutions through multiple restarts (pseudo-code Figure 2.9). The algorithm consists of two phases: the

```
Procedure Construction (seed)
1  Read input()
2  Solution $\leftarrow$ 0;
3      Evaluate the incremental cost of candidate elements;
4          while solution is not complete do
5                  Build the Restrict Candidate List (RCL);
6                  Select element s of RCL at randon;
7                  $Solution \leftarrow SolutionUs$;
8                  Reeavaliation the incremental costs;
9      end-while
10         return( Solution)
end-procedure Construction
```

Fig. 2.10 GRASP pseudo-code (Resende and Ribeiro [83])

construction phase of the initial solution and the local search phase or improvement solution, returning an optimal local of the initial solution neighborhood. The construction (pseudo-code Figure 2.10) of initial solution consists in generating a list of feasible candidate solutions

(*LC*). The elements of this list are ordered according to a random greedy adaptive function. The best solutions make up the restricted candidates list (*LRC*), so that the construction dimensioning and randomness are controlled by the real α parameter $[0,1]$. The value of $\alpha = 1$ represents a solution completely random and $\alpha = 0$ represents a solution entirely greedy. When the construction of the initial solution is finished, the *LRC* elements are subjected to a search method in the neighborhood to the minimum site. The best solution is stored as a partial result. The best partial solutions, considering all iterations, is adopted as the final result (Resende and Ribeiro [83]).

```
Procedure Local Search (Solution)
1   $iter \leftarrow 0$;
2       while $iter < maxiter$ do
3           Find S' $\in N\ (Solution) with f(s') < f(Solution)$;
4           Set Solution$\leftarrow s'$;
5           iter ++;
6       end-while
7       return(Solution)
end-Local Search
```

Fig. 2.11 Pseudo-code of the local search phase (Resende and Ribeiro [83])

Examples can be seen in Prabhaharan et al. [79]. They investigated the flow shop permutation problem to minimize the makespan. They presented the results of the jobs sequencing with *GRASP* metaheuristic and then with the *NEH* algorithm. The computational results showed that *GRASP* algorithm overcame the traditional *NEH* algorithm.

Sivasankaran et al. [99] compared the simulated annealing algorithm and the *GRASP* algorithm to schedule the operation of jobs in *m* unrelated parallel machines and in uniforms parallel machines in order to minimize the makespan. The simulated annealing algorithm performed better than *GRASP*. Shahul et al. [96] dealt with the flow shop problem with *m* machines in order to minimize the weighted sum of the makespan and the maximum delay. One of the used techniques was the *GRASP* metaheuristic (*GRASP* Bi-criteria algorithm). The proposed algorithm was effective for small problems.

Davoudpour and Ashrafi [25] addressed the problem of hybrid flow shop with sequence dependent of the setup time. The authors presented a modified *GRASP* algorithm with an additional reconstruction phase in order to determine the due dates and the setup time. The *GRASP* algorithm was tested and compared to the simulated annealing algorithm. The computational results showed that *GRASP* achieved better results than the simulated annealing.

Damodaran et al. [24] investigated the scheduling problem in a similar set of plots arranged in parallel machines using a *GRASP* approach in order to minimize the makespan considering different numbers of jobs and processing time. The performance of the proposed GRASP algorithm was compared to some heuristics published in the literature. On average, the *GRASP* algorithm overcame the compared heuristics.

2.3 Conclusion

In this chapter, the concepts of problem of production scheduling, the environment of single machine, the environment of multiple stage machine, the variants of the flow shop and the methods of resolution for the flow shop problem as well as definitions of exact methods and heuristic methods were presented. This theoretical basis is fundamental for a better understanding of subsequent chapters.

Chapter 3

Optimization approaches

This chapter presents approach methods to the flow shop scheduling problem with parallel semi-line with a final synchronization operation. To solve this problem, a mixed integer linear programming model is presented in section 3.2, methods considering Johnson's algorithm in section 3.3, heuristic methods based on *NEH* algorithm in section 3.4 and methods based on Local Search, Simulated Annealing and *GRASP* methaheuristics and in sections 3.5, 3.5.3,3.5.2 and 3.5.4.

3.1 Characteristics of the studied problem

The flow shop problem with parallel semi-lines and a final synchronization operation can be represented by a permutation of n jobs $J = 1,2,3,...,n$, where J_k is the k_th job in the processing sequence. The jobs are processed in parallel in the same technological sequence in all the machines of the two semi-lines and the synchronization process. Each semi-line processes one of the halves of the final product that is assembled in a single product in the synchronization machine. The semi-line1 has q1 machines $Q1 = 1,2,3,\ldots q1$ and the semi-line 2 has $q2$ machines $Q2 = 1,2,3,...,q2$ and ending with a synchronization machine qs.

Assumptions concerning to the problem

1. Each job is available for processing from the beginning of the scheduling period;
2. There is no due date;
3. Each operation is independent of the other;
4. Each job (operation) has finite processing time in each machine;

5. Each job cannot be processed more than once in each machine;
6. Each machine is ready to be used before starting the scheduling period;
7. Each machine in a stage operates independently of the other machines and therefore is able to operate with its own maximum output rate;
8. Each machine will always be available to process the jobs during the period;
9. Each job will be processed only when the machine is available;
10. Each job must be concluded, once started in a machine. The preemption is not allowed;
11. Each job is processed in one machine at a time;
12. Each machine has enough space to allocate the jobs that must wait the processing;
13. The storage capacity will be considered unlimited;
14. Each machine processes the jobs in a similar sequence;
15. Each semi-line operates independently from the other;
16. The semi-lines and the final synchronization operation process the jobs in a similar sequence;
17. The final synchronization operation for a product can only be started when the operations of their halves in both semi-lines are concluded.

3.2 Mathematical modeling

The mathematical formulation addresses the problem considering two parallel semi-lines followed by a sequential line. The operations in the parallel machines are grouped, and the processing time of the corresponding operation to the operations set of the slower group is adopted. A dummy machine (considered in the model as synchronization machine) is included at the end of each semi-line. These two dummy machines represent the independent processing of the last operation. The existence of two artificially independent semi-lines will enable the determination of the higher value c_n^1 of the processing time between them. The model is based on the formulations proposed by Stafford [100] and by Wagner [104]. According to the study conducted by Tseng, Stafford Jr. and Gupta [102], such kind of formulation is the best for the permutation flow shop. Let *ml* be the number of machines in semi-line $l = 1,2$ (before the synchronizing operation). In this model, a unique permutation has to be chosen for two flow shop problems with $ml+1$ machines, each subject to the additional constraint that the completion time of each job in the machines $m1+1$ and $m2+1$ is the same. The model can be generalized for an arbitrary number L of semi-lines. Thus, assuming, for modeling purposes, $l = 1,2$ flow shop problems with $ml+1$ machines each, the variables and parameters are the following: z_{ij} is a binary variable that takes the value 1 if job i is assigned to the j^{th} position of the permutation (common two both l = 1; 2 flow shop problems) and 0 otherwise; x^l_{jk} is the idle time in the machine k of problem l before the starting of the job in the j^{th} position of the permutation; y^l_{jk} is the job idle time in the j^{th} position of the permutation, after finishing the processing in the machine k, while waiting for the releasing of the machine $k+1$ of problem l to become available; p^l_{ki} is the processing time of the job i in the machine k of the problem l (pm1+1i = pm2+1i); c^l_j is the completion time in problem 1 of job j in the j^{th} position of the permutation. The makespan is given by the completion time of the job in the last position of the permutation.

$$\textbf{Minimize} \quad C^1_n \tag{3.1}$$

subject to

$$\textstyle\sum_{j=1}^{n} z_{i,j} = 1 \qquad i = 1,\ldots,n \tag{3.2}$$

$$\textstyle\sum_{i=1}^{n} z_{i,j} = 1 \qquad j = 1,\ldots,n \tag{3.3}$$

$$\begin{gathered}\textstyle\sum_{i=1}^{n} p^l_{r,i} z_{i,j+1} + y^l_{j+1,r} + x^l_{j+1,r} = y^l_{j,r} + \sum_{i=1}^{n} p^l_{r+1,i} z_{i,j} + x^l_{j+1,r+1} \\ l = 1,2; j = 1,\ldots,n-1; r = 1,\ldots,m_l\end{gathered} \tag{3.4}$$

$$\textstyle\sum_{r=1}^{k-1} \sum_{i=1}^{n} p^l_{r,i} z_{i,1} = x^l_{1,k} \qquad l = 1,2; k = 2,\ldots,m_l \tag{3.5}$$

$$\textstyle\sum_{r=1}^{mv} \sum_{i=1}^{n} p^v_{r,i} z_{i,1} \leq x^l_{1,ml+1} \qquad l,v = 1,2 \tag{3.6}$$

$$y^l_{1,k} = 0 \qquad l = 1,2; k = 1,\ldots,m_l - 1 \tag{3.7}$$

$$x^l_{1,m_l+1} - (x^l_{1,m_l} + \textstyle\sum_{i=1}^{n} p_{m_l,i} z_{i,1}) = y^l_{1,ml} \qquad l = 1,2 \tag{3.8}$$

$$\sum_{u=1}^{j}\sum_{i=1}^{n} p^{l}_{m_l+1,i} z_{i,u} + \sum_{u=1}^{j} x^{l}_{u,m_l+1} = C^{l}_{j} \quad l=1,2; j=1,\ldots,n \tag{3.9}$$

$$C^{1}_{j} = C^{2}_{j} \qquad l=1,2 \tag{3.10}$$

$$z_{ij} \in \{0,1\} \qquad j=1,\ldots,n; i=1,\ldots,n \tag{3.11}$$

$$y^{l}_{j,k}, x^{l}_{j,k} \geq 0 \qquad j=1,\ldots,n; l=1,2; k=1,\ldots,m_l+1 \tag{3.12}$$

$$C^{l}_{j} \geq 0 \qquad l=1,2; j=1,\ldots,n. \tag{3.13}$$

The objective function (3.1) minimizes the makespan. Constraints (3.2) and (3.3), ensure that each job occupies only one position in the permutation sequence and that each position of the permutation sequence is allocated by a single job. Constraint (3.4) ensures equal processing time plus waiting time for every pair of adjacent machines. The existence of two artificially independent semi-lines is considered to determine the c^{l}_{j} and longer processing time among the semi-lines. Constraint (3.5) computes, from the second machine on, the idle time in each machine while waiting for the first job. Constraint (3.6) ensures that the idle time in each machine while waiting for the first job is less than or equal to the sum of the processing time in the previous machines. Constraint (3.7) ensures that there is no idle time for the job assigned to the first position. Constraint (3.8) ensures that the job idle time in the first position of the permutation, after finishing the processing in the machine *ml* is equal to the idle time of the first job before the synchronization machine $(ml+1)$ subtracted from the sum of the job idle time before the last machine in the problem l with the processing time of the first job in the last machine of the problem l. Constraint (3.9) determines the makespan of each problem as the sum of the processing time of all jobs in the machine plus the waiting time of the jobs in the last machine. Constraint (3.10) ensures the equality of the makespan of the semi-lines. Constraint (3.11) takes the value 1 if the job i is in the j^{th} permutation position and 0 otherwise. Constraint (3.12) ensures that the waiting time in the machines of each problem is nonnegative. Constraint (3.13) ensures that the makespan of each problem is nonnegative.

3.3 Heuristics based on Johnson's algorithm

Johnson's algorithm obtains a sequence that minimizes the makespan for the flow shop problem with two machines, Let p_{1j} (resp. p_{2j}) being the processing time of job j in the first (resp. second) machine. The optimal sequence begins with the jobs such that $p_{1j} \leq p_{2j}$ sorted in non-decreasing order of processing time, i.e., the downward part of the sequence. In this study, two adaptations of Johnson's algorithm were developed: Johnson's algorithm considering the longest processing time and Johnson's algorithm considering the average processing time. The general principle is to consider the studied system as a flow shop with two machines by setting the synchronization operation as the second machine.

3.3.1 Johnson's algorithm considering the average processing time

In the adaptation denoted by $John_{av}$, for each job $j = \{1, ..., n\}$, the average processing time is computed with: $p_j = \frac{\sum_{k=1}^{m1} P_{kj}^1 + \sum_{k=1}^{m2} P_{kj}^2}{m1+m2}$. The processing time p_{fj} of the synchronization operation is the processing time in the second machine. Then, Johnson's algorithm is applied and computed for the sequence obtained. The makespan is calculated in the whole system with the actual processing time p_{kj}^l for each semi-line $l = 1; 2$. The algorithm of $John_{av}$ is as it follows in the Procedure 3.1.

```
Procedure Johnson's algorithm with average processing time
1  J_i maq1 ← $\frac{\sum_{k=1}^{m1} P_{kj}^1 + \sum_{k=1}^{m2} P_{kj}^2}{m1+m2}$ ;
2  J_i maq2 ← $P_{fj}$ of the synchronization machine;
3  Determine $\min\{P_{jk}\}$ of $J_i maq1$ and $J_i maq2$ ;
4      If $Job_j$ with $min\{P_{jk}\}$ came from $J_i maq1$:
5          S1 ← Job$_j$;
6      If $Job_j$ with $min\{P_{jk}\}$ came from $J_i maq2$:
7          S2 ← Job$_j$;
8      If $P_{j1} = P_{j2}$, may be put in either Set ($S1$ or $S2$);
9  Repeat this procedure until the lists are exhausted;
10 Sort $S1$ in increasing order of $P_{fk}$ ($SPT$);
11 Sort $S2$ in decreasing order of $P_{fk}(LPT)$ ;
12 Sequence ← $S1 + S2$;
13 Calculate the Makespan of the Sequence;
end-Procedure
```

Fig. 3.1 Pseudo-code by Johnson's algorithm with average processing time

3.3.2 Johnson's algorithm considering the longest processing time

The adaptation denoted by $John_{hi}$ works in a similar manner, but the longest processing time $max_{l=1,2;k=1,...,ml} p^l_{kj}$ among the machines of the semi-lines is used as the processing time of job j in the first machine. The algorithm of $John_{hi}$ is as it follows in the Procedure 3.2.

```
Procedure Johnson's algorithm with longest processing time
1  J_i maq1 ← $max\{P_{jk}\}$ of the semi-lines;
2  J_i maq2 ← $P_{fk}$ of the synchronization machine;
3  Determine min$\{P_{jk}\}$ of $J_i maq1$ and $J_i maq2$ ;
4      If $Job_j$ with $min\{P_{jk}\}$ came from $J_i maq1$:
5          S1 ← Job_j;
6      If $Job_j$ with $min\{P_{jk}\}$ came from $J_i maq2$:
7          S2 ← Job_j;
8      If $P_{j1} = P_{j2}$, may be put in either Set ($S1$ or $S2$);
9  Repeat this procedure until the lists are exhausted;
10 Sort $S1$ in increasing order of $P_{jk}$ ($SPT$);
11 Sort $S2$ in decreasing order of $P_{jk}(LPT)$ ;
12 Sequence ← $S1 + S2$;
13 Calculate the Makespan of the Sequence;
end-Procedure
```

Fig. 3.2 Pseudo-code by Johnson's algorithm with the longest processing time

3.4 Heuristics based in *NEH* algorithm

In this study, three adaptations of the NEH algorithm were developed: NEH_{av} - the average processing time of jobs between correspondent parallel machines, NEH_{hi} - the longest processing time of the jobs among correspondent parallel machines, and NEH_{sep} - where each semi-line is considered separately including the synchronization machine. The NEH_{av} and the NEH_{hi} heuristics require the same number of machines in each semi-line, i.e, $m = m1 = m2$. Let f designates the final synchronization machine. The general principle is to reduce the two semi-lines in a single line and to apply the NEH heuristic.

3.4.1 *NEH* algorithm considering the average processing time

In the NEH_{av} heuristic, for each job $j = 1,...,n$, it is computed $(p^1_{im} + p^2_{im})/2$, where $k = 1,...,m$ is the kth-machine in each semi-line. At this point, there is a classical permutation flow shop with $m+1$ machines where, for each job j, p_{kj} is the processing time in machine

$k = 1,...,m$ and p_{fj} is the processing time in the last machine, and the *NEH* heuristic is applied in order to obtain a sequence seq_{av}. Finally, the makespan incurred by the sequence seq_{av} is computed in the whole system with the actual processing time p^l_{kj} for each semi-line $l = 1;2$.

```
Procedure $NEH_{av}$
1  For $i = 1$ to $n$ do
2      $s_i \leftarrow 0$
3      For $m = 1$ to $q$ do
4          $\bar{p}_{im} \leftarrow (p^1_{im} + p^2_{im})/2$
5          $s_i \leftarrow s_i + \bar{p}_{im}$
6      end-for
7      $s_i \leftarrow s_i + p_{if}$
8  end-for
9  Apply NEH algorithm to obtain a sequence $seq_{av}$.
10 Calculate the makespan of $seq_{av}$ in the system.
end-procedure
```

Fig. 3.3 Pseudo-code of the *NEH* variant considering average processing time

3.4.2 *NEH* algorithm considering the longest processing time

This variant considers the longest processing time of the job i in each k^{th} parallel machine between semi-line 1 and semi-line 2. The NEH_{hi} heuristic works in a similar manner. For

```
Procedure $NEH_{hi}$
1  For $i = 1$ to $n$ do
2      $s_i \leftarrow 0$
3      For $m = 1$ to $q$ do
4          $\bar{p}_{im} \leftarrow \max\{p^1_{im}, p^2_{im}\}$
5          $s_i \leftarrow s_i + \bar{p}_{im}$
6      end-for
7      $s_i \leftarrow s_i + p_{if}$
8  end-for
9  Apply NEH algorithm to obtain a sequence $seq_{hi}$.
10 Calculate the makespan of $seq_{hi}$ in the system.
end-procedure
```

Fig. 3.4 Pseudo-code of the *NEH* variant considering the highest processing time

each job $j = 1,...,n$, it is computed $\max\{p^1_{im}, p^2_{im}\}$, where $k = 1,...,m$ is the k^{th}-machine in each semi-line to obtain a sequence seq_{hi} by the NEH heuristic for a flow shop problem with $m+1$ machines. Then, the makespan is computed incurred by the sequence seq_{hi} in the whole system with the actual processing time p^l_{kj} for each semi-line $l = 1;2$. The algorithm of the NEH variant considering the average processing time is shown in the Procedure NEH_{av}. The algorithm of NEH variant considering the longest processing time is as it follows in the Procedure NEH_{hi}.

3.4.3 *NEH* algorithm considering the semi-lines separately

In the NEH_{sep} heuristic, each semi-line along with the synchronization machine separately was considered. The NEH heuristic to a classical permutation flow shop problem with $ml+1$ machines for each semi-line $l = 1;2$ is aplied, where, for each job j, p^l_{kj} is the processing time in the machine $k = 1,...,ml$ and p_{fj} is the processing time in the last machine, to obtain a sequence $seql_{sep}$. The sequence $seql_{sep}$, $l = 1;2$ is adopted leading to the shortest makespan in the whole system with the two semi-lines. The algorithm of the NEH variant considering the semi-lines separately is as it follows in the Procedure NEH_{sep}.

```
Procedure $NEH_{sep}$
1  while $line < 3$ do
2      Sort the tasks in descending order of $STP$, getting the list: $LC := \{\Pi_1, ..., \Pi_n\}$;
3      Select the first and second jobs of the sequence from $LC$;
4      Calculate the processing time of the two jobs by allocating them in all possible positions;
5      $\Pi := \Pi_j$ a partial sequence formed by the selected job;
6      Remove the jobs $\Pi(1)$ and $\Pi(2)$ from $LC$;
7      For $i = 3$ to $n$ do
8          Select job $\Pi_j$ randomly from $LCR$;
9          Enter the job $\Pi(j)$ in all possible positions of $\Pi$, generating $i$ partial
10             sequences with $i$ jobs;
11         $\Pi'$ : = Select the best generated sequence;
12         Remove the $\Pi_(j)$ from $LC$;
13     end-for
14     return ( $seq_{linha}$)
15     return ( $Mak_{linha}$)
16 end-while
17 return ( $min\{Mak_{linha}\}$)
18 return ( $seq_{linha}$ of $min\{Mak_{linha}\}$)
end-procedure
```

Fig. 3.5 Pseudo-code of the NEH separately

3.5 Resolution methods with metaheuristics

Three different algorithms using metaheuristics were elaborated in this study. The initial solution, the perturbation methods, the local search, and the modified acceptance criterion solution were developed to each of them.

3.5.1 Representation of a solution

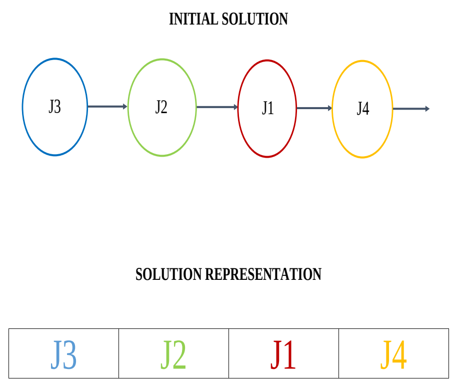


Fig. 3.6 Representation of a solution

The solution to the flow shop problem with n jobs will be represented by a vector Seq of n positions, each position Seq_i indicates the order of production of the i_{th} job. The sequence $Seq = (3,2,1,4)$ is shown in Figure 3.6 as an example.

Neighborhood generation

The purpose of this process is to find neighboring solutions of a previously obtained solution. They are constructed from position changes in the current solution to find out solutions that are viable. This study used the simple disturbance. This movement is done by drawing two changes of positions. Once the two change positions are obtained, the jobs that occupy these positions are exchanged. Figure 3.7 shows an example of this movement, where the job that occupied the first position was exchanged for the second position.

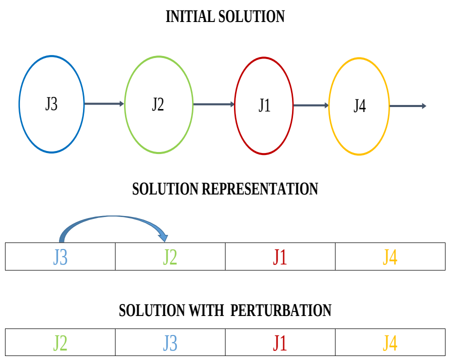


Fig. 3.7 Neighborhood structure with exchange of two processing order jobs

Generation of the initial solution

In this study, two initial solution methods were considered. The first method generates the initial population with algorithm NEH separately shown in the section 3.4 and the second method generates the initial population by Johnson's algorithm with average processing time method shown in the section 3.3.1.

