Balancing reconfigurable or robotic assembly lines : exact and hybrid methods

Youssef Lahrichi

LABORATOIRE D'INFORMATIQUE,
DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

THÈSE de DOCTORAT DE L'UNIVERSITÉ CLERMONT AUVERGNE

Opérée au sein du :

Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes

Ecole Doctorale : Sciences pour l'ingénieur

Spécialité de doctorat : Informatique

Soutenue publiquement le 15/01/2021 à 08H30, par :

Mr. Youssef Lahrichi

Balancing reconfigurable or robotic assembly lines: exact and hybrid methods

---

Devant le jury composé de :

Mme. Olga Battaïa	Professeure, Kedge Buisness School	**Rapporteure**
Mr. Jean-Charles Billaut	Professeur, Université de Tours	**Rapporteur**
Mr. Laurent Deroussi	Maître de conférences, Université Clermont Auvergne	**Co-encadrant**
Mme. Nathalie Grangeon	Maître de conférences, Université Clermont Auvergne	**Co-encadrante**
Mr. David Lemoine	Maître-assistant, IMT Atlantique	**Examinateur**
Mme. Sylvie Norre	Professeure, Université Clermont Auvergne	**Directrice de thèse**
Mr. Farouk Yalaoui	Professeur, Université de Technologie de Troyes	**Président du jury**

This research was financed by the French government IDEX-ISITE initiative 16-IDEX-0001 (CAP 20-25).

Remerciements

Tout d'abord, je tiens à remercier mes encadrants Sylvie, Nathalie et Laurent pour les compétences qu'ils m'ont transmises en recherche, en enseignement ainsi qu'en communication. Ils ont fait preuve d'un grand professionnalisme pour mener à bien ce projet de recherche.

Je tiens également à remercier mes collègues chercheurs du LIMOS ainsi que mes collègues enseignants de l'IUT. Mes remerciements vont également à mes étudiants. Les années passées en leur compagnie ont fortement contribué à ma formation doctorale.

Enfin, je dédie ma thèse à ma famille qui m'a soutenu pendant tout mon parcours académique.

Résumé

Le problème d'équilibrage des lignes d'assemblage consiste à affecter un ensemble d'opérations à un ensemble de stations placées le long d'une ligne tout en respectant un certain nombre de contraintes et en minimisant certains objectifs. Le problème est ancien et la littérature est bien fournie dessus. Cependant, peu de travaux considèrent ce problème conjointement avec d'autres problèmes qui paraissent bien justifiés au regard de la nouvelle ère de l'Industrie 4.0. En effet, de nos jours les tâches sur les lignes d'assemblage sont le plus souvent réalisées par des robots, l'opérateur humain ne se contentant que de superviser les opérations. La robotisation fait apparaître deux nouveaux problèmes de décision : le problème d'affectation des robots aux stations et le problème de séquencement des opérations sur chacune des stations. Le premier est justifié par le fait que la durée des opérations dépend du robot utilisé dans la station alors que le second est induit par des temps de setup entre deux opérations qui se suivent sur une station. Ces temps de setup sont notamment nécessaires pour réaliser des changements d'outils sur les robots où pour réaliser des manipulations sur le produit.

Dans un premier temps nous avons étudié le problème d'équilibrage conjointement avec le problème de séquencement des opérations. Le problème est noté SDSALBP (*Sequence-Dependent Simple Assembly Line Balancing Problem*). Plus précisément, nous avons étudié une généralisation de ce problème qui existe dans la littérature : le problème de l'équilibrage des lignes de transfert reconfigurables, noté RTLB (*Reconfigurable Transfert Line Balancing Problem*). Il consiste à affecter des opérations nécessaires pour fabriquer un produit à des stations de travail placées en série. Les stations peuvent être équipées de plusieurs machines qui opèrent en parallèle et des temps de setup inter-opératoires au passage d'une opération à une autre sur une même machine sont considérés. L'objectif est de minimiser le nombre de machines utilisées avec un temps de cycle fixé. La contrainte sur le temps de cycle doit être respectée (pour chaque station, la somme des temps opératoires et des temps inter-opératoires de la séquence des opérations qui lui sont affectées ne doit pas dépasser

le temps de cycle multiplié par le nombre de machines affectées à cette station). Des contraintes de précédence sur les opérations doivent aussi être respectées. Nous prenons également en compte d'autres contraintes industrielles (inclusion, exclusion, accessibilité).

Trois types d'approches ont été proposées pour ce problème :

- Approche intégrée où la répartition des opérations sur les stations de travail et le séquencement des opérations sont faits en même temps. Pour ce type d'approche, nous avons proposé un programme linéaire en nombre entiers.
- Approche de type BFSL ("Balance First, Sequence Last"): Lahrichi et al. (2020b). Elle consiste à faire la répartition des opérations entre les stations de travail d'abord puis à procéder au séquencement des opérations sur chacune des stations. Dans cette méthode, la répartition des opérations aux stations est faite à l'aide d'un programme linéaire en nombres entiers alors que le séquencement des opérations sur chacune des stations est faite à l'aide de la programmation dynamique. Nous démontrons une garantie de performance pour cet algorithme (algorithme d'approximation) et l'utilisons comme algorithme de construction dans une matheuristique à base de recuit simulé.
- Approche de type SFBL ("Sequence First, Balance Last"): Lahrichi et al. (2020a). Elle consiste à faire le séquencement des opérations d'abord puis à procéder à la répartition des opérations entre les stations de travail. Dans ce type de méthodes, nous considérons une «séquence géante» de toutes les opérations qu'il convient de découper (split) afin d'affecter chacune des sous séquences à une station de travail. Notre principale contribution se situe au niveau d'un algorithme polynomial pour la résolution exacte de ce sous-problème. Une recherche locale itérée sur l'espace des séquences géantes a été proposée où l'algorithme de split a été utilisé pour construire et évaluer les séquences. Une méthode exacte de type Branch and Bound utilisant le split a également été proposée. Les expérimentations réalisées sur des instances de benchmark montrent une nette amélioration par rapport à certaines méthodes de la littérature.

Encouragés par les résultats obtenus par la dernière approche, nous avons ajouté une décision supplémentaire à adresser conjointement avec les deux décisions précédentes : le problème d'affectation des robots aux stations. Nous étudions ainsi le *Robotic Assembly Line Balancing Problem-2 (RALBP-2)* qui comprend la décision d'équilibrage et la décision d'affection des robots puis le *Sequence-Dependent Robotic Assembly Line Balancing Problem-2 (SDRALBP-2)* qui comprend en plus des deux décisions précédentes, le séquencement des opérations sur les stations de travail.

Pour ces deux derniers problèmes, les durées des opérations ainsi que les temps de setup dépendent du type de robot choisi sur la station. Nous considérons l'objectif de minimiser le temps de cycle avec un nombre maximum de stations donné.

Le RALBP-2 et le SDRALBP-2 ont été étudiés sous deux hypothèses différentes identifiées dans la littérature :

- Les robots sont typés et chaque type de robot peut être sélectionné plusieurs fois sur des stations différentes.
- Les robots sont individualisés et chaque robot peut être affecté à au plus une station.

Nous proposons pour ces problèmes des méthodes de type Sequence-First Balance-And-Select-Last: Lahrichi et al. (2020c). Ce type de méthodes représente une extension des méthodes de type Sequence-First Balance-Last proposées pour le RTLBP. Ces méthodes reposent sur un algorithme polynomial que nous avons proposé pour résoudre le cas particulier où la séquence des opérations est donnée. Cet algorithme dynamique a été intégré dans une méthode à voisinage. Les résultats sur des instances de la littérature sont prometteurs.

Pour les trois problèmes étudiés dans la thèse, la contribution majeure est le split. Cet algorithme résout le cas particulier de la séquence géante fixée en s'appuyant sur la recherche d'un chemin optimal dans un graphe auxiliaire (Table 1). Nous notons que le cas particulier a aussi été résolu pour d'autres problèmes dans la thèse, notamment le SALBP-2 qui représente le problème d'équilibrage de base où il convient de minimiser le temps de cycle avec un nombre de stations maximum fixé. n représente le nombre d'opérations.

Problème	Problème de graphe sous-jacent	Complexité en temps
RTLBP	Plus court chemin contraint	$O(n^4)$
RALBP-2	Chemin min-max contraint	$O(n^4)$
SDRALBP-2	Chemin min-max contraint	$O(n^4)$

Table 1: Problème de graphe sous-jacent et complexité

Le cas particulier de la séquence géante fixée est posé de manière récurrente dans la littérature car il permet la réduction de l'espace de recherche dans une méthode à voisinage où une solution est codée par une séquence géante. L'utilisation du split permet de préserver une solution optimale. A notre meilleure connaissance, ce cas particulier n'avait été résolu auparavant que par des heuristiques (sans garantie de performance) et par un modèle mathématique (complexité exponentielle) dans Borisovsky, Delorme, and Dolgui (2013) pour ce qui est du RTLBP. Le même constat

est observé pour le RALBP-2, le cas particulier correspondant a été résolu par une méthode exponentielle dans Levitin, Rubinovitz, and Shnits (2006). Cette dernière a été reprise notamment par Nilakantan et al. (2015).