3.5.2 Simulated annealing

The heuristic Simulated Annealing was an adaptation of the method by Hurkała [42] and the pseudo-code is described in Figure 3.8. The algorithm proceeds by generating disturbances in the sequence of the jobs and providing another neighbor solution (S'). These changes are made as shown in the sections 3.3.1 e 3.4.3. A new change is made at each iteration and a new solution is generated. The generated solutions undergo an assessment process to check if the objective of minimizing the makespan has been achieved. The makespan is calculated and assessed. The solution (S) is updated if the acquired solution is better than the others found until then. However, the algorithm tends to get stuck in local optima when only the best solutions are accepted. To prevent this, the algorithm allows the acceptance of the worst solutions. This acceptance is made by respecting a probability in relation to the temperature. The function is calculated as shown in Jarosaw et al. [43] who chose a real number between 0 and 1 randomly. The worsening solution is accepted if this value is lower than $e^{-\delta/T}$, where T is the temperature and δ is the difference between the values of the accepted solution and the previously adopted solution. The algorithm stopping criterion is determined by the slow cooling of the initial temperature. The procedure is repeated in a number of times per temperature. The algorithm starts a local search in the sequence that generated the best solution when the temperature reaches the minimum point. The pseudo-code of the algorithm used in this study is shown in 3.8:

```
Procedure Simulated Annealing
1  Select an initial temperature $T \leftarrow T_0$;
2  Select an initial solution $S \leftarrow S_0$;
3  $it \leftarrow 0$;
4  $best_{it} \leftarrow 0$;
5  $S_{best} \leftarrow S_0$;
6  While $(\mathtt{it} \leq \mathtt{best_{it}} + \mathtt{max_{it}})$ do
7      $it_{sol} \leftarrow it$;
8      While $(\mathtt{it} \leq \mathtt{it_{sol}} + \mathtt{it_{temp}})$ do
9          $it++$;
10         $S' \leftarrow N(S)$
11         $\triangle \leftarrow cost(S') - cost(S)$;
12         If $\triangle \leq 0$ then :
13             $S \leftarrow S'$
14             $it_{sol} \leftarrow it$
15             If $\mathtt{cost(S')} < \mathtt{cost(S_{best})}$ then :
16                 $S_{best} \leftarrow S'$
17                 $best_{it} \leftarrow it$
18             end − if
19         end − if
20         Else
21              Generate random number $r$
22             If $(r < e^{\triangle/T})$Then :
23                 $S \leftarrow S'$;
24             end-if
25         end
26     end
27     $T \leftarrow \alpha T$
28 end
29 Return S;
30 Apply Local Search in the best Sequence;
end-procedure Simulated Annealing
```

Fig. 3.8 Pseudo-code of the Simulated Annealing with LS

3.5.3 Local search

```
Procedure Local Search (s,n )
$Seq_0 \leftarrow$ best Construction sequence;
$C(s) \leftarrow$ best Construction makespan;
$OK \leftarrow 1$
    while $OK == 1$ do
    $OK \leftarrow 0$
    $Best \leftarrow \infty$
        for1 $(i = 0$ to $n-1)$do
            for2 $(j = i$ to $n)$do
            $Seq' \leftarrow Seq +$ exchange of the position
            element $i$ by position element $j$;
            Calculate the cost $C(seq')$
            if $(C(seq') <$ Best$)$ them
                Best $\leftarrow C(seq')$
                seqviz $\leftarrow seq'$
            end-if
            end-for2
        end-for1
    if (Best $< C(seq'))$ them
        $seq \leftarrow seqviz$
        $Ok \leftarrow 1$
    end-if
    end-while
    return ( $seq$)
    return ( $Best$)
end-procedure
```

Fig. 3.9 LS Pseudo-code

In local search, the neighborhood space is explored in order to improve the objective function. If there are opportunities for improvement, the exchange to generate the best result is adopted. In this study, the algorithm starts with the generation of a feasible solution (S_0) by two different methods (by the NEH algorithm considering the separated semi-lines and by Johnson's adaptation considering the average of the semi-lines) shown in the section 3.3.1 and 3.4.3. The local search process starts from this solution. This procedure is an adaptation of the study by Ruiz and Sttzle [88]. It consists of taking a job from the original position and inserting it in other positions of the sequence and calculating the objective function of each modified sequence. The nearby solution to generate the best makespan is selected and becomes the new current solution (S'). This process continues as long as there is improvement in the current solution. The process is interrupted if there is a number of iterations without improvement. The pseudo-code is in Figure 3.12.

3.5.4 Greedy Randomized Adaptive Search Procedure - GRASP

The *GRASP* algorithm to the problem under study consists of two phases: the construction phase of the initial solution and the phase of local search or solution improvement. The first phase consists of constructing a viable solution. After, the solution acquired in the previous stage undergoes to a search in the neighborhood in order to achieve the local minimum. The best solution is stored as a partial result. At the end of all iterations, the best of the partial results is adopted. (Resende and Ribeiro [83]) The pseudo-code 6.4 shows the *GRASP* steps.

```
Procedure GRASP ($Maxiter$), ($Seed$)
1  Read input()
2  $iter \leftarrow 0$;
3      while $iter < maxiter$ from
4          $Solution \leftarrow Construction(seed)$;
5          $Solution \leftarrow LocalSearch(Solution)$;
6          Update-Solution $(Solution, iteratit \{Best - Solution\})$;
7      end-while
8      return (Best-Solution)
9 end-procedure GRASP
```

Fig. 3.10 GRASP Pseudo-code

The proposed GRASP-NEH algorithm

The proposed $GRASP - NEH$ algorithm follows all steps of the classical *GRASP* algorithm presented in 6.4. The construction phase starts with an empty scale and a list of candidates that will form the initial sequence considering the semi-line 1 and the semi-line 2 separately with the synchronization machine. In this algorithm, the list of candidates (*LC*) is formed by the jobs sequence in decreasing order of the processing time value in each semi-line l. The Restricted List (*RL*) is generated from the selection of the $\alpha\%$ of the candidates of the *LC* sequence. Then, one of the jobs is selected randomly from the *RL* and the allocation of the jobs in the final sequence is made in accordance with the *NEH* algorithm. $GRASP - NEH$ construction algorithm is shown in Figure 3.11.
The heuristic selects an element iteratively to the scale of the semi-line1 and of the semi-line2 through nine basic steps:

```
**Procedure** Constrution-NEH ( $itermax, \alpha$ )
1  $Solution1 \leftarrow \emptyset$;
2  $Solution2 \leftarrow \emptyset$;
3  Calculate $\sum pr_{ij}$ of the job $i$ in the machine $j$ in the semi-line $l$;
4  $LC_l \leftarrow$ descending order of the jobs for each semi-line $l$;
5  $RL_1 \leftarrow$ **the** $\alpha\%$ **of the jobs of LC**1;
6  $RL_2 \leftarrow$ **the** $\alpha\%$ **of the jobs LC**2;
7  **while** $Seq1$ and $Seq2$ is not complete  bf do:
8      Select randomly a job lists $RL1, RL2$;
9      Allocate the selected jobs in the sequences $(Seq1, Seq2)$
10         According to the NEH heuristic;
11     Update $RL_1$ e $RL_2$
12 **end-while**
13     Solution1 $\leftarrow$ $makespan\ Seq1$;
14     Solution2 $\leftarrow$ $makespan\ Seq2$;
15     Adopt a sequence of smaller makespan;
16     Return (best makespan);
17     Return (best sequence);
**end-procedure Construction-NEH**
```

Fig. 3.11 Construction pseudo-code

1. Calculate the job processing time in each semi-line separately;

2. Classify the jobs in decreasing order of the processing time;

3. Select the jobs according to a certain percentage (α) to form the restrict list of candidates (*RL* 1);

4. Select the jobs according to a certain percentage (α) to form the restrict list of candidates (*RL* 2);

5. Choose an element of the restrict list (*RL*1) randomly to compose the sequences, arranging them according to the principles of the *NEH* algorithm;

6. Select an element of the restrict list (*RL*2) randomly to compose the sequences, arranging them according to the principles of the *NEH* algorithm;

7. Update the restrict lists after selecting each element until all the elements are inserted;

8. Calculate the makespan from the sequences generated by the semi-line 1 and the semi-line 2;

9. The sequence to generate the lowest makespan is adopted;

After that, *GRASP* local search starts where the initial sequence is acquired in the previous step. This step seeks local optimal solutions in each iteration. All possible neighbors of the ”*s*” current solution are analyzed moving only to the neighbor that has the most favorable makespan value considering the objective function. A list of the best local partial solutions is formed so that the lowest partial result will be used as the final solution. The pseudo-code for the generation of the best neighbor is shown in Figure 3.12.

```
Procedure Local  Search (s,n )
1  $Seq_0 \leftarrow$ best Constrution sequence;
2  $C(s) \leftarrow$ best Constrution makespan;
3  $OK \leftarrow 1$
4      while $OK == 1$ do
5      $OK \leftarrow 0$
6      $Best \leftarrow \infty$
7          for1 $(i = 0$ to $n-1)$do
8              for2 $(j = i$ to $n)$do
9              $Seq' \leftarrow Seq +$ exchange of the position
10             element $i$ by position element $j$;
11             Calculate the cost $C(seq')$
12             if $(C(seq') <$ Best$)$ them
13                 Best $\leftarrow C(seq')$
14                 seqviz $\leftarrow seq'$
15             end-if
16             end-for2
17         end-for1
18     end-while
19     return ( $seq$)
20     return ( $Best$)
end-procedure
```

Fig. 3.12 Local search pseudo-code

The GRASP algorithm with Johnson’s algorithm

The construction phase of *GRASP*-Johnson algorithm considers the highest processing average time of the job *i* in all the machines of the semi-lines and the time of the synchronization machine. At this stage, the algorithm applies the adaptation of Johnson’s rules shown in **??**.

```
Procedure Construction ( $itermax, \alpha$ )
1  $Solution \leftarrow \emptyset$;
2  machine1 $\leftarrow$ increasing order of highest value
3      of $\sum prijl/m$ of the semi-line jobs;
4  machine2 $\leftarrow$ increasing order of $prij$ sync of the jobs
5      $RL_1 \leftarrow \alpha$ % jobs of the machine1;
6      $RL_2 \leftarrow \alpha$ % jobs of the machine2;
7  while Seq is not complete do:
8      Randomly select a job of the $(RL1)e$(RL2);
9      $seq \leftarrow$ seq de Johnson;
10     Update $RL_1$ e $RL_2$
11 end-while
12 Solution$\leftarrow$ makespan $seq$;
13 Return( Solution);
end-procedure Construction
```

Fig. 3.13 Construction pseudo-code

In this sense, the highest average processing time of the jobs in the machines of the semi-lines is used as the processing time of the jobs from Johnson's first machine. The job processing time of the synchronization machine is regarded as the processing time of Johnson's second machine. The heuristic selects an element iteratively for each scale through the following steps:

1. Form the list of candidates $LC1$ with the jobs of Johnson's first machine in ascending order;

2. Form the list of candidate $LC2$ with the jobs of Johnson's second machine in ascending order;

3. Select the jobs of $LC1$, according to a certain percentage (α) to form the restrict lists of candidates $RL1$;

4. Select the jobs of $LC2$, according to a certain percentage (α) to form the restrict lists of candidates $RL2$;

5. Draw one element of the restricted list ($RL1$) to compose the sequence;

6. Draw one element at a restricted list ($RL2$) to compose the sequence;

7. The jobs drawn of the list $RL1$ will be allocated at the beginning of the sequence.

8. The jobs drawn of the list *RL*2 will be allocated at the end of the sequence;
9. Update the restrict lists after selecting each element until all elements are inserted.

The pseudo-code generation construction by Johnson's algorithm is shown in 6.6. The sequence acquired at this stage from GRASP Johnson's algorithm is used as the initial solution of the local search. This stage seeks local optimal solutions. In each iteration, all possible neighbors of the "*s*" current solution are analyzed, moving only to the neighbor that has the most favorable makespan value considering the objective function. A list of the best local partial solutions is formed so that the lowest partial result will be used as the final solution. The pseudo-code for generating the best neighbor is the 3.12.

3.6 Conclusion

In this chapter, methods for solving the flow shop scheduling with parallel semi-lines and final synchronization operation are exhibited. The mathematical model for the problem as well as the methodologies of Johnson's rule, the *NEH* heuristics and the methods using local search *ILS*, simulated annealing and *GRASP* and the pseudo-code of these methods are shown in order to understand how the heuristic methods are applied to the problem.

Chapter 4

COMPUTATIONAL EXPERIMENTS

This chapter presents and analyzes the achieved results to solve the flow shop problem with parallel semi-lines with final synchronization operation. Section 4.1 presents the generation procedure of the test problems (instances) to analyze the problem. Section 4.2 shows the results for mathematical modeling. Section 4.3 presents the results achieved for the problem by adaptation of Johnson's rule and the *NEH* algorithm, and section 4.4 shows the results achieved by the metaheuristics local search, simulated annealing and greedy randomized adaptive search procedure. Finally, a comparison of the results from the proposed methods is done in order to determine which methods had better results for the studied environment.

4.1 Generation of instances

As in most studies on production scheduling and planning (eg Li et al.[50], Ruiz and Maroto [87]), the processing time was generated within the range of $U\ [1-99]$. The different instances used in this study were generated with the same principle of the method reported by Taillard [101]. The instances were composed of processing time d_{ij} of job i in the machine j $(1 < i < n, 1 < j < m)$. The values of d_{ij} are randomly created with a good generator of random number. The instances were defined by an initial seed of the random generator, the number of jobs, the number of machines, the maximum amount of time (upper limit), and the minimum amount of time (lower limit). The algorithm by Taillard [101]) was adapted to this problem considering the formation of a single line with the number of jobs, the machine number of the semi-line 1, the machine number of the semi-line 2, and the synchronization machine. The used algorithm is shown in the pseudo-code of the instance generation 4.1.

An example of instances is shown in Table 4.1. It represents an instance of a manufacturing line that contains two semi-lines and one synchronization machine, so that the semi-line 1

```
Procedure Instance Generation
1     m ← 2147483647 ;
2     a ← 16807;
3     b ← 127773;
4     c ← 2836;
5     njobs ← number of jobs;
6     nmachines ← number of machines(semi-line1 + semi-line2 + sincronization);
7       For $i = 1$ to $njobs$ do
8         For $j = 1$ to mmachine do
9           k ← seed div b;
10          seed ← a * (seed mod b) - k * c ;
11          If seed < 0 then:
12          seed ← seed+m;
13          endif
14        value ← seed/m;
15        unif ← low + (value * (high - low + 1));
16        matrix ← unif;
17        End-for
18      End-for
end-procedure
```

Fig. 4.1 Pseudo-code of the instance generation

	Semi-line1			Semi-line2		Synchronization
Jobs	Maq1	Maq2	Maq3	Maq4	Maq5	Maq6
Job1	10	5	5	15	10	15
Job2	5	5	10	10	5	10
Job3	10	5	15	20	15	5
Job4	10	15	20	25	10	10

Table 4.1 A sample of instances

has three machines and the semi-line 2 has two machines. In table 4.1, the processing time of the jobs in each machine of the semi-lines is shown. The first column shows the jobs that will be processed throughout the production line. The remaining columns present each processing time of the job in a given machine of the semi-line 1 ($maq\ 1, maq\ 2, maq\ 3$), the semi-line 2 ($maq\ 4, maq\ 5$) and the synchronization machine ($maq\ 6$). For the sequence $S = J3, J2, J1, J4$, the completion time of each job in each machine is shown in table 4.2 and Gantt's Graph is shown in Figure 4.2.

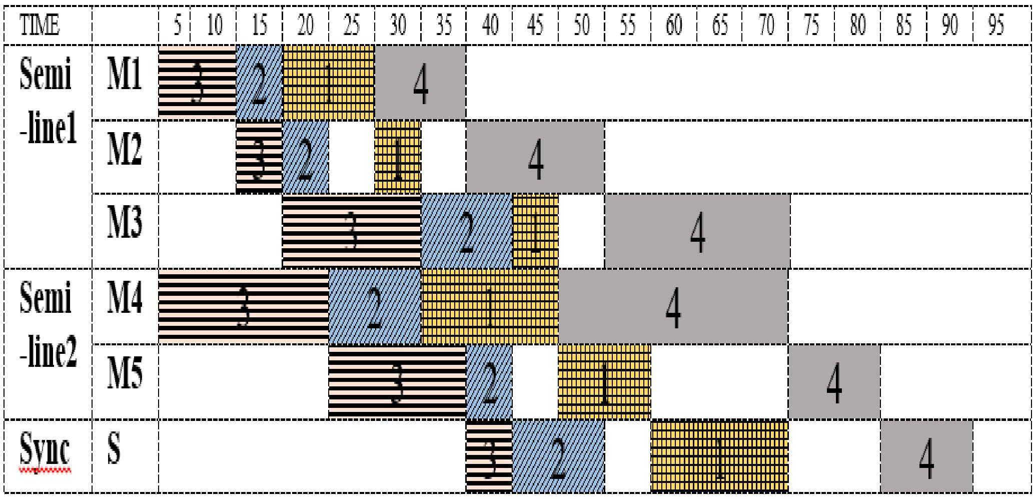


Fig. 4.2 Gantt's Graphs of the sequence $J3, J2, J1, J4$

The Makespan = 90 ($C3 = 40, C2 = 50, C1 = 70, C4 = 90$). The makespan and the waiting time of each job in the semi-line 1, in the semi line 2 and in the synchronization machine are shown in Table 4.2.

	Semi-line1	Semi-line2	Synchronization
Jobs	J1– J2–J3–J4	J1– J2–J3–J4	J1– J2–J3–J4
Makespan	30– 40– 45–70	35–40–55–80	40–50–70–90
Waiting time of the jobs	5—10–20–10	0—5—0—0	0—0—0—0
Time that a half waits the other	5 —0–10—10	0—0 —0 —0	x—x—x —x

Table 4.2 Makespan and waiting time

4.2 Criteria used to achieve the computational experiments

The different methods to solve the flow shop problem with parallel semi-lines with a final synchronization operation were tested among the different instances. There are 240 different instances. They are composed by combinations of number of jobs x number of machines of the semi-line 1 x number of machines of the semi-line 2. There are 10 different instances for each combination. The first 130 instances were created considering that the two semi-lines have the same number of machines. The other 110 instances were created considering different numbers of machines in the semi-lines. For these problems, 10, 20 and 50 jobs and 3, 5, 7 and 11 machines were considered. Among the 240 analyzed instances, the optimal solution was not found in 41 instances. The presented tests were conducted using a computer with the following settings: Intel C*ore TM iR*3.1 GHz with 4*GB* of memory. The results of the mathematical model were achieved using the software *CPLEX*12.6.1 and the algorithms were implemented in $C++$. The statistical analysis of the results was acquired using the Minitab 17.2.1. software. To compare the results, the absolute relative deviation (*GAP*) was used as in Ruiz and Maroto [87], which in this case is the variation between the optimal solution and the created method solutions. The *GAP* will be calculated by the equation: 4.1.

$$GAP = \frac{Met_{sol} - OPT_{sol}}{OPT_{sol}} * 100 \tag{4.1}$$

where: Met_{sol} corresponds to the makespan achieved by a proposed method; and OPT_{sol} is the makespan achieved with the mathematical model.

The standard deviation of mean relative deviation was also used to compare the best methods. The standard deviation of a sample measures the degree of dispersion of the elements around the mean. In this study, the standard deviation of the *GAP* is the value of the variation of the relative deviations of a class of problems around the mean relative deviation. The lower the value of the standard deviation the better the method applied, when comparing one method with another. The standard deviation is calculated by the equations 4.2 and 4.3:

$$GAP_{inst} = \frac{1}{Ninst} * \sum_{i=1}^{Ninst} (GAP_i - \overline{GAP_i})^2 \tag{4.2}$$

$$SD = \sqrt{GAP_{inst}} \tag{4.3}$$

where: $Ninst$ corresponds to the instance number; AVG_{inst} to the the absolute relative deviation to each set of instance; AVG_i is the absolute relative deviation of the instance i; and SD is the standard deviation.

The tests were conducted in two environments. In the first, the same number of machines in each semi-line was considered and in the second, different numbers of machines in the semi-lines were considered. A limit time of $7,200$ seconds to achieve an optimal solution was considered.

4.3 Mathematical modeling results

Two parallel semi-lines followed by a sequential line were considered for the mathematical approach of the problem by grouping the operations into parallel machines and by adopting as processing time a corresponding operation or set of operations of the slower group. A dummy machine (considered in the model as synchronization machine) is included at the end of each semi-line. These two dummy machines represent the last operation as if this last operation was processed in two independent machines. The existence of two artificially independent semi-lines will allow the determination of the highest value of c_n^1 the total processing time (makespan) between them.

Table 4.3 results for different instances with the same number of machines and for the different number of machines

Instances	time(s)	opt	Instances	time(s)	time(s)
Equal (E)	Aver.	%	Different(D)	Aver.	%
E10×03	0.26	100	Dl0×03 × 05	0.12	100
E10×05	0.48	100	Dl0×03 × 07	0.17	100
E10×07	1.94	100	Dl0×03 × 11	0.94	100
E10×11	10.75	100	Dl0×05 × 07	0.32	100
E20×03	0.34	100	Dl0×05 × 11	1.43	100
E20×05	405.01	100	D20×03 × 05	4.11	100
E20×07	2289.06	100	D20×03 × 07	52.94	100
E20×11	7541.78	40	D20×03 × 11	14250.2	40
E50×03	5.76	100	D20×05 × 07	200.1	100
E50×05	4915.71	80	D20×05 × 11	18208.3	20
E50×07	13695.50	10	D50×03 × 05	153.78	100
E50×11	12746.20	0	***********	******	***
E100×03	160.69	100	***********	******	***
Average	3213.34	79.2%	Average	2988.41	87.3%

The achieved results for different instances with the same number of machines (E) and for the different number of machines (D) is shown in Table 4.3 and in Figure 4.3. The tables show the average computational time of the mathematical model in seconds and the achieved percentage of optimal values (opt). Through the results, it was observed that for smaller

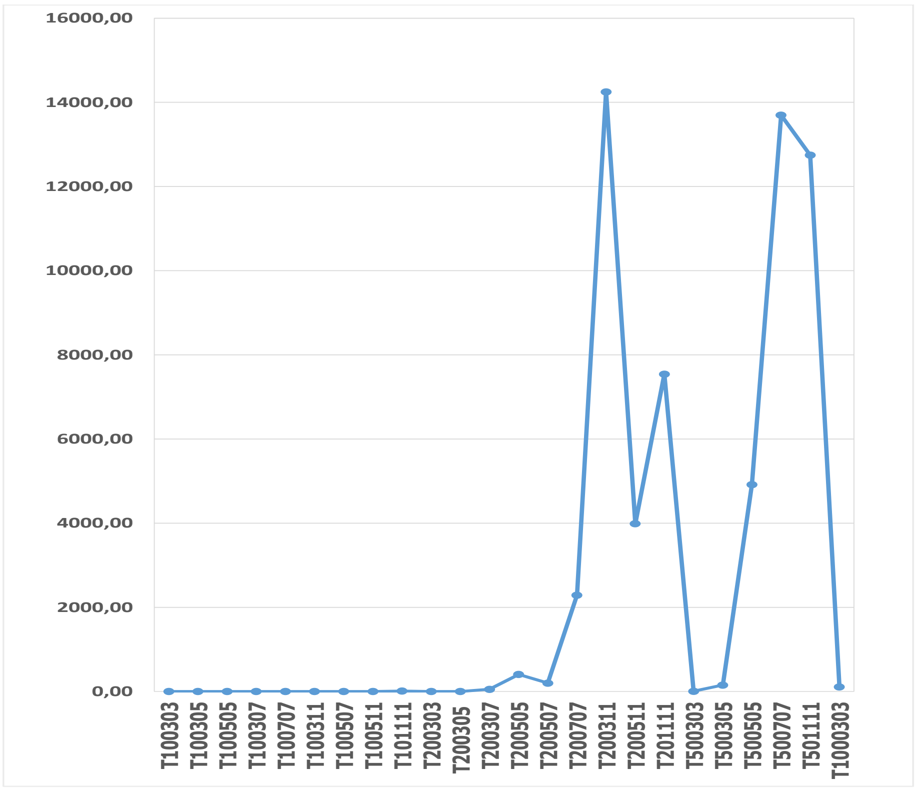


Fig. 4.3 Average time for each instance set to find the makespan with the mathematical modeling (sec.)

number of jobs and machines, the makespan was found in low computational time. On the other hand, it was not possible to find out the optimal solution for larger instances with low computational time. Due to this fact, other approach strategies for the problem will be used (constructive heuristics such as the NEH algorithm and metaheuristics).

4.4 Results by the adaptation of Johnson's algorithm and the *NEH* algorithm

This section presents the achieved results by the adaptations of the *NEH* heuristic and Johnson's algorithm. The proposed methods are tested among different instances with environment of same number of machines in each semi-line, where three adaptations of the *NEH* heuristic (NEH_{av}, NEH_{hi}, NEH_{sep}) and two adaptations of Johnson's algorithm ($John_{hi}$, $John_{av}$) were considered. In the second environment, different numbers of machines in the semi-lines, NEH_{sep}, $John_{hi}$ and $John_{av}$, were considered.

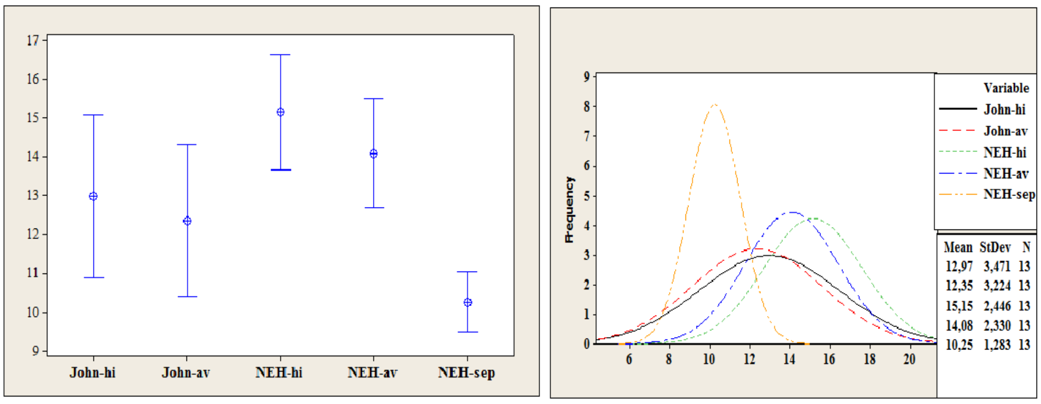
	John-hi	John-av	NEH-hi	NEH-av	NEH-sep
Instances	gap- sd -time	gap -sd -time	gap -sd- time	gap -sd -time	gap -sd -time
$E10 \times 03$	6.97 -0.06 - 3.90	9.28 - 0.04 -3.90	13.2 -0.03 -7.20	15.2 -0.04 -4.30	11.4 -0.10 -6.70
$E10 \times 05$	12.9 -0.04 - 4.40	14.1 - 0.06 -4.40	13.9 -0.03 -7.50	15.2 -0.03 -5.70	11.1 -0.05 -8.50
$E10 \times 07$	16.0 -0.02 - 5.20	10.0 - 0.05 -5.20	16.0 -0.02 -6.10	11.0 -0.05 -5.20	9.20 -0.05 -8.60
$E10 \times 11$	15.0 -0.03 - 7.20	13.0 - 0.05 -7.20	15.0 -0.03 -7.30	11.0 -0.05 -6.90	10.0 -0.03 -9.80
$E20 \times 03$	8.40 -0.04 - 4.60	9.10 - 0.04 -4.60	16.3 -0.01 -6.80	14.5 -0.03 -6.90	9.30 -0.04 -11.9
$E20 \times 05$	15.2 -0.02 - 5.50	16.9 - 0.03 -5.50	16.9 -0.03 -9.10	12.3 -0.06 -8.20	10.9 -0.03 -16.5
$E20 \times 07$	11.5 -0.05 - 8.60	17.7 - 0.06 -8.70	12.3 -0.02 -9.80	16.9 -0.04 -9.90	11.3 -0.04 -21.3
$E20 \times 11$	17.3 -0.02 - 11.6	14.9 - 0.04 -11.6	18.6 -0.06 -11.9	16.9 -0.05 -10.4	10.6 -0.10 -36.5
$E50 \times 03$	7.60 -0.05 - 9.90	7.20 - 0.06 -15.7	16.8 -0.03 -15.7	12.4 -0.09 -10.2	13.1 -0.03 -41.6
$E50 \times 05$	15.1 -0.03 - 20.2	13.2 - 0.04 -20.2	17.6 -0.02 -10.2	13.6 -0.09 -9.00	9.30 -0.03 -44.2
$E50 \times 07$	14.2 -0.04 - 22.2	14.5 - 0.02 -22.2	17.4 -0.07 -34.2	15.2 -0.04 -19.4	8.60 -0.05 -45.4
$E50 \times 11$	16.5 -0.05 - 31.8	10.4 - 0.05 -31.6	11.7 -0.07 -41.6	17.6 -0.12 -43.6	9.60 -0.07 -67.4
$E100 \times 03$	11.9 -0.03 -37.1	10.3 - 0.08 -36.8	11.2 -0.07 -47.9	11.3 -0.05 -50.5	8.90 -0.05 -72.2
Average	13% -0.03 -13.2	12% -0.02 -12.6	16% -0.02 -16.6	14% -0.02 -14,6	10% -0.05 -26.90

Table 4.4 Gap, sd, time (s) of Johnson's and the *NEH* adaptations for same number of machines (E)

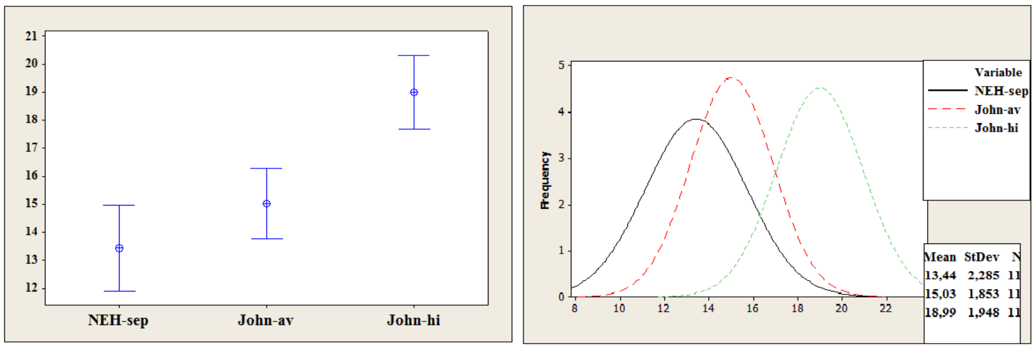
The achieved results for different instances with the same number of machines (*E* is shown in Table 4.6 and for different number of machines (*D*) in Table 4.7. The tables show the average relative deviation (*GAP*), the standard deviation (*SD*) and the average computational time in seconds (*CPU*) for each set of instance.

Observing the environment in which the number of machines is the same, it can be asserted that the *NEH* adaptations that showed the best performance were those used the *NEH* in the semi-lines separately (NEH_{sep}) with *GAP* average of 10% and standard deviation of 0.05. The variant that showed the worst result was that which used the *NEH* with the longest processing time (NEH_{hi}) in each parallel machine with *GAP* average of 16% and standard deviation of 0.02.

In relation to Johnson's variants, it can be observed a small difference between the deviation of $John_{hi}$ adaptation and $Jonh_{av}$. The variant with the best performance was that



a- Same number of machine



b- Different number of machine

Fig. 4.4 Gap and sd for *NEH* and Johnson's adaptation for same and different number of machine

Instances	NEH-sep gap- sd -time	John-av gap -sd -time	John-hi gap -sd- time
$D10 \times 03 \times 05$	13.3 -0.10 - 3.40	14.3 - 0.04 -2.90	17.2 -0.06 -4.20
$D10 \times 03 \times 07$	12.0 -0.09 - 3.60	15.2 - 0.07 -3.64	17.2 -0.06 -4.52
$D10 \times 03 \times 11$	12.9 -0.07 - 4.20	14.9 - 0.04 -4.34	17.8 -0.04 -4.10
$D10 \times 05 \times 07$	12.4 -0.05 - 5.20	15.7 - 0.04 -5.33	16.1 -0.08 -5.30
$D10 \times 05 \times 11$	16.0 -0.08 - 4.63	17.3 - 0.05 -4.12	19.6 -0.06 -4.80
$D20 \times 03 \times 05$	14.0 -0.07 - 4.50	15.0 - 0.04 -4.12	19.0 -0.07 -4.10
$D20 \times 03 \times 07$	10.0 -0.07 - 3.60	12.0 - 0.04 -4.71	20.1 -0.07 -5.80
$D20 \times 03 \times 11$	16.0 -0.06 - 5.81	14.9 - 0.07 -5.63	21.0 -0.05 -5.23
$D20 \times 05 \times 07$	14.0 -0.06 - 5.92	16.0 - 0.10 -5.74	18.8 -0.05 -5.88
$D20 \times 05 \times 11$	17.0 -0.05 - 6.23	18.0 - 0.07 -6.26	23.0 -0.03 -7.20
$D50 \times 03 \times 05$	10.2 -0.05 - 11.04	12.0 - 0.08 -11.10	19.1 -0.07 -11.19
Average	14% -0.07 -5.3	15% -0.06 -5.3	19% -0.06 -5.7

Table 4.5 Gap, sd, time (s) of Johnson's and NEH adaptations for different number of machines (D)

which considered the longest average processing time $Jonh_{av}$ in all the machines of the semi-lines for the first machine and the processing time in the synchronization machine for the second machine.

For the environment where the number of machines is different in the semi-lines, NEH_{sep} performed better with 14%. Joh_{hi} variant presented a GAP average slightly higher than $John_{av}$ (19%). To check the statistical validity of the results, an analysis of the variance ($ANOVA$) with a confidence level of 95% was taken as shown in Figure 4.4. However, these results can be improved in order to be closer to the optimal solution. Therefore, metaheuristics have been used to solve the problem in order to achieve optimal or near optimal results.

4.5 Metaheuristics results

4.5.1 Results by the iterated local search

The results by iterated of local search is presented considering as initial solution two of the methods presented in section 4.4. The first method considers Johnson's algorithm with the average processing as an initial solution ($John_{av}$), and the second considers NEH_{sep} algorithm as the initial solution.

The type of neighborhood used in this method was the reinsert move, where a neighboring sequence is basically achieved from the current sequence by removing a job from its position and inserting it in another position. The input parameter values of the algorithms were

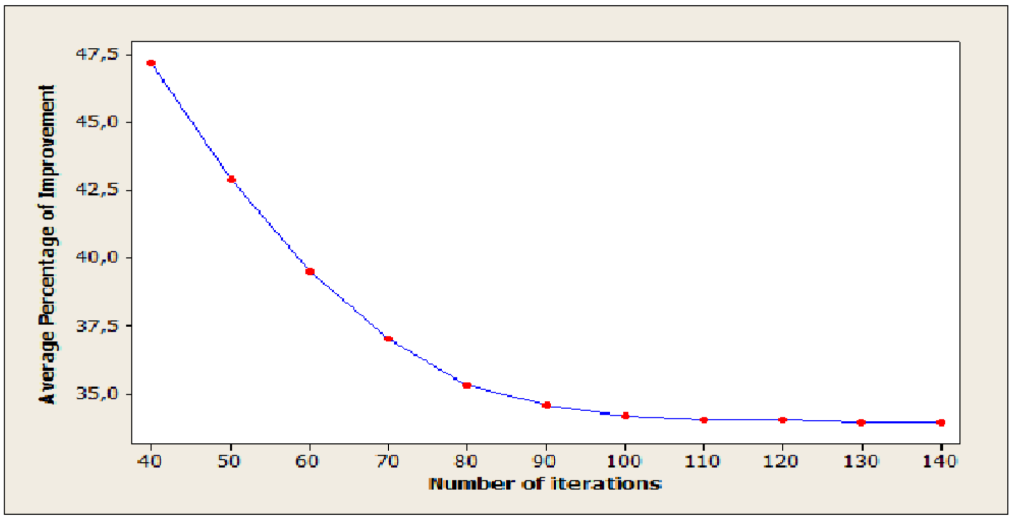


Fig. 4.5 Variation in the average percentage of improvement for each set of instances

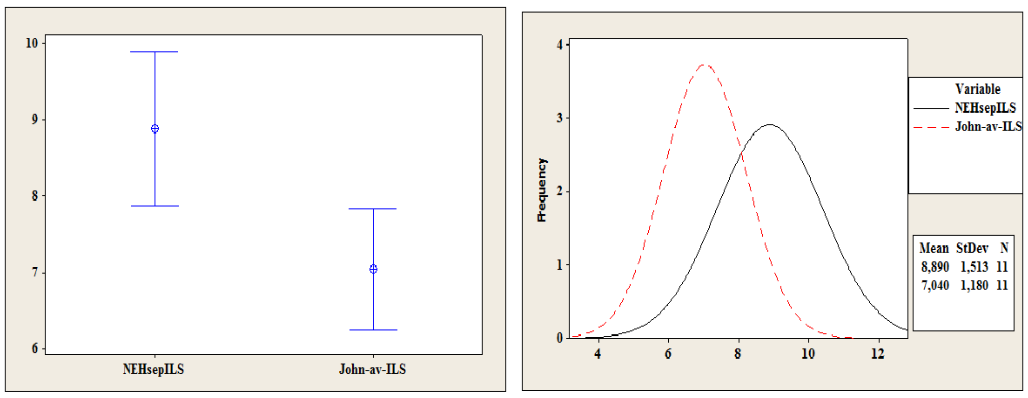
selected after some preliminary experiments as observed by Gomes Jr. et al. [36] and Nogueira et al. [10].

Stopping criterion of the *ILS* algorithm is the number of iterations with lower percentage of improvement. To define the number of iterations, some tests with this algorithm were made. From 20 to 140 iterations with minor improvements were considered. In this initial

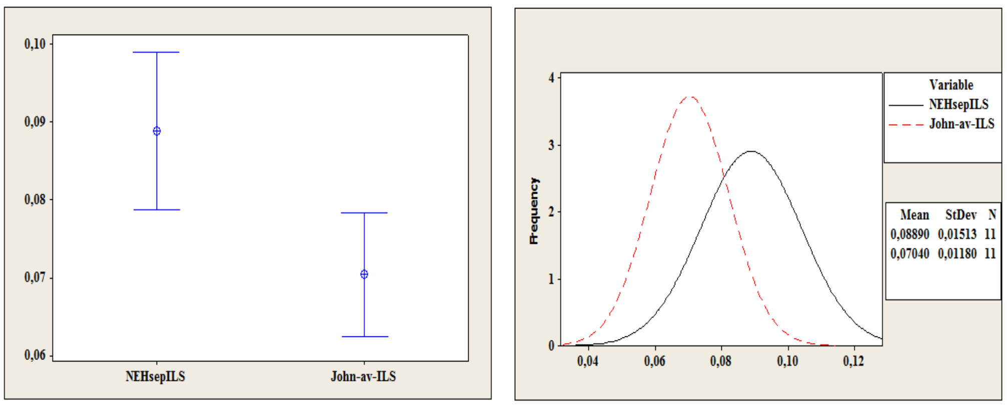
	NEH-sep-ILS	John-av-ILS
Instances	gap- sd -time	gap -sd -time
$E10 \times 03$	7.17 -0.04 - 4.05	4.60 - 0.02 -3.45
$E10 \times 05$	8.97 -0.02 - 5.40	5.78 - 0.02 -4.22
$E10 \times 07$	6.56 -0.02 - 6.10	4.57 - 0.02 -5.20
$E10 \times 11$	6.55 -0.03 - 6.16	6.46 - 0.03 -5.81
$E20 \times 03$	7.56 -0.03 - 6.12	6.46 - 0.02 -4.94
$E20 \times 05$	7.60 -0.03 - 7.10	6.61 - 0.03 -6.60
$E20 \times 07$	8.10 -0.03 - 7.62	8.85 - 0.03 -8.70
$E20 \times 11$	7.69 -0.04 - 8.69	6.59 - 0.03 -10.6
$E50 \times 03$	7.60 -0.05 - 19.24	7.20 -0.06 -15.7
$E50 \times 05$	7.69 -0.03 - 19.23	6.93 -0.04 -16.3
$E50 \times 07$	8.94 -0.03 - 18.22	6.53 -0.03 -18.2
$E50 \times 11$	5.75 -0.03 - 22.80	5.07 -0.04 -19.6
$E100 \times 03$	4.58 -0.02 -37.10	4.31 - 0.02 -38.6
Average	6.78% -0.03 -12.52	6.15% -0.02 -12.14

Table 4.6 Gap, sd and time (s) of Johnson's and the *NEH* adaptation for the same number of machine (E)

test ten instances were considered ($E10X03X03$, $E10X11X11$, $E20X03X03$, $E20X11X11$, $E50X7X7$, $D10X03X11$, $D20X03X05$, $D20X05X07$, $D20X05X11$, $D50X03X05$) and the average percentage of improvement for each of them. In the graph, the average percentage of improvement is obtained by the average of the percentages of improvement obtained by the ILS method to the initial solution. The percentage of improvement is the difference of the percentage between the initial solution value and the solution value with the ILS method. In the graph of Figure 4.5, it can be observed that from the point where the iteration is equal to 90 the variation in the average percentage of improvement is low, then, the maximum number of iterations equal to 90 was adopted.



a- GAP and SD for Same number of Machine - ILS



b- GAP and SD for different n umber of Machines - ILS

Fig. 4.6 Gap and sd for $NEH_{sep} - ILS$ and $John_{av} - ILS$ adaptation

The results achieved for different instances with the same number of machines in each semi-line are shown in Table 4.6, and for different number of machines in Table 4.7. The tables show the average relative deviation (*GAP*), the standard deviation (*SD*) and the average computational time in seconds (*CPU*) for each set of instance. The *ILS* methods

Instances	NEH-sep-ILS gap- sd -time(s)	John-av-ILS gap -sd -time(s)
$D10 \times 03 \times 05$	11.7 -0.05 - 3.02	7.48 - 0.03 -3.44
$D10 \times 03 \times 07$	6.86 -0.04 - 3.86	5.34 - 0.03 -4.24
$D10 \times 03 \times 11$	9.31 -0.03 - 4.20	8.71 - 0.03 -4.34
$D10 \times 05 \times 07$	8.3 -0.03 - 5.28	4.49 - 0.03 -5.13
$D10 \times 05 \times 11$	10.7 -0.03 - 4.97	8.03 - 0.05 -5.02
$D20 \times 03 \times 05$	8.93 -0.02 - 5.44	6.80 - 0.02 -5.19
$D20 \times 03 \times 07$	7.99 -0.03 - 5.60	7.16 - 0.03 -5.22
$D20 \times 03 \times 11$	7.18 -0.02 - 5.91	7.57 - 0.02 -5.83
$D20 \times 05 \times 07$	9.75 -0.05 - 6.12	7.06 - 0.04 -6.74
$D20 \times 05 \times 11$	7.55 -0.02 - 6.73	7.23 - 0.04 -6.47
$D50 \times 03 \times 05$	9.33 -0.03 - 13.27	7.52 -0.03 -10.52
Average	8.87% -0.03 -5.84	7.03% -0.03 -5.06

Table 4.7 Gap, sd and time (s) of Johnson' s and the *NEH* adaptations for different number of machines (D)

achieved better results than the methods NEH_{sep} and $John_{av}$. It is also observed that for both environments with equal number of machines in the semi-lines and with different number of machines in each semi-line, the method that achieved a smaller average deviation in relation to optimal results was the one that used $John_{av}$ with *ILS* $(6, 15, 7.03)$.