Étant donnés les résultats de la thèse, nous envisageons deux principales perspectives :

- L'optimisation multi-objectif intégrant la minimisation du coût et la maximisation du rendement des lignes d'assemblage robotisées peut être considérée. En particulier, le split peut être adapté pour déterminer toutes les solutions optimales au sens de Pareto respectant une séquence géante, une hybridation avec une métaheuristique multi-objectif est alors envisageable.
- L'intégration du facteur humain dans les lignes d'assemblage robotisées est une perspective intéressante d'un point de vue industriel. Les opérateurs humains peuvent collaborer avec des robots pour réaliser des opérations à forte valeur ajoutée. Des considérations ergonomiques peuvent alors être envisagées.

List of publications

Publications in peer-reviewed journals

A new split-based hybrid metaheuristic for the reconfigurable transfer line balancing problem. Y Lahrichi, N Grangeon, L Deroussi, S Norre *International Journal of Production Research*, 1-18. Published online: February 2020 (18 pages). `www.tandfonline.com/eprint/EH6EE3T3UUFQSP4TZPSV/full?target=10.1080/00207543.2020.1720929`

A Balance-First Sequence-Last Algorithm to design RMS. A Matheuristic with performance guaranty to balance Reconfigurable Manufacturing Systems. Y. Lahrichi, L. Deroussi, N. Grangeon, S. Norre. *Journal of Heuristics*. Special issue on Matheuristics. Accepted on May 2020, to appear. (26 pages)

Publications in international conferences

A Min-Max Path Approach For Balancing Robotic Assembly Lines With Sequence-Dependent Setup Times. Y Lahrichi, L Deroussi, S Norre, N Grangeon. 13^{th} *International Conference on Modeling, Optimization and Simulation - MOSIM* - Agadir (Morocco) - 12-14 Novembre 2020 (10 pages). **Awarded with the third prize for the best paper.**

A Polynomial Algorithm For Balancing a Sequence of Operations in Reconfigurable Transfer Lines. Y Lahrichi, L Deroussi, N Grangeon, S Norre. *IFAC-PapersOnLine* 52 (13), 981-986. 9^{th} *IFAC conference on Manufacturing, Modeling, Management, and Control* - MIM2019 - Berlin (Germany) - 28-30 August 2019. (6 pages)

Reconfigurable transfer line balancing problem: A new MIP approach and approximation hybrid algorithm. Y Lahrichi, L Deroussi, S Norre, N Grangeon. 12^{th}

International Conference on Modeling, Optimization and Simulation - MOSIM - Toulouse (France) - 27-29 June 2018 (8 pages). **Selected for IJPR**.

Multi-Objective Robotic Assembly Line Balancing Problem: A NSGA-II Approach Using Multi-Objective Shortest Path Decoders. Y Lahrichi, N Grangeon, L Deroussi, S Norre. 17^{th} *International Conference on Project Management and Scheduling* - PMS2020 - Toulouse (France) - 15-17 April 2020. (Extended abstract, 4 pages)

Reconfigurable transfer line balancing problem: deterministic and stochastic context. Y Lahrichi, L Deroussi, N Grangeon, S Norre. 29^{th} *European Conference on Operational Research* - EURO - Valencia (Spain) - 8-11 July 2018. (Abstract)

Publications in national conferences

Un algorithme de plus court chemin polynomial pour l'équilibrage des lignes de production reconfigurables. Y Lahrichi, L Deroussi, N Grangeon, S Norre. *ROADEF* - Le Havre - 19-21 Février 2019.

Contents

List of Figures **xii**

List of Tables **xiv**

List of Algorithms **xv**

Introduction **1**

Chapter 1 Assembly line balancing problems **7**
1.1 Introduction 8
1.2 An overview of assembly line balancing problems 9
1.2.1 Line layouts and products 9
1.2.2 The Simple Assembly Line Balancing problem 12
1.3 Reconfigurable Transfer Lines 16
1.3.1 Components of the problem 16
1.3.1.1 Sequence-dependent setup times 17
1.3.1.2 Parallel workstations 21
1.3.1.3 Industrial constraints 25
1.3.2 The Reconfigurable Transfer Line Balancing Problem 25
1.3.2.1 Illustrative example 27
1.3.2.2 State of the art 28
1.3.2.3 A new mathematical model for the RTLBP 29
1.3.2.4 Complexity and lower bounds 32
1.4 Robotic Assembly Lines 32
1.4.1 The Robotic Assembly Line Balancing Problem 33
1.4.1.1 Illustrative example 34
1.4.1.2 State of the art 35
1.4.1.3 A new mathematical model for RALBP 36
1.4.1.4 Complexity and lower bounds 38
1.4.2 The Sequence-Dependent Robotic Assembly Line Balancing Problem 38
1.4.2.1 Illustrative example 39
1.4.2.2 An adapted mathematical model 39