4.5.2 Results generated by simulated annealing

For even better results, the simulated annealing with iterated local search method was chosen to refine the results and build a better quality solution. This method has two types of initial solutions as its starting point that is generated similarly to the previous methods. The first method considers Johnson's algorithm with the average processing time of the semi-line as an initial solution ($John_{av}$) and the second considers the NEH_{sep} algorithm. For this method, the selection of the parameters is described below. The type of neighborhood that used this method has already been presented in chapter 3 in section 3.5.1 where a neighboring sequence is achieved basically from the current sequence by removing a task from its position and inserting it into another position.

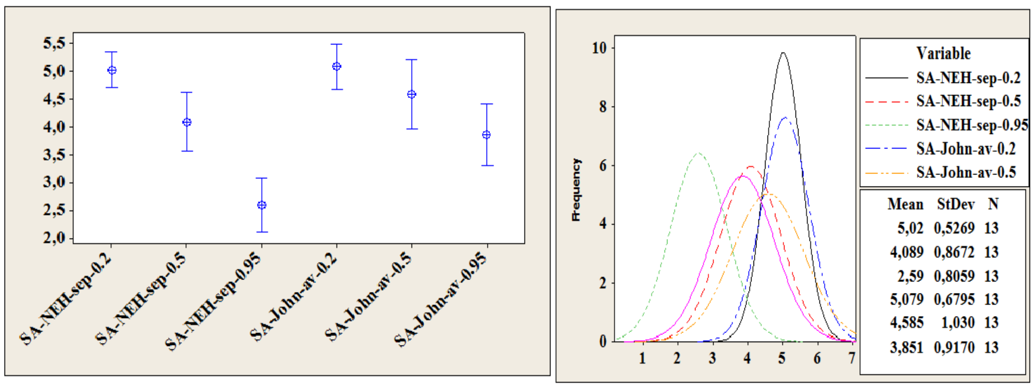
For this study the *GAP* is considered as the variation between the optimal solution or the best solution found by the mathematical method and the solutions of the developed methods: simulated annealing with initial solutions with the NEH heuristic and Johnson's rule considering the same number of machines and different number of machines in each semi-line. Table 4.8 presents the average relative deviation (GAP) for each method of the simulated annealing with local search, average standard deviation (SD), the percentage of optimal values (OPT) and average computational time in seconds (CPU) for environment with same (E) and different number of machines (D). The used parameters were based in

Method	α	gap	sd	opt	time
		%	%	%	sec.
$E-John_{av}-SA-ILS$	0.2	5.07	6.52	18.4	53.17
$E-John_{av}-SA-ILS$	0.5	4.58	9.89	17.6	50.81
$E-John_{av}-SA-ILS$	0.95	3.85	8.81	20.0	51.32
$E-NEH_{sep}-SA-ILS$	0.2	5.02	5.06	19.2	51.32
$E-NEH_{sep}-SA-ILS$	0.5	4.08	8.33	20.0	53.12
$E-NEH_{sep}-SA-ILS$	0.95	2.59	7.74	17.6	54.10
$D-John_{av}-SA-ILS$	0.2	5.61	2.75	18.4	55.82
$D-John_{av}-SA-ILS$	0.5	4.56	2.52	25.2	56.62
$D-John_{av}-SA-ILS$	0.95	3.45	2.78	19.2	50.72
$D-NEH_{sep}-SA-ILS$	0.2	5.48	2.81	32.2	52.22
$D-NEH_{sep}-SA-ILS$	0.5	4.52	2.89	28.1	49.32
$D-NEH_{sep}-SA-ILS$	0.95	3.96	2.91	28.2	50.42

Table 4.8 Results of Gap, sd, opt and time for Simulated Annealing methods

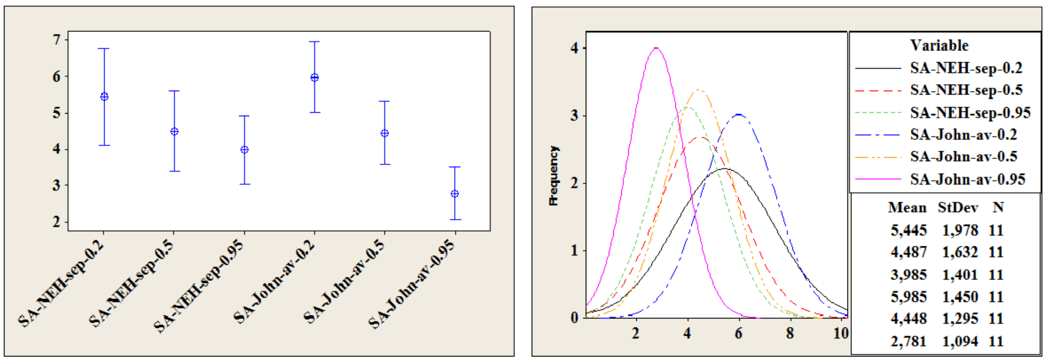
the studies by Senthilkumar and Narayanan [94] and Salem [5] with some adjustments for the characteristics of this problem. The parameters that yielded significant results were those with initial temperature $Ti = 6000$. The cooling factor value used in the algorithm affects the execution time and the quality desired for the solution. In this study, the cooling factor $\alpha = 0.2, 0.5$ and 0.95 and the final temperature $= 0.0001$ were analysed. For each scenario (semi-lines with same number of machines and with different number of machines), Figure 4.7 presents the results of using variants of the simulated annealing method for the problem in question. $NEH_{sep}-SA-ILS$ (simulated annealing starting with NEH_{sep} algorithm) and $John_{av}-SA-ILS$ (simulated annealing starting with $John_{av}$ and ending with local search) methods presented better results then those previously used in the same set of instances in two environments.

For the environment with the same number of machines in the parallel semi-lines, the best average relative deviation (GAP) was 2.59 in $NEH_{sep}-SA-ILS$ method with $\alpha =$ 0.95. The worst GAP was 5.07 in $John_{av}-SA-ILS$ method with $\alpha = 0.2$. For the environment with different numbers of machines, the best GAP was 3.45 in $Jonh_{av}-SA-ILS$ method for $\alpha = 0.95$. The worst GAP was 5.61 in $John_{av}-SA-ILS$ method for $\alpha = 0.2$. These results also showed GAP improvement when compared with the results of the previous method.



Mean	StDev	N
5,02	0,5269	13
4,089	0,8672	13
2,59	0,8059	13
5,079	0,6795	13
4,585	1,030	13
3,851	0,9170	13

a- Same number of machine –SA



Mean	StDev	N
5,445	1,978	11
4,487	1,632	11
3,985	1,401	11
5,985	1,450	11
4,448	1,295	11
2,781	1,094	11

b- Different number of machine -SA

Fig. 4.7 Gap and sd for Simulated Annealing methods

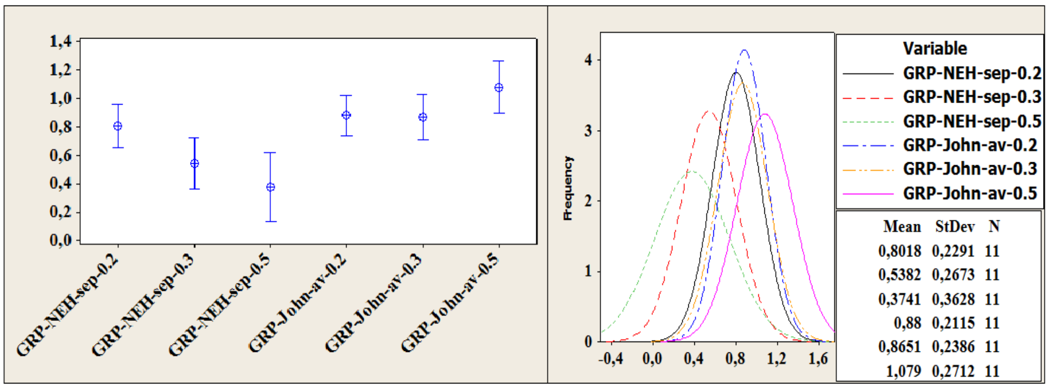
4.5.3 Results generated by GRASP

The last proposed method was an adaptation of the original *GRASP* algorithm with the principles of the NEH_{sep} algorithm or $John_{av}$ algorithm. This method, as described in section 3.5.4, has two phases. The first phase is the construction of the initial solution, and the second one is the improvement of this solution (local search). In this study, the construction phase of the initial solution first forms the candidate list (*LC*) with the sequence of the jobs in descending order of the processing time, when the NEH_{sep} is used and in ascending order when Johnson' s algorithm is used. After this step, the restricted list is formed by the draw of $\alpha\%$ of the candidate list, which makes each initial solution different from the other. The values of α for the algorithm were defined as $\alpha = 0.2, 0.3$ and 0.5% of the number of jobs in the candidate list. The sequence is formed by allocating the jobs using the NEH_{sep} or $John_{av}$. Upon finishing the initial solution construction process, *GRASP* second phase starts as described in section 3.5.4. The algorithm finishes when there are 100 iterations without improvement of the solution.

Method	α	gap	sd	opt	time
		%	%	%	sec.
$E-John_{av}-GRP-ILS$	0.2	0.91	0.33	32.0	49.82
$E-John_{av}-GRP-ILS$	0.3	0.93	0.37	50.4	44.81
$E-John_{av}-GRP-ILS$	0.5	1.07	0.28	32.1	58.23
$E-NEH_{sep}-GRP-ILS$	0.2	0.65	0.22	40.0	50.32
$E-NEH_{sep}-GRP-ILS$	0.3	0.53	0.17	54.4	45.12
$E-NEH_{sep}-GRP-ILS$	0.5	0.35	0.15	63.0	44.45
$D-John_{av}-GRP-ILS$	0.2	0.88	0.21	32.1	44.31
$D-John_{av}-GRP-ILS$	0.3	0.86	0.23	39.6	48.58
$D-John_{av}-GRP-ILS$	0.5	1.08	0.27	42.1	40.12
$D-NEH_{sep}-GRP-ILS$	0.2	0.80	0.22	44.6	42.52
$D-NEH_{sep}-GRP-ILS$	0.3	0.53	0.26	48.8	51.32
$D-NEH_{sep}-GRP-ILS$	0.5	0.37	0.33	52.8	49.42

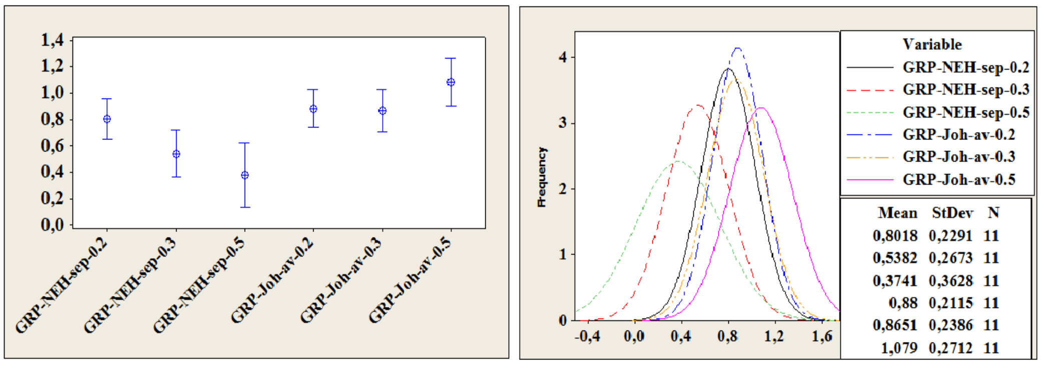
Table 4.9 Results of *gap*, *sd*, *opt* and time for GRASP methods

Table 4.10 and Figure 4.8 aid at the observation of the *GAP* behaviors of the $NEH_{sep}-GRP-ILS$ and $John_{av}-GRP-ILS$ *GAP* methods. Theses methods performed better than all others previously used. It is also observed that this method achieved larger number of optimal solution in relation to all other methods. $NEH_{sep}-GRP-ILS$ method with $\alpha = 0.5$ achieved the best average *GAP* in the two types of environments 0.35 and 0.37 respectively in relation to the environments with the same number of machines and different number of machines in the semi-lines. $John_{av}-GRP-ILS$ with α =0.5 method was the GRASP variant that achieved the highest *GAP* for this method (1.07 and 1.08).



Mean	StDev	N
0,8018	0,2291	11
0,5382	0,2673	11
0,3741	0,3628	11
0,88	0,2115	11
0,8651	0,2386	11
1,079	0,2712	11

a- Same number of machine



Mean	StDev	N
0,8018	0,2291	11
0,5382	0,2673	11
0,3741	0,3628	11
0,88	0,2115	11
0,8651	0,2386	11
1,079	0,2712	11

b- Different number of machine

Fig. 4.8 Gap and sd for GRASP methods with the same and different number of machines

4.6 Comparison of results

The results for the flow shop problem with parallel semi-lines and synchronization machine at the end of the semi-lines were evaluated through the average relative absolute deviation (*GAP*) in relation to the optimal solution or the best solution found by the mathematical model. In this study, linear programming methods, Johnson's algorithm, the *NEH* algorithm, local search (*ILS*), simulated annealing (SA), Greedy Randomized Adaptive Search Procedure (*GRASP*), and the combination of these methods were used. The results and performance of these methods were presented in this chapter. First of all, the linear programming model was implemented and analyzed in 240 instances. The first 130 instances were constructed for environments with same number of machines in each semi-line and the other 110 instances were constructed for environment with different numbers of machines in the semi-lines. The results are presented in Table 4.10.

	IP model		NEH-SA-ILS				NEH-GRASP			
Instances	opt	time(s)	opt	gap(%)	sd	time(s)	opt	gap(%)	sd	time(s)
$E10 \times 03$	10	0.3	3	2.01	0.72	38.9	6	0.97	0.22	27.2
$E10 \times 05$	10	0.5	4	2.86	0.89	38.3	6	0.12	0.14	34.2
$E10 \times 07$	10	1.9	4	3.29	0.77	50.2	6	0.12	0.13	48.5
$E10 \times 11$	10	10.7	2	1.04	0.76	51.3	4	0.26	0.13	46.3
$E20 \times 03$	10	0.3	4	2.03	0.72	49.9	6	0.95	0.16	43.5
$E20 \times 05$	10	405.0	0	2.42	0.66	50.6	5	0.12	0.15	45.2
$E20 \times 07$	10	2289.0	0	2.12	0.86	50.5	5	0.19	0.09	43.9
$E20 \times 11$	4	7541.8	1	2.07	0.76	55.1	4	0.30	0.15	43.2
$E50 \times 03$	10	5.8	3	3.21	0.78	52.4	5	0.19	0.16	46.2
$E50 \times 05$	8	4915.7	2	3.12	0.78	53.2	4	0.16	0.14	47.5
$E50 \times 07$	1	13695.5	0	4.32	0.74	65.2	0	0.20	0.13	57.4
$E50 \times 11$	0	12746.2	0	2.59	0.75	74.2	0	0.13	0.14	57.4
$E100 \times 03$	10	160.69	3	2.62	0.82	88.2	4	0.13	0.22	57.4

Table 4.10 SA-ILS-NEH-0.95 / GRASP-NEH-0.5

Optimum solutions were found in 79.2% of the instances related to the same number of machines in the semi-lines, and 87.2% of instances related to the environments with different numbers of machines in the semi-lines. From these tests, it was possible to observe that it is unfeasible to solve this problem by the mathematical model in larger instances. Therefore, other alternatives were developed to find out good solutions for larger instances. For this, the *NEH* heuristic and Johnson's algorithm were applied considering two variants of Johnson's algorithm ($John_{hi}$ and $John_{av}$) and three variants of the *NEH* algorithm (NEH_{hi}, NEH_{av} and NEH_{sep}).

The results indicated that by solving this problem through variants of the *NEH* heuristic and Johnson's algorithm it was possible to find out which of the variants presented a lower relative deviation in relation to the optimal result using the *GAP* calculation. In this case, they were the NEH_{sep} and $John_{av}$ algorithms. In relation to the environment with the same number of machines, the NEH_{sep} variant showed better average relative deviation (10%). For the environment with different number of machines, $John_{av}$ and the NEH_{sep} variants showed the best results (14% and 15%). However, these results can be improved in order to be near the optimal solution.

Other solutions were found by metaheuristics using the same set of 240 instances. The methods of iterated local search presented better results than the computational results presented by Johnson's and the *NEH* algorithms. The method was implemented using the NEH_{sep} and $John_{av}$ algorithms as the initial solution. It can be verified that for both environments with the same number of machines in the parallel semi-lines and for environments with different numbers of machines in the parallel semi-lines, the iterated local search method with initial solution formation by $John_{av}$ performed better. For the environment with the same number of machines, the best average relative deviation was 6.15 for $John_{av} - ILS$ and the worst average deviation was 6.78 for the $NEH_{sep} - ILS$. For the environment with different machines numbers, the deviation for $John_{av} - ILS$ was 7.03 and for the $NEH_{sep} - ILS$ was 8.87. These results show improvements in the relative deviations in relation to the optimum if they are compared to the results of the methods previously used. However, these results can be improved to a better approximation to the optimal solutions.

	IP model		John$_{av} - SA - ILS$				NEH$_{sep}$$GRASP$			
Instances	opt	time(s)	opt	gap(%)	sd	time(s)	opt	gap(%)	sd	time(s)
$D10 \times 03 \times 05$	10	0.1	3	4.32	2.84	42.5	6	0.07	0.4	27.0
$D10 \times 03 \times 07$	10	0.2	4	4.21	1.95	46.6	6	0.65	0.4	33.4
$D10 \times 03 \times 11$	10	0.9	2	3.22	3.86	50.2	6	0.84	0.4	47.8
$D10 \times 05 \times 07$	10	0.3	3	3.35	3.21	55.4	5	0.69	0.3	45.7
$D10 \times 05 \times 11$	10	1.4	3	4.03	1.86	48.7	5	0.01	0.3	44.0
$D20 \times 03 \times 05$	10	4.1	4	3.02	2.81	49.6	5	0.65	0.2	47.8
$D20 \times 03 \times 07$	10	52.9	3	2.32	2.84	50.7	4	0.89	0.3	43.2
$D20 \times 03 \times 11$	4	14250.2	1	3.11	2.54	52.8	0	0.09	0.3	50.7
$D20 \times 05 \times 07$	10	200.1	0	4.03	3.25	48.0	3	0.12	0.2	48.1
$D20 \times 05 \times 11$	2	18208.3	0	4.25	3.11	50.2	0	0.01	0.3	51.8
$D50 \times 03 \times 07$	10	153.8	2	2.15	2.41	48.6	4	0.08	0.3	57.4

Table 4.11 John$_{av} - SA - ILS - 0.95/NEH_{sep} - GRASP - 0.5$

Simulated annealing with the local search-*ILS* (see Table 4.11) was another used metaheuristic. For this method, three different cooling factors ($\alpha = 0.2, 0.5$ and 0.95) were used for the two types of studied environments. For the environment with the same number of machines in the parallel semi-lines, the best average relative deviation was 2.59 for the $NEH_{sep} - SA - ILS$ method with $\alpha = 0.95$, the worst average relative deviation was 5.07 in

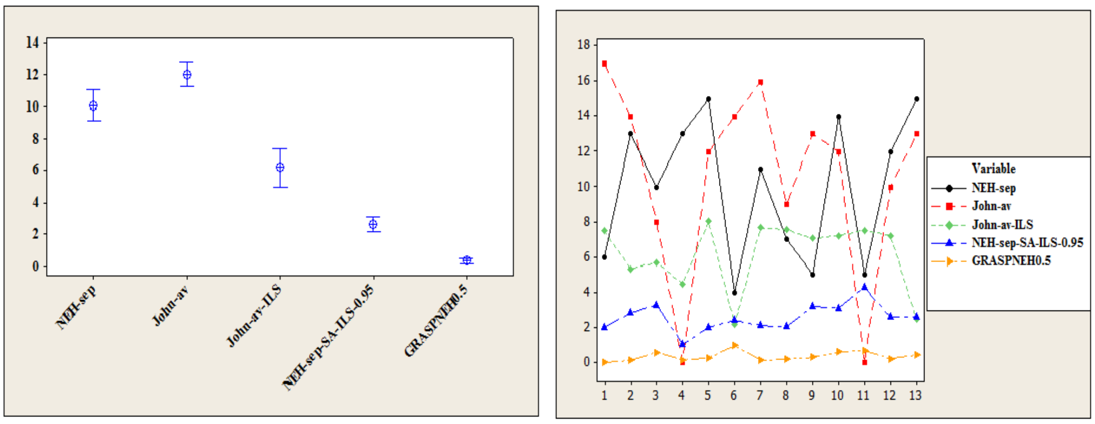


Fig. 4.9 Comparative graph of the best results of the resolution methods for the same number of machine environments in the semi-lines

the method $John_{av}-SA-ILS$ with $\alpha = 0.2$. For the environment with different numbers of machines, the best average relative deviation for $John_{av}-SA-ILS$ method was 3.96 with $\alpha = 0.95$ and for the worst average relative deviation was 5.61 for $Jonh_{av}-SA-ILS$ method with $\alpha = 0.2$. These results also show improvements in the average relative deviation if they are compared to the results of the methods previously used. The *GRASP* method used in this study follows the same steps of the original *GRASP*. In the construction phase, the restrict list is generated by selecting α % from the candidate list. In this study, three values of α $(0.2, 0.3$ and $0.5)$ were evaluated. The results of each α value were the best in relation to all the approached heuristic and metaheuristic methods. For the environment with the same number of machines, the best average relative deviation was 0.35 for the $NEH_{sep}-GRASP$ with $\alpha = 0.5$. For the environment with different numbers of machine in each parallel semi-line, the best average relative deviation for the $NEH_{sep}-GRASP$ was 0.37 with $\alpha = 0.5$. and for the worst average relative deviation for $John_{av}-GRASP$ was 1.07 with $\alpha = 0.2$. Tables 4.10 and 4.11 show the comparison of the best results of the proposed algorithm with respect to the two methods indicated for the initial solution. In the first column of these tables, the sets of instances are displayed. In the second and third columns, the percentage of instances that achieved the optimal values (OPT) and the average computational time of the mathematical model in seconds (CPU) are shown. In the following

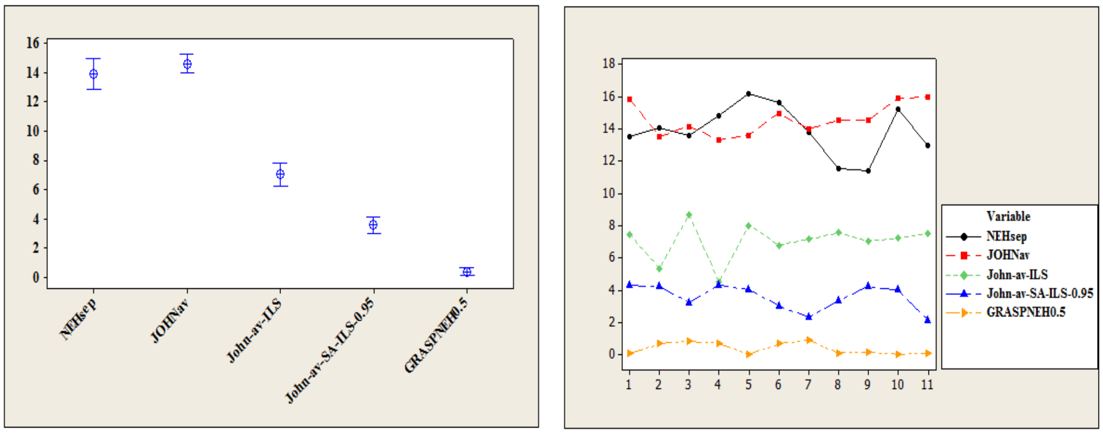


Fig. 4.10 Comparative graph of the best results of the resolution methods for different number of machines in the semi-lines

columns, results of the $SA-ILS$ and $GRASP$ methods $GAPs$, the standard deviation SD and the average computational time of the best are displayed.

The comparative graphs of the best results with the elaborated methods are presented in Figures 4.9 and 4.10. In these graphs, it is possible to visualize the behavior of each method in relation to the average relative deviation and the difference of the $GAPs$ between the methods. In this study, it was intended to make a first approach through heuristics and metaheuristics to the flow shop scheduling problem with parallel semi-lines and synchronization machines at the end considering an environment with the same number of machines in each semi-line and another environment with different number of machines in each semi-lines. However, other methods can be developed in order to achieve better solutions to the discussed problem. The methods elaborated in this study are shown in Figure 4.11.

4.7 Conclusion

In this chapter, computational experiments for solving the flow shop scheduling with parallel semi-lines and final synchronization operation were exhibited. It reported how the test problems were generated, as well as it presented analyses and it compared the achieved results through mathematical modeling and through the developed heuristic methodologies. In a first approach, a mixed integer linear programming model was defined. Due to the

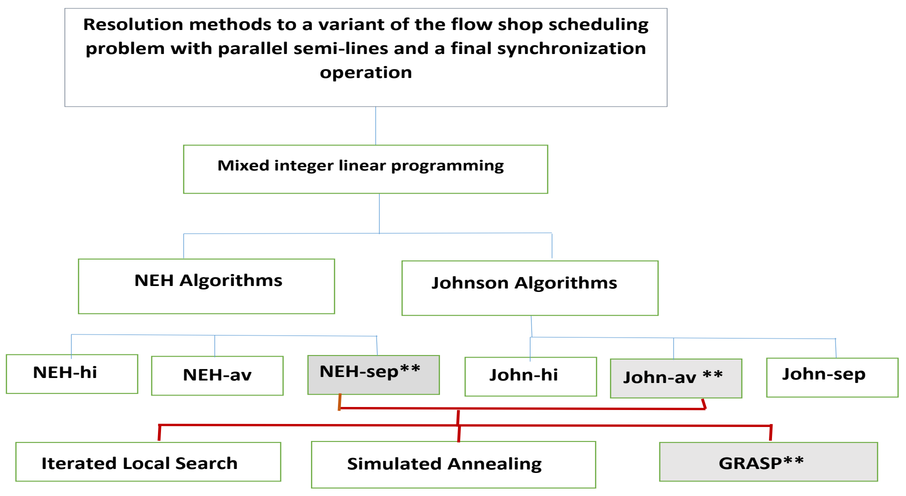

Resolution methods to a variant of the flow shop scheduling problem with parallel semi-lines and a final synchronization operation
Mixed integer linear programming
NEH Algorithms
Johnson Algorithms
NEH-hi
NEH-av
NEH-sep**
John-hi
John-av **
John-sep
Iterated Local Search
Simulated Annealing
GRASP**

Fig. 4.11 Optimization methods used

complexity of the problem, it was also solved by Johnson' s rule, the *NEH* heuristics and by the Iterated Local Search-*ILS*, Simulated Annealing-*SA* and *GRASP* metaheuristics. Finally, an extensive computational experiment was performed and comparisons of the proposed methods were made. The method that used the NEH_{sep} algorithm with *GRASP* metaheuristic presented better quality than the other methods.

Chapter 5

Conclusion

In this chapter, the conclusion of the study and the potentiality of the proposed methods to solve the variant of the flow shop scheduling problem with parallel semi-lines and a final synchronization operation are presented. Some suggestions for future researchings are given.

5.1 Conclusion

This work focused on the development of optimization methods to solve a variant of the flow shop scheduling problem with parallel semi-lines and a final synchronization operation. This is a common problem in many industrial sector that requires efficient methods for its resolution. Thus, this studied emerged from a practical situation in a welding process of an industry that manufactures electrical-electronic system products.

This study aimed to model the flow shop scheduling problem in parallel semi-lines with a final synchronization operation at the end of the semi-lines to solve it with heuristics and metaheuristics. In a first approach, a mixed integer linear programming model was defined based on studies developed for environments similar to the ones already studied. The model required adaptations regarding to the treatment of the semi-lines separately and a particular attention in the finalization of the semi-lines, since this is the point of junction of the items that are produced in the two semi-lines. The model was efficient for smaller instances, but for larger instances it presented some difficulties to find the optimal solutions in a viable computational time. Due to the complexity of the problem, it was also solved by heuristics and metaheuristics. From the computational results it was possible to prove that the constructions of methods using heuristics and metaheuristics yielded viable results. The presented algorithms, mainly the methods combination, have good options for resolutions of larger instances.

The resolution of the problem through methods such as Johnson's rule, heuristic methods as the *NEH* algorithm and the metaheuristics such as *ILS*, simulated annealing and *GRASP* show how this methods behaved in this environment and the possibility of finding optimal result using them. In addition, among the methods, the NEH_{hi}, the NEH_{av}, the NEH_{sep}, $John_{av}$, $John_{hi}$, the $NEH_{sep}-ILS$, $Joh_{av}-ILS$, the $NEH_{sep}-SA-ILS$, $John_{av}-SA-ILS$, the $NEH_{sep}-GRASP$, $John_{av}-GRASP$, the $NEH_{sep}-GRASP$ with 0.5% was the method that achieved the most amount of optimal results and the lowest *GAP* in the instances. The lowest average GAP was 0.35% for environments with the same number of machines in the semi-lines and 0.37% for environments with different number of machines in each semi-line. On the other hand, the highest value of the average *GAP* generated by the best presented algorithm ($GRASP-NEH$) was 1.07%.

In relation to the used heuristic methods, the NEH_{hi} with average *GAP* of 16.0 % was the worst proposed method for environments with same number of machines in each semi-line and for environments with different number of machines in each semi-line, $John_{hi}$ with average *GAP* of 19%. Among the methods using metaheuristic, the hybrid method $NEH_{sep}-ILS$ presented the worst result with *GAP* of 6.78 % for environments with the same machines in the semi-lines and *GAP* of 8.87% for environments with different machines in the semi-lines. The achieved results by the simulated annealing methods applied to the different instances of the problem presented less satisfactory results compared to the *GRASP* methods. However, it was possible to visualize that as the cooling factor (α) increased the results improved, that is, when the cooling factor approaches the value 1, the number of the explored solutions in the neighborhood increases, thus leading to a large number of solutions researched. In this sense, the simulated annealing may also be a good ally to other methods to solve this problem, since such method has the advantage of exploiting larger quantities of neighboring solutions with the possibility of escaping from the local optimal.

Finally, the results of the experiments indicate that the methods that used GRASP represent a good alternative to solve the problem, especially for larger instances. This fact increases the viability of using these methods as a means of obtaining better solutions or reaching higher rate at optimum values and application in real problems in viable computational time.

5.2 Suggestion for future studies

Further directions of researching involve more complex problems and multi-objective functions, considering the same environment, since there is a new tendency to use approximate methods to solve this class of problems. Such an approach, while not guaranteeing a good solution to the problem, is able to provide good quality solutions in acceptable processing time, which is perfectly considered in many real situations. Therefore, the development of algorithm with other metaheuristics on this problem (genetic algorithm, ant colony optimization, Particle Swarm optimization and Tabu Search, for example) can be subject matters to be considered in other researchings.

Tuning the algorithms with packages as Irace [55] is another suggestion for improvement of the developed algorithms.

To improve the results generated by the GRASP methods, it is suggested a study of the use of techniques such as Path Relinking, including a constructive procedure, a post-processing phase in which the best obtained solution would be a combination of Parh Relinking procedure and some movements of improvement. This step would have the purpose of obtaining improvements at the quality the solution and in the execution time. The incorporation of new restrictions to the models are also pointed out, such as the inclusion of the setup time and the intermediate stock between the semi-lines and the synchronization machine with the objective of minimizing the sum of tardiness, earliness (finished goods inventory holding), and intermediate (work-in-process) inventory holding costs.

The development of new instances to the flow shop scheduling with real situation is another issue to be considered in future studies, since they are the first tests performed for this variant of the flow shop scheduling and still without reference to compare the parameters pertinent to this problem.

Chapter 6

French Resume

Ordonnancement dans un atelier de type flow shop semi-parallel avec operation de synchronisation

UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

utt
UNIVERSITÉ DE TECHNOLOGIE TROYES

Résumé

Cette étude est une variante du problème flow shop motivée par une situation pratique. Dans ce contexte, il y a une ligne d'assemblage composée de deux demi-ligne parallèles avec des activités indépendantes. La premiére demi-ligne a un nombre de machines q_1 et le second demi-ligne a un nombre de machines q_2 . A la fin des deux demi-ligne il y a une machine responsable par l'union des produits des deux demi-lignes. Chaque demi-ligne est dédiée à différentes tâches, en raison chaque travail nécessite des opérations dans chacune des machines des demi-ligne avec temps de transformation différents. Les tâches d'une demi-ligne ne dépendent pas de la réalisation d'une autre tâche dans autre demi-ligne, de sorte qu'une tâche est traitée en parallèle dans chaque demi-ligne. La séquence des travaux dans chaque demi-ligne paralléle devrait être la même, bien qu'une tâche d'une demi-ligne n'ait pas de besoin de commencer en même temps a une autre demi-ligne. La dernière operation de synchronisation ne peut pas être démarrée que lorsque les opérations dans les deux demi-lignes aient ont été complétées. La solution à ce problème est de déterminer une séquence de travail possible pour optimiser le makespan. En ce sens, l'objectif de cette étude est de modéliser le problème flow shop dans l'environnement de production propose et les résoudre avec des algorithmes spécialisés. Une première approche, nous avons défini un modelé de programmation linéaire mixte en nombres entiers et au vu des la complexité du problème, il a également été résolu par la règle Johnson, heuristiques *NEH* et la recherche locale Meta-heuristiques Iterated, Recuit Simule et *GRASP*. Enfin, une importante campagne de tests a été menée et une comparaison des méthodes proposées a été réalisée. Le méthode hybride qui utilise l'algorithme *NEH* avec le *GRASP* a démontré sa supériorité par rapport autres méthodes proposées.

mots-clés: Ordonnancement (Gestion), Ordonnancement (Informatique), Heuristique, Métaheuristiques, Synchronisation

6.1 Introduction Générale

L'ordonnancement dans le système de production est l'une des principales activités et il est considéré par certains chercheurs comme un domaine vaste et diversifié, voir, par exemple, Pinedo [77]. Nous nous sommes intéressés à l'ordonnancement de la production dans une variante de l'environnement flow shop de permutation.

Le problème d'ordonnancement dans le système de production classique d'ateliers flow shop de permutation se compose de *n* tâches à traiter parmi un ensemble de *m* machines disposées en série. Les tâches doivent suivre le même ordre technologique vers les diverses machines, et chaque tâche a un temps de traitement dans chaque machine spécifique. L'objectif est de déterminer, parmi toutes les séquences possibles, celui qui permet d'optimiser une certaine mesure de performance.

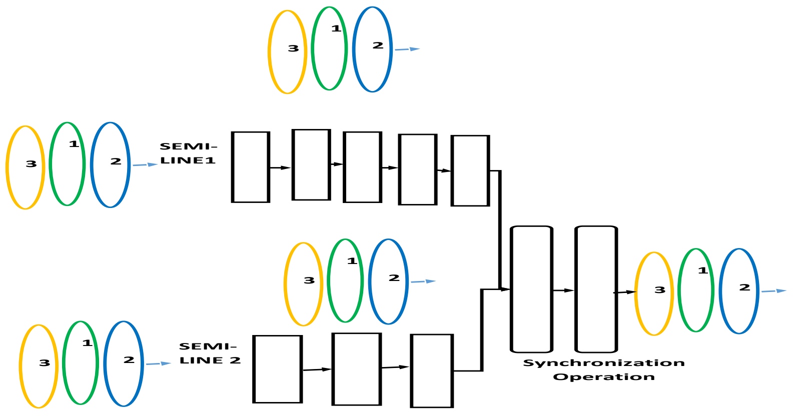


Fig. 6.1 Système étudié

Dans cette étude, une variante du problème d'ateliers flow shop de permutation est analysée, motivée par une application pratique dans le secteur d'activité de soudage d'une industrie électronique. L'environnement d'atelier est constitué de deux demi-lignes parallèles avec des activités indépendantes et d'une opération de synchronisation finale. Chaque demi-ligne produit l'une des moitiés du produit final qui est assemblé en un seul produit dans l'opération de synchronisation finale. L'ordre des moitiés des produits finaux dans chaque demi-ligne doit être égal, et il doit également être suivi dans l'opération de synchronisation finale. La Figure 6.1 montre un schéma de l'environnement étudié. Une opération dans une machine d'une demi-ligne n'a pas besoin de démarrer au même moment d'une opération dans la machine de l'autre demi-ligne.

Cependant, l'opération de synchronisation finale d'un produit ne peut être démarrée que lorsque les opérations de ses moitiés dans les deux demi-lignes sont terminées. Compte tenu de cet arrangement, il est possible de classer cet environnement comme un cas particulier du flow shop de permutation. L'objectif de cette étude est d'obtenir des solutions optimales ou presque optimales pour minimiser le makespan. Pour trouver ces solutions, une formulation mathématique, heuristique et métahéhuristique pour le problème sont proposés.

Diverses approches ont été proposées pour le problèmes d'ordonnancement d'ateliers de type flow shop puisque Johnson a présenté la résolution pour le flow shop avec deux machines. Gupta et Stafford [40] ont fait une perspective historique de la recherche dans le probléme du flow shop et de ses variantes. L'algorithme bien connu de Johnson [45] a trouvé en temps polynomial une séquence optimale pour un groupe de n travaux à traiter dans $m = 2$ machines. Une difficulté majeure est une solution optimale lorsque le nombre de machines est supérieur à deux, car on sait que ce problème est fortement NP-Hard (Garey et al. [33]). Ainsi, des études ont été développés dans la littérature d'ordonnancement d'ateliers flow shop avec techniques exactes et heuristiques.

Tseng, Stafford and Gupta [102] rapportent une analyse empirique détaillée pour évaluer l'efficacité des formulations de programmation linéaire en nombres entiers mixte (MIP) pour le flow shop de permutation. Nous donnons brièvement quelques exemples plus récents de l'utilisation des modèles MIP pour résoudre les problèmes de flow shop dans la littérature. Frach et al. [30] présentent également un MIP pour résoudre les problèmes de flow shop avec un nombre limité de stock intermédiaire. Naderi et al. [64] proposent un MIP pour minimiser le makepan et le retard total dans un environnement de flow shop. Ronconi et Bergin [86] résolvent par MIP le problème de minimiser la anticipation totale et le retard des tâches pour le problème flow shop avec un nombre illimité et zéro stock tampon. Hnaien et al. [41] proposent deux modèles MIP pour le problème de ordonnancement flow shop de deux machines avec une contrainte d'indisponibilité dans la première machine afin de minimiser le makepan. Les auteurs proposent une algorithme branche and bound basé sur de nouvelles limites inférieures et des heuristiques qui se comportent mieux que les deux modèles *MIP*.

Les heuristiques ont été proposées dans la littérature pour obtenir de bonnes solutions dans un coût computationnel moins élevés, voir, par exemple, Mainieri et Ronconi [57] Nawaz et al. [66] Rad et al. [80], et Widmer Hertz [106]. L'algorithme de Johnson a été adapté par Allahverdi et al. [1] pour minimiser le temps d'achèvement total des deux machines en flow shop avec des temps de traitement aléatoires et limitées. Pan et al. [74] a abordé le problème de flow shop avec stock intermédiaire égal à zéro. Dans l'étude, l'heuristique de Nawaz et al. [66] ont exploré des caractéristiques spécifiques du problème pour trouver de bonnes solutions avec un coût computationnel moins élevés. Allaoui et Artiba [8] ont fait une étude pour minimiser le makespan en flow shop hybride de deux étage avec une seule machine dans le première étage et *m* machines dans le deuxième étage. Fernandez-Viagas et Framinan [29] ont proposé des mécanismes efficaces le critère d'arrêt à utiliser dans l'heuristique de Nawaz et al. [66] lorsqu'ils traitent avec le retard total.

Des approches méta-heuristiques ont également été proposées dans la littérature pour résoudre de grandes instances dans un coût computationnel raisonnable. Le recuit simulé a été appliqué par Low et al. [54] et par Nearchou [67] pour minimiser le makepan dans le problème de flow shop, et par Mirsanei et al. [62] et par Santosa et Rofiq [91] au problème du flow shop hybride avec les *m*-machines à chaque étape. Le *GRASP* a été appliqué par Prabhaharan et al. [79]. Shahul Hamid Khan et al. [96] a étudié un atelier de type bicriteria flow shop afin de minimiser la somme pondérée de l'accroissement et la retardée maximale par la méthode *GRASP*. Leur algorithme a été capable de surpasser le recuit simulé a proposé précédemment par Chakravarthy et Rajendran [18] pour le même problème. D'autre part, une importante campagne de tests menées dans l'étude de Sivasankaran et al. [99] a montré que l'algorithme avec recuit simulé appliqué au problème a surpassé l'algorithme avec *GRASP* pour un problème d'ordonnancement a une seule étage.

Sur la base des études mentionnés ci-dessus, cette étude présente une premier approche, pour un modele de programmation lineaire mixte en nombres entiers et au vu de la complexite du problème, il a également été resolu par la regle Johnson, heuristiques *NEH* et la recherche locale Metaheuristiques, Recuit Simule et *GRASP*. Enfin, une importante campagne de tests a été effectuée et une comparaison des méthodes proposées a été realisée. Approches de modélisation et d'optimisation

6.2 Approches de modélisation et d'optimisation

Dans cette étude, certaines approches pour résoudre le problème de ordonnancement dans un atelier de type flow shop semi-parallel avec opération de synchronisation finale ont été effectuées. Les méthodes proposées sont: un modèle de programmation linéaire entier mixte présenté dans la section 6.2.1, les méthodes qui ont utilisé l'algorithme de Johnson et les méthodes heuristiques basées sur l'algorithme *NEH* dans la section 6.2.2 et les méthodes basées sur la methaheuristics recherche locale, le recuit simulé et *GRASP* dans le section 6.2.3.

6.2.1 Formulation mathématique

La formulation *MIP* sur la base des études antérieurs développés par Wagner [104] et par Stafford [100] a été proposé. Selon l'étude menée par Teng et al.[102] ce genre de formulation est la meilleure pour le flow shop de permutation. Soit le nombre de machines ml en demi-ligne $l = 1;2$ (avant l'opération de synchronisation). Dans notre modèle, une permutation unique doit être choisie pour deux problèmes du type flow shop avec $ml+1$ machines, chacun étant soumis à la contrainte supplémentaire que les temps de fin de chaque travail sur les machines $m1+1$ et $m2+1$ sont identiques. Le modèle peut être généralisé pour un nombre arbitraire de demi-lignes L.

Ainsi, en supposant que, à des fins de modélisation, $l = 1,2$; flow shop problèmes avec $ml+1$ machines chacun, les variables et les paramètres sont les suivants: z_{ij} est une variable binaire que prend la valeur 1 si i est attribué à la j^{ime} position de la permutation (commun aux deux problèmes du flow shop L = 1, 2), et 0 dans le cas contraire; x^l_{jk} porte sur le temps d'inactivité de la machine k de problème l avant que le tâche commence dans la j^{ime} position de la permutation; y^l_{jk} est au temps d'attente de la tâche en j^{ime} position de la permutation après avoir terminé le traitement du problème l sur la machine k, en attendant la machine de problème $kl+1$ devient disponible; p^l_{ki} est dédié le temps de traitement de la tâche i sur le problème l de la machine k $(p_{m1+1i} = p_{m2+1i})$; C_{lj} est le temps dans l'achèvement du problème l de tâche dans la j^{ime} position. Le makespan est donné par le temps d'achèvement de la tâche dans le dernière position de la permutation.

$$\textbf{Minimize} \quad C^1_n \tag{6.1}$$

subject to

$$\textstyle\sum_{j=1}^{n} z_{i,j} = 1 \qquad i = 1,\dots,n \tag{6.2}$$

$$\textstyle\sum_{i=1}^{n} z_{i,j} = 1 \qquad j = 1,\dots,n \tag{6.3}$$

$$\begin{gathered}\textstyle\sum_{i=1}^{n} p^l_{r,i} z_{i,j+1} + y^l_{j+1,r} + x^l_{j+1,r} = y^l_{j,r} + \sum_{i=1}^{n} p^l_{r+1,i} z_{i,j} + x^l_{j+1,r+1} \\ l = 1,2; j = 1,\dots,n-1; r = 1,\dots,m_l\end{gathered} \tag{6.4}$$

$$\textstyle\sum_{r=1}^{k-1} \sum_{i=1}^{n} p^l_{r,i} z_{i,1} = x^l_{1,k} \qquad l = 1,2; k = 2,\dots,m_l \tag{6.5}$$

$$\textstyle\sum_{r=1}^{mv} \sum_{i=1}^{n} p^v_{r,i} z_{i,1} \leq x^l_{1,ml+1} \qquad l,v = 1,2 \tag{6.6}$$

$$y^l_{1,k} = 0 \qquad l = 1,2; k = 1,\ldots,m_l - 1 \tag{6.7}$$

$$x^l_{1,m_l+1} - (x^l_{1,m_l} + \textstyle\sum_{i=1}^n p_{m_l,i} z_{i,1}) = y^l_{1,ml} \qquad l = 1,2 \tag{6.8}$$

$$\textstyle\sum_{u=1}^j \sum_{i=1}^n p^l_{m_l+1,i} z_{i,u} + \sum_{u=1}^j x^l_{u,m_l+1} = C^l_j \quad l = 1,2; j = 1,\ldots,n \tag{6.9}$$

$$C^1_j = C^2_j \qquad l = 1,2 \tag{6.10}$$

$$z_{ij} \in \{0,1\} \qquad j = 1,\ldots,n; i = 1,\ldots,n \tag{6.11}$$

$$y^l_{j,k}, x^l_{j,k} \geq 0 \qquad j = 1,\ldots,n; l = 1,2; k = 1,\ldots,m_l + 1 \tag{6.12}$$

$$C^l_j \geq 0 \qquad l = 1,2; j = 1,\ldots,n. \tag{6.13}$$

La fonction d'objectif (6.1) minimise le makepan. Les contraintes (6.2) et (6.3) attribuent une tâche à une position exacte dans la permutation. La contrainte (6.4) assure l'égalité du temps de traitement plus les temps d'attente pour chaque paire de machines adjacentes. La contraint (6.5) calcule, à partir de la deuxième machine activée, le temps d'inactivité dans chaque machine de chaque demi line en attendant le premier tâche. La contrainte (6.6) garantit que le temps d'inactivité sur la machine de synchronisation en attente de la premier tâche est égale à la plus grande temps de transformation totale sur chaque demi-ligne. La contrainte (6.7) garantit qu'il n'y a pas de temps d'inactivité pour le tâche assigné à la première position dans chaque machine de chaque demi-ligne, mais le premier tâche peut attendez d'être traité sur la machine de synchronisation, ce qui est assuré par une contrainte (6.8). Les contraintes (6.9) et (6.10) assurent la synchronisation, car le temps d'achèvement pour chaque tâche dans chaque problème flow shop est calculé dans (6.9) et imposé pour être égal en (6.10). Les contraintes (6.11) et (6.12) imposent le domaine des variables.

6.2.2 Heuristique

Nous proposons des adaptations de la heuristique *NEH* par Nawaz et al. [66] et aussi des adaptations de l'algorithme de Johnson [45] pour obtenir des solutions réalisables pour le problème d'ordonnancement du type flow shop avec des lignes semi-parallèles et une opération de synchronisation finale.

Des adaptations de la heuristique *NEH*

La heuristique *NEH* pour le flow shop classique de permutation commence en triant les n tâches en ordre décroissant de la somme des temps de traitement sur toutes les m machines . Ensuite, un ordonnancement partiel consiste en la séquence des deux premiers tâches qui minimise le makespan. Les autres tâches, à partir du troisième, sont insérées (un à la fois) dans la position de l'ordonnancement partiel conduisant à un makepan plus court. La position relative entre les tâches déjà insérées dans l'ordonnancement partiel ne change pas. Nous développons trois adaptations de l'algorithme NEH: NEH_{av} - le temps moyen de traitement des tâches entre les machines parallèles correspondantes, NEH_{hi} - le temps de traitement plus long des tâches entre les machines parallèles correspondantes et NEH_{sep} - où chaque demi-ligne est considérée séparément, y compris la machine de synchronisation.

Les heuristiques NEH_{av} et NEH_{hi} nécessitent le même nombre de machines dans chaque demi-ligne, c'est-à-dire $m = m1 = m2$. Soit f désigne la machine de synchronisation finale. Le principe général est de réduire les deux demi-lignes à une seule ligne et d'appliquer la heuristique NEH.

Dans la heuristique NEH_{av}, pour chaque tâche $j = 1,...,n$, est calculé $(p^1_{im} + p^2_{im}/2$, où $k = 1,...,m$ est la k^{th}-machine dans chaque demi-ligne. À cet endroit, il existe un flow shop de permutation classique avec $m+1$ machines où, pour chaque tâche j, p_{kj} est le temps de traitement dans la machine $k = 1,...,m$ et p_{fj} est le temps de traitement dans la dernière machine, et la heuristique NEH est appliquée afin d'obtenir la séquence seq_{av}. Enfin, le makepan engagé par la séquence seq_{av} est calculé dans l'ensemble du système avec le temps de traitement actuel p^l_{kj} pour chaque demi-line $l = 1;2$.

La heuristique NEH_{hi} fonctionne de la même manière. Pour chaque tâche $j = 1,...,n$, nous calculons $p_{kj} = max\{p^1_{kj}, p^2_{kj}\}$, où $k = 1,...,m$ est la k^{th}-machine dans chaque demi-line, pour obtenir par la heuristique NEH une séquence seq_{hi} pour un problème du type flow shop avec $m+1$ machines. Nous calculons ensuite le makepan encouru par la séquence seq_{hi} dans l'ensemble du système avec les temps de traitement courants p^l_{kj} pour chaque demi-ligne $l = 1;2$.

Pour chaque tâche $j = 1,...,n$, nous calculons $\max\{p^1_{im}, p^2_{im}\}$, où $k = 1,...,m$ est la k^{th}-machine dans chaque demi-line, pour obtenir par la heuristique NEH une séquence seq_{hi} pour un problème du type flow shop avec $m+1$ machines. Nous calculons ensuite le makepan encouru par la séquence seq_{hi} dans l'ensemble du système avec les temps de traitement courants p^l_{kj} pour chaque demi-ligne $l = 1;2$.

Dans la heuristique NEH_{sep} nous considérons chaque demi-ligne avec la machine de synchronisation séparément. Nous appliquons, pour chaque demi-ligne $l = 1;2$, la heuristique NEH à un problème classique du type flow shop de permutationp avec $ml+1$ machines où, pour chaque tâche j, p^l_{kj} est le temps de traitement sur la machine $k = 1,...,ml$ et p_{fj}. Nous avons adopté la séquence seq^l_{sep}, $l = 1;2$, ce qui conduit à plus bas valeur du makespan dans l'ensemble du système avec les deux demi-lignes.

Des adaptations de algorithme de Johnson

L'algorithme de Johnson obtient une séquence qui minimise le makepan pour le problème du type flow shop avec deux machines, Soit p_{1j} (resp. p_{2j}) le temps de traitement de la tâche j dans la première machine (resp. seconde). La sèquence optimale commence par les tâches de sorte que $p_{1j} \leq p_{2j}$ a trié dans un ordre de temps de traitement non décroissant, c'est-à-dire la partie ascendante de la séquence. Dans cette étude, deux adaptations de l'algorithme Johnson 's ont été développées: algorithme de Johnson, compte tenu du temps de traitement le plus long et de l'algorithme de Johnson, compte tenu du temps de traitement moyen. Le principe général est de considérer le système étudié comme un flow shop avec deux machines en configurant l'opération de synchronisation comme deuxième machine.

Dans l'adaptation dénoté par $John_{av}$, pour chaque tâche $j = \{1,...,n\}$, le temps de traitement moyen est calculé avec: $p_j = \frac{\sum_{k=1}^{m1} P^1_{kj} + \sum_{k=1}^{m2} P^2_{kj}}{m1+m2}$. Le temps de traitement p_{fj} de l'opération de synchronisation est le temps de traitement dans la deuxième machine. Ensuite, l'algorithme de Johnson est appliqué et calculé pour la séquence obtenue. Le makepan est calculé dans l'ensemble du système avec le temps de traitement actuelle p^l_{kj} pour chaque demi-ligne $l = 1;2$.

L'adaptation dénoté par $Jonh_{hi}$ est développé d'une manière similaire, mais le temps de traitement le plus long $max_{l=1;2}; k = \{1,...,ml\}$ p^l_{kj} parmi les machines des demi-lignes, on utilise comme temps de traitement de la tâche j sur la premire machine.

6.2.3 Metaheuristique

Trois algorithmes différents avec des métaheuristiques ont été appliqués dans cette étude. La solution initiale, les méthodes de perturbation, la recherche locale et le critère d'acceptation ont été développées pour chacune d'elles.

Recherche locale

```
Procedure Local  Search (s,n )
$Seq_0 \leftarrow$ best Construction sequence;
$C(s) \leftarrow$ best Construction makespan;
$OK \leftarrow 1$
    while $OK == 1$ do
    $OK \leftarrow 0$
    $Best \leftarrow \infty$
        for1 $(i = 0$ to $n-1)$do
            for2 $(j = i$ to $n)$do
            $Seq' \leftarrow Seq +$ exchange of the position
            element $i$ by position element $j$;
            Calculate the cost $C(seq')$
            if $(C(seq') <$ Best$)$ them
                Best $\leftarrow C(seq')$
                seqviz $\leftarrow seq'$
            end-if
            end-for2
        end-for1
    if (Best $< C(seq')$) them
        $seq \leftarrow seqviz$
        $Ok \leftarrow 1$
    end-if
    end-while
    return ( $seq$)
    return ( $Best$)
end-procedure
```

Fig. 6.2 Recherche locale Pseudo-code

Dans la recherche locale (*LS*), l'espace de recherche est exploré afin d'améliorer la fonction objective. S'il y a des possibilités d'amélioration, l'échange pour générer le meilleur résultat est adopté. Dans cette étude, l'algorithme commence par la génération d'une solution réalisable (S_0) générée par deux méthodes différentes (par l'algorithme *NEH* en considérant les demi-lignes séparées et par l'adaptation de Johnson en considérant le temps de traitement moyen des demi-lines). Le processus de recherche local part de cette solution. Cette procédure est une adaptation de l'étude par Ruiz and St*ü*tzle [88] et Dong et al. [26] ce qui consiste à prendre une tâche à partir d'une position d'origine et à l'insérer dans les autres positions de la séquence et à calculer la fonction objective de chaque séquence modifiée. La solution voisine qui génère le meilleur makepan est sélectionnée et devient la nouvelle solution actuelle (S'). Ce processus se poursuit jusqu'à l'amélioration de la solution actuelle. Le processus est interrompu s'il existe un certain nombre d'itérations sans amélioration. Le pseudocode est indiquée sur le Figure 6.2.

Recuit simulé

Le recuit simulé était une adaptation de la méthode par Hurkała [42] et pseudo-code de la méthode est décrite dans le Figure 6.3. Dans cet algorithme, deux méthodes de solution initiales ont été considérées. La première méthode génère la population initiale avec l'algorithme *NEH* séparément et la deuxième méthode génère la population initiale par l'algorithme de Johnson avec la méthode du temps de traitement moyen. L'algorithme produit des perturbations dans la séquence des tâches et fournit une autre solution voisine (S'). Une nouvelle modification est faite dans chaque itération et une nouvelle solution est générée. Les solutions générées sont soumises à un processus d'évaluation afin de vérifier si l'objectif de minimiser le makepan a été atteint. Le makespan est calculé et évalué. La solution (S) est mise à jour si la solution est meilleure que les autres solutions obtenues jusque là. Cependant, l'algorithme tend à se coincer dans un optimal local. Par conséquent, seules les meilleures solutions sont acceptées. Pour éviter cela, l'algorithme permet des solutions d'aggravation. Cela se fait en respectant une probabilité d'allocation par rapport à la température. La fonction est calculée comme indiqué dans Jarosław et al. [43] qui cherche à faire un random et permet de choisir une valeur entre 0 et 1. Sinon, elle n'est acceptée qu'avec une probabilité égale à $e^{-\triangle/T}$, Où T est la température et $\triangle$ est la différence entre la solution et la solution précédemment adoptée. Le critère d'arrêt de l'algorithme est déterminé par le refroidissement lent de la température initiale. La procédure est répétée plusieurs fois par température. L'algorithme commence la recherche locale dans la séquence qui a généré la meilleure solution lors de la température.

Greedy Randomized Adaptive Search Procedure - GRASP

L'algorithme *GRASP* pour ce problème se compose de deux phases: la phase de construction de la solution initiale et la phase de recherche locale ou l'amélioration de la solution. La

```
Procedure Simulated Annealing
1  Select an initial temperature $T \leftarrow T_0$;
2  Select an initial solution $S \leftarrow S_0$;
3  $it \leftarrow 0$;
4  $best_{it} \leftarrow 0$;
5  $S_{best} \leftarrow S_0$;
6  While $(\mathtt{it} \leq \mathtt{best}_{\mathtt{it}} + \mathtt{max}_{\mathtt{it}})$ do
7      $it_{sol} \leftarrow it$;
8      While $(\mathtt{it} \leq \mathtt{it}_{\mathtt{sol}} + \mathtt{it}_{\mathtt{temp}})$ do
9          $it++$;
10         $S' \leftarrow N(S)$
11         $\triangle \leftarrow cost(S') - cost(S)$;
12         If $\triangle \leq 0$ then :
13             $S \leftarrow S'$
14             $it_{sol} \leftarrow it$
15             If $\mathtt{cost}(\mathtt{S}') < \mathtt{cost}(\mathtt{S}_{\mathtt{best}})$ then :
16                 $S_{best} \leftarrow S'$
17                 $best_{it} \leftarrow it$
18             end − if
19         end − if
20         Else
21              Generate random number $r$
22             If $(r < e^{\triangle/T})$Then :
23                 $S \leftarrow S'$;
24             end-if
25         end
26     end
27     $T \leftarrow \alpha T$
28 end
29 Return S;
30 Apply Local Search in the best Sequence;
end-procedure Simulated Annealing
```

Fig. 6.3 Pseudo-code de Recuit simulé

première phase consiste à construire une solution réalisable. Après cette phase, la solution acquise au stade précédent subit une recherche locale pour atteindre le minimum local. La meilleure solution est stockée en résultat partiel. À la fin de toutes les itérations, le meilleur des résultats partiels est adopté. (Arroyo e Pereira [108] and Resende and Ribeiro [83]) Le pseudo-code 6.4 montre les étapes du *GRASP*.

```
Procedure GRASP ($Maxiter$), ($Seed$)
1  Read input()
2  $iter \leftarrow 0$;
3      while $iter < maxiter$ do
4          $Solution \leftarrow Construction(seed)$;
5          $LocalSearch(Solution)$;
6          Update-Solution (Solution, Best-Solution);
7      end-while
8      return (Best-Solution)
9  end-procedure GRASP
```

Fig. 6.4 GRASP Pseudocode

L'algorithme GRASP-NEH

L'algorithme $GRASP-NEH$ proposé suit toutes les étapes de l'algorithme *GRASP* classique. La phase de construction commence par une échelle vide et la liste des candidats qui formeront la séquence initiale, compte tenu de la demi-ligne 1 et de la semi-ligne 2 séparément avec la machine de synchronisation. Dans cet algorithme, la liste des candidats (CL) est formée par la séquence des tâches en ordre décroissant de la valeur du temps de traitement dans chaque demi-ligne parallèle l. La liste des candidats restreints (*RCL*) est générée à partir de la sélection de $\alpha\%$ des candidats de la séquence *CL*. Ensuite, l'une des tâches sont sélectionnés ramdomly à partir de *RCL* et l'allocation des tâches dans la séquence finale est effectuée selon l'algorithme NEH. L'algorithme de construction $GRASP-NEH$ est présenté dans la Figure 6.5.

Ensuite, l'étape d'exploration locale *GRASP* commence à rechercher des solutions optimales, de sorte que la séquence initiale est acquise à l'étape précédente. Dans chaque itération, tous les voisins possibles de la solution actuelle *S* sont analysés en se déplaçant uniquement vers le voisin qui a la valeur la plus favorable de makespan compte tenu de la fonction objective. La liste des meilleures solutions partielles locales est formée et le résultat partiel de la valeur plus bas du makespan sera utilisé comme solution finale.

```
**Procedure** Construction-NEH ( $itermax, \alpha$ )
1     $Solution1 \leftarrow \emptyset$;
2     $Solution2 \leftarrow \emptyset$;
3     Calculate $\sum pr_{ij}$ of the job $i$ in the machine $j$ in the semi-line $l$;
4     $CL_l \leftarrow$ descending order of the jobs for each semi-line $l$;
5     $RCL_1 \leftarrow$ the $\alpha\%$ of the jobs of LC1;
6     $RCL_2 \leftarrow$ the $\alpha\%$ of the jobs LC2;
7     **while** $Seq1$ and $Seq2$ is not complete **do:**
8         Select randomly a job list $RCL1$;
9         Select randomly a job list $RCL2$;
10        Allocate the jobs in Seq1 according to the NEH heuristic;
11        Allocate the jobs in Seq2 according to the NEH heuristic;
12        Update $RCL_1$
13        Update $RCL_2$
14    **end-while**
15    Solution1 $\leftarrow$ $makespan\ Seq1$;
16    Solution2 $\leftarrow$ $makespan\ Seq2$;
17    Adopt a sequence of smaller makespan;
18    Return (best makespan);
19    Return (best sequence);
**end-procedure Constrution-NEH**
```

Fig. 6.5 Constrution *NEH* Pseudo-code

L’algorithme GRASP-JOHN

La phase de construction de l’algorithme GRASP-Johnson considère le temps de traitement moyen le plus élevé de la tâche dans toutes les machines des demi-lignes et le temps de traitement de la machine de synchronisation. À ce stade, l’algorithme s’applique à l’adaptation des règles de Johnson. Dans ce sens, le temps de traitement moyen le plus élevé des tâches dans les machines des demi-lignes est utilisé comme temps de traitement des tâches de la première machine de Johnson. Le temps de traitement de la tâche dans la machine de synchronisation est considéré comme le temps de traitement de la deuxième machine de Johnson. L’heuristique sélectionne un élément itérativement pour chaque échelle en fonction de ce qui suit:

```
Procedure Constrution Johnson( $itermax, \alpha$ )
1  $Solution \leftarrow \emptyset$;
2  machine1 ← increasing order of highest
3              value of $\sum prijl/m$ of the semi-lines jobs;
4  machine2 ← increasing order of $prij$
5                 of the job in the synchronization machine;
4      $RCL_1 \leftarrow \alpha$ % machine 1 jobs;
5      $RCL_2 \leftarrow \alpha$ % machine 2 jobs;
6  while Seq is not complete do:
7      Randomly select a job of the $(RCL1)$;
8      Randomly select a job of the $(RCL2)$;
9      $seq$ ← seq de Johnson;
10     Update $RCL_1$
11     Update $RCL_2$
12 end-while
13 Solution← makespan $seq$;
14 Return( Solution);
end-procedure Construction Johnson
```

Fig. 6.6 construction par algorithmes de Johnson Pseudo-code

La construction de génération de pseudo-code par les algorithmes de Johnson est montrée dans la Figure 6.6. La séquence générée dans cette étape de l’algorithme *GRASP* Johnson est utilisée comme solution initiale de la recherche locale. Tous les voisins possibles de la solution ”*s*” actuelle sont analysés à chaque itération qui se déplace vers le voisin qui a la valeur la plus favorable de makespan en fonction de la fonction objective. La liste des meilleures solutions partielles locales est formée de sorte que le résultat partiel partiel soit utilisé comme solution finale. Le pseudo-code pour générer le meilleur voisin est présenté dans la Figure 6.2.

6.3 Résultats et méthodologie

Les différentes méthodes pour résoudre le problème de flow shop avec des lignes finales parallèles avec une opération de synchronisation finale ont été testées parmi les différentes instances. Il y a 240 cas différents. Ils sont composés de combinaisons de nombre de tâches x nombre de machines de la demi-ligne 1 x nombre de machines de la demi-ligne 2. Les différentes instances utilisées dans cette étude ont été générées avec le même principe de la méthode rapportée par Taillard [101]. L'algorithme de Taillard [101]) a été adapté à ce problème en tenant compte de la formation d'une seule ligne avec le nombre de tâches, le nombre de machine de la semi-ligne 1, le nombre de machine de la semi-ligne 2 et la machine de synchronisation.

Il y a 10 différents ensembles des données pour chaque combinaison. Les premières 130 combinaisons ont été crées étant donné que les deux demi-lignes ont le même nombre de machines. Les autres 110 combinaisons ont été crées compte tenu de différents nombres de machines dans les demi-lignes. Pour ces problèmes, des nombres de 10, 20, 50 et 100 tâches et des nombres de 3, 5, 7 et 11 machines ont été pris en compte. Parmi les 240 cas, la solution optimale n'a pas été trouvée dans les 41 d'entre eux. Les tests actuels ont été menés à l'aide d'un ordinateur avec les paramètres suivants: Intel C emph Core TM iR 3.1 GHz avec 4 Go de mémoire. Les résultats du modèle mathématique ont été obtenus en utilisant le logiciel emph CPLEX 12.6.1 et les algorithmes ont été implémentés en $C++$. L'analyse statistique des résultats a été obtenue en utilisant le Minitab 17.2.1. software. Pour comparer les résultats, l'écart relatif absolu (GAP) a été utilisé comme as in Ruiz and Maroto [87], qui dans ce cas est la variation entre la solution optimale et les solutions de méthode créées. Le GAP sera calculé par l'équation: 6.14.

$$GAP = \frac{Met_{sol} - OPT_{sol}}{OPT_{sol}} * 100 \tag{6.14}$$

Où: Met_{sol} Correspond au makepan réalisé par une méthode proposée; et OPT_{sol} au makepan réalisé avec le modèle mathématique.

L'écart type de l'écart relatif moyen a également été utilisé pour comparer les meilleures méthodes. L'écart-type d'un échantillon mesure le degré de dispersion des éléments autour de la moyenne. Dans cette étude, l'écart-type de la GAP est la valeur de la variation des écarts relatifs d'une classe de problèmes autour de l'écart relatif moyen. Plus la valeur de l'écart-type est faible, meilleure est la méthode utilisée lors de la comparaison d'une méthode avec l'autre. L'écart type est calculé par les équations 6.15 and 6.16:

$$GAP_{inst} = \frac{1}{Ninst} * \sum_{i=1}^{Ninst} (GAP_i - \overline{GAP_i})^2 \tag{6.15}$$

$$SD = \sqrt{GAP_{inst}} \tag{6.16}$$

Où: $Ninst$ correspond au numéro d'instance; AVG_{inst} à l'écart relatif absolu pour chaque ensemble d'instances; AVG_i à l'écart relatif absolu de l'instance i; et SD l'écart type.

Les tests ont été effectués dans deux environnements. Dans le premier, on a considéré le même nombre de machines dans chaque demi-ligne et, dans le deuxième, on a considéré différents nombres de machines dans les demi-lignes. Un temps limite de 7.200 secondes pour obtenir une solution optimale a été considéré.

6.3.1 Mathematical modeling

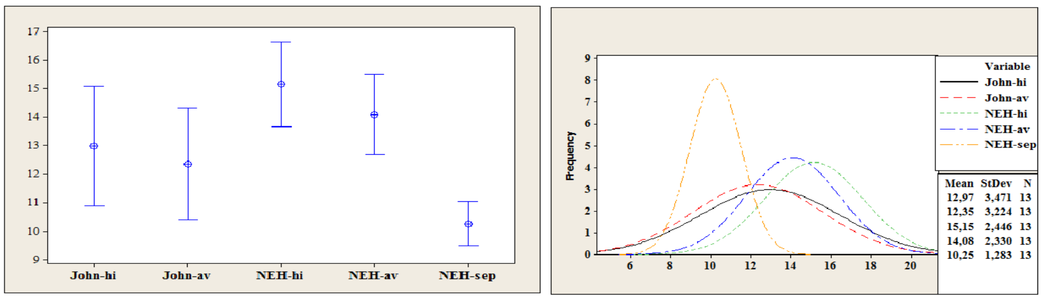
Deux lignes parallèles suivies d'une ligne séquentielle ont été considérées pour l'approche mathématique du problème, en regroupant les opérations en machines parallèles et en adoptant en tant que temps de traitement d'une opération correspondante ou d'un ensemble d'opérations du groupe plus lent. L'existence de deux semi-lignes artificiellement indépendantes permettra de déterminer la valeur la plus élevée de c_n^1 le temps de traitement total (makespan) entre eux. Ces machines fictives représentent lorsque les deux opérations indépendantes traitent la dernière opération.

À travers les résultats, on a observé que pour un petit nombre de tâches et de machines, le makespan a été trouvé en temps de calcul faible. D'autre part, il n'a pas été possible de trouver la solution optimale pour les instances plus vastes avec faible temps de calcul. En raison de ce fait, d'autres stratégies d'approche pour le problème seront utilisées (heuristiques constructives telles que l'algorithme NEH et les métaheuristiques).

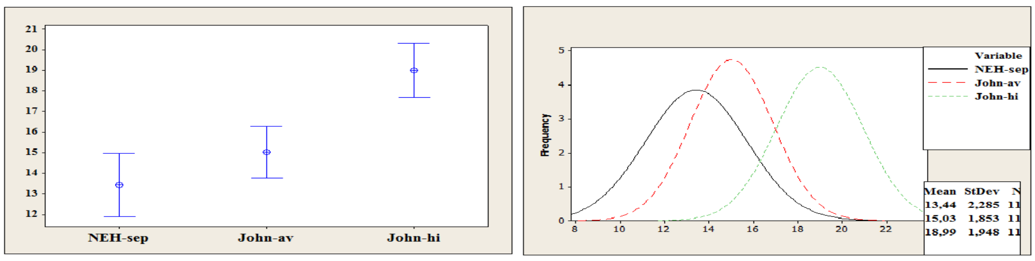
6.3.2 *NEH* heuristic and Johnson's algorithm

Les méthodes proposées sont testées sur différentes instances avec un même nombre de machines dans chaque semi-ligne, où trois adaptations de l'algorithme de NEH.

Les méthodes proposées ont été testées pour différentes instances avec un même nombre de machines dans chaque semi-ligne, où trois adaptations heuristiques (NEH_{av}, NEH_{hi}, NEH_{sep}) et deux adaptations de l'algorithme Johnson ($John_{hi}$, $John_{av}$) ont été considerés.



a- Same number of machine



b- Different number of machine

Fig. 6.7 Gap et sd pour *NEH* et Johnson Adaptation

Dans le deuxième environnement, différents nombres de machines dans les demi-lignes, NEH_{sep}, $John_{hi}$ and $John_{av}$ ont été considerés.

Les résultats obtenus pour différentes instances avec le même nombre de machines (E) sont présentés dans la Figure 6.7. Lorsque l'environnement dans lequel le nombre de machines est le même, on peut affirmer que les adaptations de NEH ont montré que la meilleure performance était celle qui a utilisé l'analyse des demi-lignes séparément (NEH_{sep}) avec un GAP e moyen de 10 % et écart type de 0.05. La variante qui a montré le pire résultat a été celle qui a analysé le temps de traitement le plus long (NEH_{hi}) dans chaque machine parallèle avec GAP moyen de 16 % et écart type de 0.02.

En ce qui concerne les variantes de Johnson, on peut observer une petite différence entre l'écart de l'adaptation $John_{hi}$ et $Jonh_{av}$. La variante avec la meilleure performance était celle qui considérait le temps de traitement moyen le plus long $Jonh_{av}$ dans toutes les machines des semi-lignes pour la première machine et le temps de traitement dans la machine de synchronisation pour la deuxième machine.

Pour l'environnement où le nombre de machines est différent dans les demi-lignes, NEH_{sep} a mieux performé avec 14%. La variante $John_{hi}$ a présenté une GAP moyen légèrement supérieure à $John_{av}$ (19%). Pour vérifier la validité statistique des résultats, une analyse de la variance ($ANOVA$) avec un niveau de confiance de 95% a été prise comme indiqué dans la Figure 6.7.

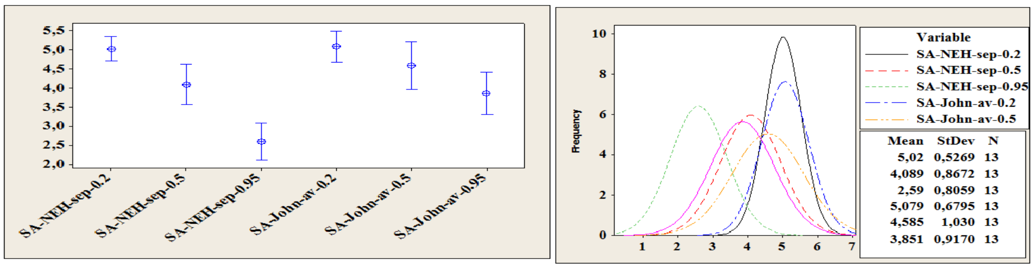
Cependant, ces résultats peuvent être améliorés afin d'être plus proches de la solution optimale. Par conséquent, les métaheuristiques ont été utilisées pour résoudre le problème afin d'obtenir des résultats optimaux ou proche de l'optimum.

6.3.3 Recuit simulé

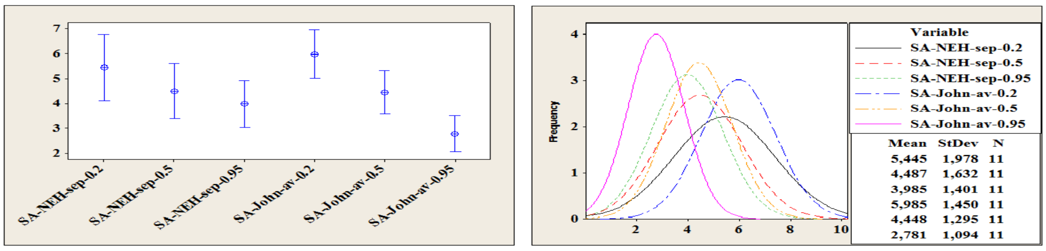
Pour des résultats encore meilleurs, le recuit simulé avec une méthode de recherche locale itérée a été choisi pour affiner les résultats et créer une solution de meilleure qualité.

Cette méthode comporte deux types de solutions initiales comme point de départ qui est généré de manière similaire aux méthodes antérieures. La première méthode considère l'algorithme de Johnson avec le temps de traitement moyen de la semi-ligne comme solution initiale ($John_{av}$) et le deuxième considère l'algorithme NEH_{sep}. Pour cette méthode, la sélection des paramètres utilisés dans la méthode est décrite ci-dessous. Le type de recherche locale utilisé pour cette méthode était la simple perturbation, où une séquence voisine est obtenue essentiellement à partir de la séquence actuelle en supprimant une tâche de sa position et en l'insérant dans une autre position.

Pour cette étude, le GAP est considéré comme la variation entre la solution optimale ou la meilleure solution trouvée par la méthode mathématique, et les solutions trouvées par:



a- Same number of machine –SA



b- Different number of machine -SA

Fig. 6.8 Gap et sd pour Simulated Annealing méthodes

recuit simulé avec des solutions initiales avec la heuristique NEH_{sep} et l'algorithme $John_{av}$ compte tenu du même nombre de machines et de différents numbres de machine dans chaque demi-ligne.

Les paramètres utilisés ont été basés dans les études de Senthilkumar et Narayanan [94] et Salem [5] avec quelques ajustements pour les caractéristiques de ce problème. Les paramètres qui ont donné des résultats significatifs étaient ceux avec une température initiale de $Ti = 6000$. La valeur du facteur de refroidissement utilisée dans l'algorithme affecte le temps d'exécution et la qualité souhaitée pour la solution. Dans cette étude ont été analysés le facteur de refroidissement $\alpha = 0,2, 0,5$ et $0,95$ et la température finale $= 0,0001$. Analyse de la variance (*ANOVA*) avec un niveau de signification de $0,05$.

Pour chaque scénario (semi-lignes avec le même nombre de machines et avec différents nombres de machines), la Figure 6.8 présente les résultats d'utilisation de variantes de la méthode de recuit simulée pour le problème en question. $NEH_{sep} - SA - ILS$ (recuit simulé à partir de NEH_{sep} algorithme et $Jonh_{av} - SA - ILS$ (recuit simulé à partir de $John_{av}$ et se terminant par une recherche locale itérée) ont présenté de meilleurs résultats que ceux précédemment utilisés dans les mêmes instances définies dans deux environnements. Pour l'environnement avec le même nombre de machines dans les demi-lignes parallèles, la meilleure différence relative moyenne (*GAP*) était de $2,59$ pour $NEH_{sep} - SA - ILS$ avec $\alpha = 0,95$. Le pire GAP était de $5,07$ en méthode $John_{av} - SA - ILS$ avec $\alpha = 0,2$. Pour l'environnement avec différents nombres de machines, le meilleur *GAP* était 3.45 en $Jonh_{av} - SA - ILS$ méthode pour $\alpha = 0.95$. Le pire GAP était de $5,61$ en méthode $John_{av} - SA - ILS$ pour $\alpha = 0,2$. Ces résultats ont également montré une amélioration du *GAP* par rapport aux résultats de la méthode antérieure.

6.3.4 GRASP

La dernière méthode proposée était une adaptation de l'algorithme *GRASP* original avec les principes de l'algorithme NEH_{sep} ou $John_{av}$. La première phase est la construction de la solution initiale et la deuxième phase est l'amélioration de cette solution (recherche locale).

Dans cette étude, la phase de construction de la solution initiale établit d'abord la liste des candidats (*LC*) avec la séquence des tâches par ordre décroissant du temps de traitement, lorsque NEH_{sep} est utilisé et par ordre croissant lorsque l'algorithme de Johnson est utilisé. Après cette étape, la liste restreinte est formée par le tirage de $\alpha\%$ de la liste des candidats, ce qui rend chaque solution initiale différente de l'autre. Les valeurs de α pour l'algorithme ont été définies comme $\alpha = 0,2, 0,3$ et $0,5\%$ du nombre de tâches dans la liste des candidats. La séquence est formée par l'attribution des tâches en utilisant NEH_{sep} ou $John_{av}$. À la fin

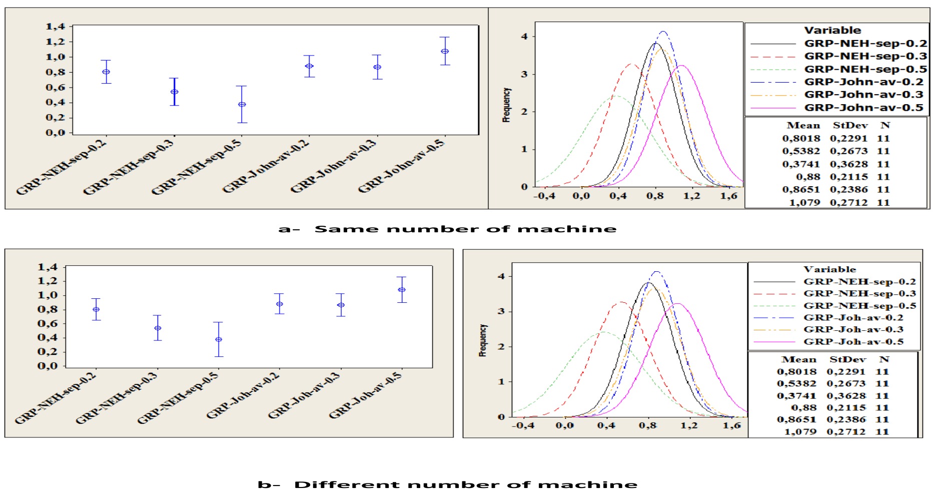


Fig. 6.9 Gap et sd pour GRASP méthode

	IP model		SA-ILS				GRASP			
Instances	opt	time(s)	opt	gap(%)	sd	time(s)	opt	gap(%)	sd	time(s)
$E10\times03$	10	0.3	3	2.01	0.72	38.9	6	0.97	0.22	27.2
$E10\times05$	10	0.5	4	2.86	0.89	38.3	6	0.12	0.14	34.2
$E10\times07$	10	1.9	4	3.29	0.77	50.2	6	0.12	0.13	48.5
$E10\times11$	10	10.7	2	1.04	0.76	51.3	4	0.26	0.13	46.3
$E20\times03$	10	0.3	4	2.03	0.72	49.9	6	0.95	0.16	43.5
$E20\times05$	10	405.0	0	2.42	0.66	50.6	5	0.12	0.15	45.2
$E20\times07$	10	2289.0	0	2.12	0.86	50.5	5	0.19	0.09	43.9
$E20\times11$	4	7541.8	1	2.07	0.76	55.1	4	0.30	0.15	43.2
$E50\times03$	10	5.8	3	3.21	0.78	52.4	5	0.19	0.16	46.2
$E50\times05$	8	4915.7	2	3.12	0.78	53.2	4	0.16	0.14	47.5
$E50\times07$	1	13695.5	0	4.32	0.74	65.2	0	0.20	0.13	57.4
$E50\times11$	0	12746.2	0	2.59	0.75	74.2	0	0.13	0.14	57.4
$E100\times03$	10	160.69	3	2.62	0.82	88.2	4	0.13	0.22	57.4

Table 6.1 SA-ILS-NEH-0.95 / GRASP-NEH-0.5

du processus de construction de la solution initiale, la deuxième phase de *GRASP* commence. L'algorithme se termine quand il y a 100 itérations sans aucune amélioration de la solution.

Figure 6.9 aide à l'observation des comportements du *GAP* des méthodes $NEH_{sep}-GRP-ILS$ et $John_{av}-GRP-ILS$. Ces méthodes ont été meilleures que toutes les autres utilisées précédemment. Il est également observé que cette méthode a atteint un grand nombre de solutions optimales par rapport à toutes les autres méthodes. $NEH_{sep}-GRP-ILS$ avec α

	IP model		$\text{John}_{av} - SA - ILS$				$\text{NEH}_{sep}GRASP$			
Instances	opt	time(s)	opt	gap(%)	sd	time(s)	opt	gap(%)	sd	time(s)
$D10 \times 03 \times 05$	10	0.1	3	4.32	2.84	42.5	6	0.07	0.4	27.0
$D10 \times 03 \times 07$	10	0.2	4	4.21	1.95	46.6	6	0.65	0.4	33.4
$D10 \times 03 \times 11$	10	0.9	2	3.22	3.86	50.2	6	0.84	0.4	47.8
$D10 \times 05 \times 07$	10	0.3	3	3.35	3.21	55.4	5	0.69	0.3	45.7
$D10 \times 05 \times 11$	10	1.4	3	4.03	1.86	48.7	5	0.01	0.3	44.0
$D20 \times 03 \times 05$	10	4.1	4	3.02	2.81	49.6	5	0.65	0.2	47.8
$D20 \times 03 \times 07$	10	52.9	3	2.32	2.84	50.7	4	0.89	0.3	43.2
$D20 \times 03 \times 11$	4	14250.2	1	3.11	2.54	52.8	0	0.09	0.3	50.7
$D20 \times 05 \times 07$	10	200.1	0	4.03	3.25	48.0	3	0.12	0.2	48.1
$D20 \times 05 \times 11$	2	18208.3	0	4.25	3.11	50.2	0	0.01	0.3	51.8
$D50 \times 03 \times 07$	10	153.8	2	2.15	2.41	48.6	4	0.08	0.3	57.4

Table 6.2 $\text{John}_{av} - SA - ILS - 0.95/NEH_{sep} - GRASP - 0.5$

= 0.5 a obtenu la meilleure *GAP* moyenne dans les deux types d'environnement 0.35 et 0.37 respectivement par rapport aux environnements avec le même nombre de machines et nombre différentes de machines dans les demi-lignes. $John_{av} - GRP - ILS$ avec α = 0.5 méthode était la variante *GRASP* qui a atteint le *GAP* le plus élevé pour cette méthode (1.07 et 1.08). Les Tables 6.1 et 6.2 affichent la comparaison des meilleurs résultats de l'algorithme proposé par rapport aux deux méthodes indiquées pour la solution initiale. Dans la première colonne de ces tableaux, les ensembles d'instances sont affichés.

Dans la deuxième et la troisième colonnes, le pourcentage d'occurrences que les valeurs optimales (OPT) ont été réalisées et le temps de calcul moyen du modèle mathématique en secondes (CPU). Dans les colonnes suivantes, les *GAP* acquis, l'écart-type *SD* et le temps de calcul moyen de la meilleure méthode de recuit *ILS* et de la meilleure méthode *GRASP* sont affichés.

Les graphiques comparatifs des meilleurs résultats avec les méthodes élaborées sont présentés dans la Figure 6.10. Les graphiques aident à l'observation des comportements *GAP* de la méthode NEH_{sep}, Joh_{av}, $SA - NEH - ILS$ avec $\alpha = 0,95$, et pour le méthode $SA - Jonh - ILS$ avec $alpha = 0,95\%$, $GRASP_{NEH}$ et $GRASP_{John}$. Le degré de dispersion des valeurs *GAP* par rapport à la valeur moyenne (écart type) a également été analysé pour les méthodes qui ont obtenu les meilleurs résultats dans chaque catégorie. Dans la deuxième et la troisième colonnes, le pourcentage d'occurrences que les valeurs optimales (OPT) ont été réalisées et le temps de calcul moyen du modèle mathématique en secondes (CPU). Dans les colonnes suivantes, les *GAP* acquis, l'écart-type *sd* et le temps de calcul moyen de la meilleure méthode de recuit *ILS* et de la meilleure méthode *GRASP* sont affichés.

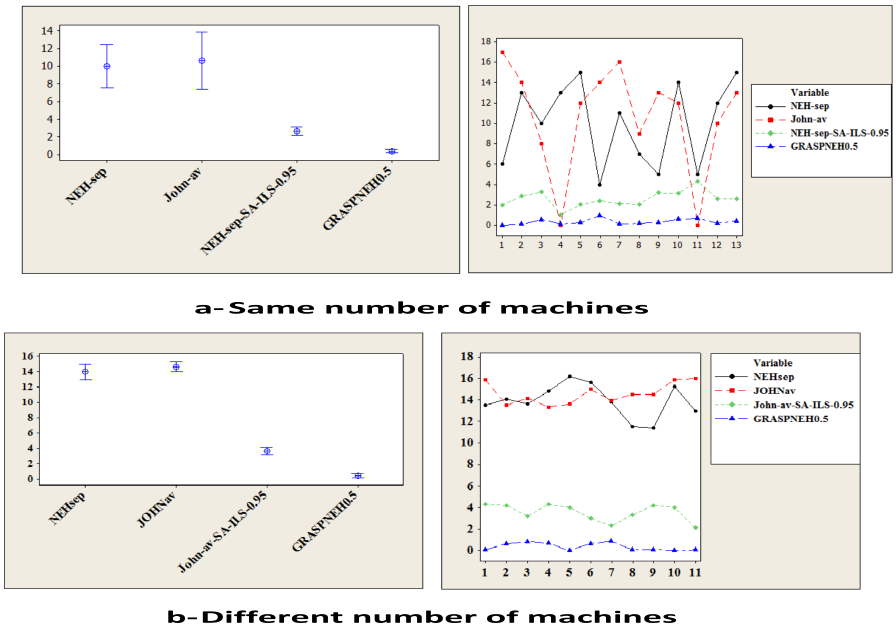


Fig. 6.10 Graphique comparatif des meilleurs résultats des méthodes de résolution

6.4 Conclusion et perspectives

Ce travail a porté sur le développement de méthodes d’optimisation pour résoudre une variante du problème d’ordonnancement atelier de type flow shop semi-parallel avec et une opération de synchronisation finale. Il s’agit d’un problème commun de nombreux secteurs industriels qui requiert des méthodes efficaces pour sa résolution, ce qui a été étudié à partir d’une situation pratique dans un processus de soudage d’une industrie qui fabrique des produits de systèmes électro-électriques.

Cette étude visait à modéliser le problème méthode dans un atelier de type flow shop semi-parallel avec operation de synchronisation finale à la fin des demi-lignes pour la résoudre avec des heuristiques et des métaheuristiques. Dans une première approche, un modèle de programmation linéaire mixte a été défini sur la base d’études déjà développées pour des environnements semblables à ceux étudiés. Le modèle nécessitait des adaptations concernant le traitement des demi-lignes séparément et une attention particulière dans la finalisation des demi-lignes car c’est le point de jonction des éléments produits dans les deux demi-lignes. Le modèle était efficace pour les petites instances, mais pour de plus grandes instances, il présente certaines difficultés pour trouver les solutions optimales dans un temps viable.

En raison de la complexité du problème, il a également été résolu par des heuristiques et des métaheuristiques. A partir des résultats d'une importante campagne de tests, il a été possible de prouver que les constructions de méthodes par heuristiques et métaheuristiques ont donné des résultats réalisables au problème étudié. Les algorithmes présentés, y compris des combinaisons et des hybridations entre elles, ont de bonnes options pour les résolutions d'instances plus vastes. Les algorithmes présentés ont donné de bonnes options pour les résolutions d'instances plus vastes.

La résolution du problème à l'aide de méthodes telles que la règle de Johnson, les méthodes heuristiques comme algorithme *NEH* et les métaheuristiques telles que *ILS*, le recuit simulé et *GRASP* montrent comment ces méthodes se sont comportées dans cet environnement et le possibilité de trouver un résultat optimal en les utilisant. De plus, parmi les méthodes $NEH_{hi}, NEH_{av}, NEH_{sep}, John_{av}, John_{hi}, NEH_{sep} - ILS, Joh_{av} - ILS, NEH_{sep} - SA - ILS, John_{av} - SA - ILS, NEH_{sep} - GRASP, John_{av} - GRASP$. Le $NEH_{sep} - GRASP$ avec $0,5\%$ était la méthode qui a atteint le plus grand nombre des résultats optimaux et le plus bas *GAP* dans les instances.

Le *GAP* moyen le plus bas a été de $0,35\%$ pour les environnements avec le même nombre de machines dans les demi-lignes et de $0,37\%$ pour les environnements avec un nombre différent de machines dans chaque demi-ligne. D'autre part, la valeur la plus élevée de la moyenne *GAP* générée par l'algorithme présenté le mieux ($GRASP - NEH$) était de 1.07%.

En ce qui concerne les méthodes heuristiques utilisées NEH_{hi} avec *GAP* moyen de $16,0$ % était la pire méthode proposée pour les environnements avec le même nombre de machines dans chaque semi-ligne et pour les environnements avec différents nombres de machines dans chaque demi-ligne $John_{hi}$ moyen *GAP* de 19%. Parmi la méthode utilisant la métaheuristique, la méthode hybride $NEHsep - ILS$ a présenté le pire résultat avec *GAP* de $6,78$ % pour les environnements avec les mêmes machines dans les demi-lignes et *GAP* de $8,87\%$ pour les environnements avec différentes machines dans les demi-lignes. Les résultats obtenus par les méthodes de recuit simulé appliquées aux différents instances du problème ont présenté des résultats moins satisfaisants par rapport aux méthodes *GRASP*.

Cependant, il a été possible de visualiser lorsque le facteur de refroidissement (α) augmente, les résultats s'améliorent. C'est-à-dire lorsque le facteur de refroidissement s'approche de la valeur de 1, le nombre de solutions explorées dans le voisinage augmente, conduisant ainsi à un grand nombre de solutions recherchées. Dans ce sens, le recuit simulé peut également être un bon allié à d'autres méthodes pour résoudre ce problème, puisque cette méthode présente l'avantage d'exploiter de grandes quantités de solutions voisines avec la possibilité d'échapper à l'emplacement optimal.

Enfin, les résultats des expériences indiquent que les méthodes *GRASP* représentent une bonne alternative pour résoudre le problème, en particulier pour les instances plus grande. Cela augmente la viabilité de son utilisation comme moyen d'obtenir de meilleures solutions ou un taux d'atteinte plus élevé aux valeurs optimales et à l'application dans des problèmes réels en temps viable.

References

[1] E.H. Aarts and J.K. Lenstra. *Local search in combinatorial optimization*. Princeton University Press, 2003.

[2] A. H Abdekhodaee and A. Wirth. Scheduling parallel machines with a single server: some solvable cases and heuristics. *Computers & Operations Research*, 29(3):295–315, 2002.

[3] A.H. Abdekhodaee, A. Wirth, and H.S. Gan. Equal processing and equal setup time cases of scheduling parallel machines with a single server. *Computers & Operations Research*, 31(11):1867–1889, 2004.

[4] A. Agnetis, A. Alfieri, and G. Nicosia. A heuristic approach to batching and scheduling a single machine to minimize setup costs. *Computers & Industrial Engineering*, 46(4):793–802, 2004.

[5] A. Al-Salem. A heuristic to minimize makespan in proportional parallel flow shops. *International Journal of Computing & Information Sciences*, 2(2):98, 2004.

[6] K. Alaykŷran, O. Engin, and A. Döyen. Using ant colony optimization to solve hybrid flow shop scheduling problems. *The international journal of advanced manufacturing technology*, 35(5-6):541–550, 2007.

[7] A. Allahverdi and H. Aydilek. Heuristics for the two-machine flowshop scheduling problem to minimise makespan with bounded processing times. *International Journal of Production Research*, 48(21):6367–6385, 2010.

[8] H. Allaoui and A. Artiba. Scheduling two-stage hybrid flow shop with availability constraints. *Computers & Operations Research*, 33(5):1399–1419, 2006.

[9] C. Almeder and R. F. Hartl. A metaheuristic optimization approach for a real-world stochastic flexible flow shop problem with limited buffer. *International Journal of Production Economics*, 145(1):88–95, 2013.

[10] J. E. C. Arroyo, R. dos Santos Ottoni, and A. de Paiva Oliveira. Multi-objective variable neighborhood search algorithms for a single machine scheduling problem with distinct due windows. *Electronic Notes in Theoretical Computer Science*, 281:5–19, 2011.

[11] S. Binato, W. J. Hery, D. M. Loewenstern, and M.G.C. Resende. A grasp for job shop scheduling. In *Essays and surveys in metaheuristics*, pages 59–79. Springer, 2002.

[12] J. Blazewicz, K. H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Handbook on scheduling: from theory to applications*. Springer, 2007.

[13] W. Buzzo and J. V. Moccellin. Production scheduling in flow shop systems using a hybrid heuristic genetic-simulated annealing algorithm. *Gestão & Produção*, 7(3):364–377, 2000.

[14] W. D. Callister, D. G. Rethwisch, et al. *Materials science and engineering: an introduction*, volume 7. Wiley New York, 2007.

[15] H. G. Campbell, R.A. Dudek, and M.L. Smith. A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, 16(10):B–630, 1970.

[16] G. Campos C., F. Dugardin, F. Yalaoui, and R. Kelly. Open shop scheduling problem with a multi-skills resource constraint: a genetic algorithm and an ant colony optimisation approach. *International Journal of Production Research*, 54(16):4854–4881, 2016.

[17] C. F. M. C. Cavalcanti, M. J. F. Souza, F. S.H. Souza, and V. S. Coelho. A heuristic methodology based on grasp, vnd and vns for the solution of the sizing problem in i.p. networks. 2004.

[18] Karunakaran Chakravarthy and Chandrasekharan Rajendran. A heuristic for scheduling in a flowshop with the bicriteria of makespan and maximum tardiness minimization. *Production Planning & Control*, 10(7):707–714, 1999.

[19] J. Chen, J. C. Pan, and C. Lin. A hybrid genetic algorithm for the re-entrant flow-shop scheduling problem. *Expert Systems with Applications*, 34(1):570–577, 2008.

[20] Y. Chen, A. Zhang, G. Chen, and J. Dong. Approximation algorithms for parallel open shop scheduling. *Information Processing Letters*, 113(7):220–224, 2013.

[21] J. Cheng, Y. Karuno, and H. Kise. A shifting bottleneck approach for a parallel-machine flowshop scheduling problem. *Journal of the Operations Research Society of Japan-Keiei Kagaku*, 44(2):140–156, 2001.

[22] TC E. Cheng, J.ND Gupta, and G. Wang. A review of flowshop scheduling research with setup times. *Production and Operations Management*, 9(3):262–282, 2000.

[23] I. M. Coelho, M. N. Haddad, L. S. Ochi, M. J. F. Souza, and R. Farias. A hybrid cpu-gpu local search heuristic for the unrelated parallel machine scheduling problem. In *Applications for Multi-Core Architectures (WAMCA), 2012 Third Workshop on*, pages 19–23. IEEE, 2012.

[24] P. Damodaran, M. C Vélez-Gallego, and J. Maya. A grasp approach for makespan minimization on parallel batch processing machines. *Journal of Intelligent Manufacturing*, 22(5):767–777, 2011.

[25] H. Davoudpour and M. Ashrafi. Solving multi-objective sdst flexible flow shop using grasp algorithm. *The International Journal of Advanced Manufacturing Technology*, 44(7-8):737–747, 2009.

[26] Xingye Dong, Houkuan Huang, and Ping Chen. An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Computers & Operations Research*, 36(5):1664–1669, 2009.

[27] F. Dugardin, F. Yalaoui, and L. Amodeo. New multi-objective method to solve reentrant hybrid flow shop scheduling problem. *European Journal of Operational Research*, 203(1):22–31, 2010.

[28] M. Feldmann and D. Biskup. Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches. *Computers & Industrial Engineering*, 44(2):307–323, 2003.

[29] V. Fernandez-Viagas and J.M. Framinan. NEH-based heuristics for the permutation flowshop scheduling problem to minimise total tardiness. *Computers & Operations Research*, 60:27–36, 2015.

[30] J. V. Frasch, S. O. Krumke, and S. Westphal. Mip formulations for flowshop scheduling with limited buffers. In *Theory and Practice of Algorithms in (Computer) Systems*, pages 127–138. Springer, 2011.

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

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

[33] Michael R Garey, David S Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, 1(2):117–129, 1976.

[34] J. W. Gavett. Three heuristic rules for sequencing jobs to a single production facility. *Management Science*, 11(8):156–166, 1965.

[35] C.A. Glass, C.N. Potts, and P. Shade. Unrelated parallel machine scheduling using local search. *Mathematical and Computer Modelling*, 20(2):41–52, 1994.

[36] A.C. Gomes Jr., C.R.V. Carvalho, P.L.A. Munhoz, and M.J.F. Souza. A hybrid heuristic method to solve the problem of sequencing in a machine with penalties for anticipation and delay of the production. *Anais do XXXIX Simpósio Brasileiro de Pesquisa Operacional-XXXIX SBPO, Fortaleza, Brazil*, pages 1649–1660, 2007.

[37] E González-Neira, J Montoya-Torres, and D Barrera. Flow-shop scheduling problem under uncertainties: Review and trends. *International Journal of Industrial Engineering Computations*, 8(4):399–426, 2017.

[38] I.F.G. Guimarães, Y. Ouazene, M. C. de Souza, and F. Yalaoui. Semi-parallel flow shop with a final synchronization operation scheduling problem. *IFAC-PapersOnLine*, 49(12):1032–1037, 2016.

[39] A. Gupta and S. Chauhan. A heuristic algorithm for scheduling in a flow shop environment to minimize makespan. *International Journal of Industrial Engineering Computations*, 6(2):173–184, 2015.

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

[41] F. Hnaien, F. Yalaoui, and A. Mhadhbi. Makespan minimization on a two-machine flowshop with an availability constraint on the first machine. *International Journal of Production Economics*, 164:95–104, 2015.

[42] J. Hurkała and A. Hurkała. Effective design of the simulated annealing algorithm for the flowshop problem with minimum makespan criterion. *Journal of Telecommunications and Information Technology*, pages 92–98, 2012.

[43] P. Jarosław, S. Czesław, and Ż. Dominik. Optimizing bicriteria flow shop scheduling problem by simulated annealing algorithm. *Procedia Computer Science*, 18:936–945, 2013.

[44] C. Jing, G. Tang, and X. Qian. Heuristic algorithms for two machine re-entrant flow shop. *Theoretical Computer Science*, 400(1):137–143, 2008.

[45] S. M. Johnson. Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, 1(1):61–68, 1954.

[46] P. J. Kalczynski and J. Kamburowski. An improved NEH heuristic to minimize makespan in permutation flow shops. *Computers & Operations Research*, 35(9):3001–3008, 2008.

[47] S. Kirkpatrick, C .D. Gelatt, M. P. Vecchi, et al. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[48] C. Koulamas and G. J. Kyparisis. Single-machine and two-machine flowshop scheduling with general learning functions. *European Journal of Operational Research*, 178(2):402–407, 2007.

[49] J. Li and Q. Pan. Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm. *Information Sciences*, 316:487–502, 2015.

[50] K.P. Li, V.K. Ganesan, and A.I. Sivakumar*. Synchronized scheduling of assembly and multi-destination air-transportation in a consumer electronics supply chain. *International Journal of Production Research*, 43(13):2671–2685, 2005.

[51] X. Li, L. Amodeo, F. Yalaoui, and H. Chehade. A multiobjective optimization approach to solve a parallel machines scheduling problem. *Advances in Artificial Intelligence*, vol.2010:10.1155/2010/943050, 2010.

[52] S. Lin, C. Huang, C. Lu, and K. Ying. Minimizing total flow time in permutation flowshop environment. *International Journal of Innovative Computing, Information and Control*, 8(10A):6599–6612, 2012.

[53] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search. In *Handbook of metaheuristics*, pages 320–353. Springer, 2003.

[54] C. Low, J. Yeh, and K. Huang. A robust simulated annealing heuristic for flow shop scheduling problems. *The International Journal of Advanced Manufacturing Technology*, 23(9-10):762–767, 2004.

[55] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3(Supplement C):43 – 58, 2016.

[56] M. Magazine, J. François, and N. Hall. Generalized preemption models for single-machine dynamic scheduling problems. *IIE transactions*, 29(5):359–372, 1997.

[57] G.B. Mainieri and D. P. Ronconi. New heuristics for total tardiness minimization in a flexible flowshop. *Optimization Letters*, pages 1–20, 2013.

[58] M. K. Marichelvam, T. Prabaharan, and X. S. Yang. A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems. *IEEE transactions on evolutionary computation*, 18(2):301–305, 2014.

[59] G. N. Maschietto, Y. Ouazene, F. Yalaoui, M. C. de Souza, and M. G. Ravetti. Two formulations for non-interference parallel machine scheduling problems. *IFAC-PapersOnLine*, 48(3):272–276, 2015.

[60] S. T. McCormick and U. S. Rao. Some complexity results in cyclic scheduling. *Mathematical and Computer Modelling*, 20(2):107–122, 1994.

[61] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[62] H.S. Mirsanei, M. Zandieh, M. J. Moayed, and M.R. Khabbazi. A simulated annealing algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 22(6):965–978, 2011.

[63] T.E. Morton and D.W. Pentico. *Heuristic scheduling systems*. John Wiley Sons, 1993.

[64] B. Naderi, M. Aminnayeri, M. Piri, and M.H. Ha'iri Yazdi. Multi-objective no-wait flowshop scheduling problems: models and algorithms. *International Journal of Production Research*, 50(10):2592–2608, 2012.

[65] M.S. Nagano and J.V. Moccellin. Reducing mean flow time in permutation flow shop. *Journal of the Operational Research Society*, 59(7):939–945, 2008.

[66] M. Nawaz, E. E. Enscore, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95, 1983.

[67] Andreas C Nearchou. Flow-shop sequencing using hybrid simulated annealing. *Journal of Intelligent manufacturing*, 15(3):317–328, 2004.

[68] M. Nikabadi and R. Naderi. A hybrid algorithm for unrelated parallel machines scheduling. *International Journal of Industrial Engineering Computations*, 7(4):681–702, 2016.

[69] E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1):160–175, 1996.

[70] E. Nowicki and C. Smutnicki. The flow shop with parallel machines: A tabu search approach. *European Journal of Operational Research*, 106(2):226–253, 1998.

[71] G. C. Onwubolu. *Emerging optimization techniques in production planning and control*, volume 84. World Scientific, 2002.

[72] I.H. Osman and C.N. Potts. Simulated annealing for permutation flow-shop scheduling. *Omega*, 17(6):551–557, 1989.

[73] G. Özdağoğlu. A simulated annealing application on plowshop sequencing problem: A comparative case study. *Atatürk Üniversitesi İktisadi ve İdari Bilimler Dergisi*, 22(2), 2008.

[74] Q-K. Pan, L. Wang, L. Gao, and W.D. Li. An effective hybrid discrete differential evolution algorithm for the flow shop scheduling with intermediate buffers. *Information Sciences*, 181(3):668–685, 2011.

[75] S. Parveen and H. Ullah. Review on job-shop and flow-shop scheduling using. *Journal of Mechanical Engineering*, 41(2):130–146, 2011.

[76] M. Pincus. Letter to the editor—a monte carlo method for the approximate solution of certain types of constrained optimization problems. *Operations Research*, 18(6):1225–1228, 1970.

[77] M. Pinedo. *Planning and scheduling in manufacturing and services*. Springer, 2005.

[78] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, 2008.

[79] G. Prabhaharan, B. S. H. Khan, and L. Rakesh. Implementation of grasp in flow shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 30(11-12):1126–1131, 2006.

[80] S. F. Rad, R. Ruiz, and N. Boroojerdian. New high performing heuristics for minimizing makespan in permutation flowshops. *Omega*, 37(2):331–345, 2009.

[81] E. Rashidi, M. Jahandar, and M. Zandieh. An improved hybrid multi-objective parallel genetic algorithm for hybrid flow shop scheduling with unrelated parallel machines. *The International Journal of Advanced Manufacturing Technology*, 49(9-12):1129–1139, 2010.

[82] C. R. Reeves. Genetic algorithms for the operations researcher. *Informs Journal on Computing*, 9(3):231–250, 1997.

[83] M. G.C. Resende and C. C. Ribeiro. Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In *Handbook of metaheuristics*, pages 283–319. Springer, 2010.

[84] I. Ribas, R. Leisten, and J. M. Framiñan. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Computers & Operations Research*, 37(8):1439–1454, 2010.

[85] A. P Rifai, H. Nguyen, and S. Z. Md. Dawal. Multi-objective adaptive large neighborhood search for distributed reentrant permutation flow shop scheduling. *Applied Soft Computing*, 40:42–57, 2016.

[86] D. P. Ronconi and E. G. Birgin. Mixed-integer programming models for flowshop scheduling problems minimizing the total earliness and tardiness. In *Just-in-Time systems*, pages 91–105. Springer, 2012.

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

[88] R. Ruiz and T. Stützle. 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, 2008.

[89] R. Ruiz and J. A. Vázquez-Rodríguez. The hybrid flow shop scheduling problem. *European Journal of Operational Research*, 205(1):1–18, 2010.

[90] R. K. Samuel and P. Venkumar. Some novel methods for flow shop scheduling. *International Journal of Engineering Science and Technology*, 3(12):8395–8403, 2011.

[91] B. Santosa and A. Rofiq. The development of simulated annealing algorithm for hybrid flow shop scheduling problem to minimize makespan and total tardiness. In *International Conference on Industrial Engineering and Operations Management*, pages 1348–1355, 2014.

[92] M. Sayadi, R. Ramezanian, and N. Ghaffari-Nasab. A discrete firefly meta-heuristic with local search for makespan minimization in permutation flow shop scheduling problems. *International Journal of Industrial Engineering Computations*, 1(1):1–10, 2010.

[93] P. Schuurman and G. J. Woeginger. Preemptive scheduling with job-dependent setup times. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 759–767. Society for Industrial and Applied Mathematics, 1999.

[94] Panneerselvam Senthilkumar, Sockalingam Narayanan, et al. Simulated annealing algorithm to minimize makespanin single machine scheduling problem withuniform parallel machines. *Intelligent Information Management*, 3(01):22, 2011.

[95] S. A. Seyed-Alagheband, H. Davoudpour, S. H. Doulabi, and M. Khatibi. Using a modified simulated annealing algorithm to minimize makespan in a permutation flow-shop scheduling problem with job deterioration. In *Proceedings of the world congress on engineering and computer science*, volume 2, pages 20–22, 2009.

[96] B. Shahul H. K., G. Prabhaharan, and P. Asokan. A grasp algorithm for m-machine flowshop scheduling problem with bicriteria of makespan and maximum tardiness. *International Journal of Computer Mathematics*, 84(12):1731–1741, 2007.

[97] O. Shahvari and R. Logendran. Hybrid flow shop batching and scheduling with a bi-criteria objective. *International Journal of Production Economics*, 179:239–258, 2016.

[98] X. Shao, B. Wang, Y. Rao, L. Gao, and C. Xu. Metaheuristic approaches to sequencing mixed-model fabrication/assembly systems with two objectives. *The International Journal of Advanced Manufacturing Technology*, 48(9-12):1159–1171, 2010.

[99] P. Sivasankaran, T. Sornakumar, and R. Panneerselvam. Design and comparison of simulated annealing algorithm and grasp to minimize makespan in single machine scheduling with unrelated parallel machines. 2010.

[100] E.F. Stafford. On the development of a mixed-integer linear programming model for the flowshop sequencing problem. *Journal of the Operational Research Society*, pages 1163–1174, 1988.

[101] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.

[102] F.T. Tseng, E. F. Stafford Jr., and J.N.D Gupta. An empirical analysis of integer programming formulations for the permutation flowshop. *Omega*, 32(4):285–293, 2004.

[103] C. R. Vaz and L. A. Araki. Sequenciamento da produção em linhas paralelas não-idênticas. 2007.

[104] H. M. Wagner. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6(2):131–140, 1959.

[105] L. Wang, Q. Pan, and M. F. Tasgetiren. A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem. *Computers & Industrial Engineering*, 61(1):76–83, 2011.

[106] M. Widmer and A. Hertz. A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*, 41(2):186–193, 1989.

[107] J. Yang. A two-stage hybrid flow shop with dedicated machines at the first stage. *Computers & Operations Research*, 40(12):2836–2843, 2013.

[108] Mehmet Mutlu Yenisey and Betul Yagmahan. Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends. *Omega*, 45:119–135, 2014.

[109] X. Zhang and S. van de Velde. Approximation algorithms for the parallel flow shop problem. *European Journal of Operational Research*, 216(3):544–552, 2012.