1.4.2.3 Complexity and lower bounds 42
1.5 Research gap and goals of the thesis 43

Chapter 2 A Balance-First Sequence-Last Approach for the RTLBP 47
2.1 Introduction 47
2.2 Heuristic BFSL (H-BFSL): an approximation algorithm of type BFSL 49
2.2.1 Balancing generation 50
2.2.2 Operation sequencing 53
2.2.3 Constraint generation scheme 55
2.3 M-BFSL: A matheuristic of type BFSL 57
2.3.1 Solution' encoding 59
2.3.2 Neighborhood moves 59
2.3.3 Adaptive simulated annealing 62
2.4 Conclusion 63

Chapter 3 A new polynomial split approach for line balancing problems 65
3.1 Introduction 66
3.2 Split approach: an illustrative example 66
3.3 RTLBP 68
3.3.1 Auxiliary graph 69
3.3.2 A constrained shortest path 70
3.3.3 Illustrative example 71
3.4 RALBP 75
3.4.1 RALBP-1/SDRALBP-1 75
3.4.1.1 Auxiliary graph 76
3.4.1.2 A shortest path 76
3.4.1.3 Illustrative example 77
3.4.2 RALBP-2/SDRALBP-2 79
3.4.2.1 Auxiliary graph 79
3.4.2.2 A constrained minmax path 79
3.4.2.3 Illustrative example 80
3.4.2.4 Extension to one robot per type 83
3.5 Conclusion 87

Chapter 4 Sequence-First Balance-Last approaches using split approach 89
4.1 Introduction 89
4.2 A split-based metaheuristic 90
4.2.1 General scheme 90
4.2.1.1 Encoding-decoding scheme 92
4.2.1.2 Metaheuristic scheme 93

4.2.2 Local search and perturbation moves . . . 95
4.2.3 Computing a compatible giant sequence . . . 95
4.2.3.1 RALBP-2 and SDRALBP-2 . . . 96
4.2.3.2 RTLBP . . . 96
4.3 A split-based branch and bound algorithm . . . 101
4.3.1 Branching strategy . . . 102
4.3.2 Exploration strategy . . . 102
4.3.3 Split-based lower bounding technique . . . 104
4.4 Conclusion . . . 106

Chapter 5 Experimental results 107
5.1 Introduction . . . 108
5.2 Reconfigurable Transfer Lines . . . 108
5.2.1 Instances . . . 109
5.2.1.1 Small-size instances . . . 109
5.2.1.2 Large-size instances . . . 109
5.2.2 Exact methods . . . 110
5.2.3 Heuristics . . . 112
5.2.4 Metaheuristics . . . 115
5.2.4.1 M-BFSL . . . 115
5.2.4.2 Split-based ILS . . . 121
5.3 Robotic Assembly lines . . . 127
5.3.1 RALBP-2 . . . 127
5.3.1.1 Instances . . . 127
5.3.1.2 Split-based ILS . . . 127
5.3.2 Sequence-dependent Robotic Assembly Line Balancing Problem-2 . . . 130
5.3.2.1 Instances . . . 131
5.3.2.2 Split-based ILS . . . 131
5.4 Conclusion . . . 133

Conclusion and perspectives 135

Bibliography 138

List of Figures

Figure 1.1 Straight assembly line. 10
Figure 1.2 U-assembly line. 10
Figure 1.3 Parallel workstations. 11
Figure 1.4 Single-model, multi-model and mixed-model lines (Becker and Scholl (2006)) 12
Figure 1.5 Precedence graph. 14
Figure 1.6 Example of feasible solution. 15
Figure 1.7 Example of feasible solution. 23
Figure 1.8 Example of feasible solution for the RTLBP. 28
Figure 1.9 Example of feasible solution for the RALBP problem. 34
Figure 1.10 Position of the RTLBP and the SDRALBP in the assembly line balancing literature. 45

Figure 2.1 General scheme of H-BFSL algorithm 50
Figure 2.2 General scheme of the proposed matheuristic 58

Figure 3.1 Precedence graph. 71
Figure 3.2 Auxiliary graph 74
Figure 3.3 Constrained shortest path and corresponding solution. 75
Figure 3.4 Precedence graph. 77
Figure 3.5 Auxiliary graph and optimal solution for SDRALBP-1 79
Figure 3.6 Auxiliary graph for SDRALBP-2 82
Figure 3.7 Min-max path not exceeding 3 arcs and corresponding solution. 82

Figure 4.1 Scheme of the split-based metaheuristic 91
Figure 4.2 Split is a decoding procedure, mapping each sequence into a solution of the problem. 92
Figure 4.3 Split versus heuristic decoder 93
Figure 4.4 Tree structure for an instance of 4 operations. The subsequence represented by a node and the sequence represented by a leaf are explicited. 103

List of Tables

Table 1 Problème de graphe sous-jacent et complexité v

Table 1.1 Processing times. 14
Table 1.2 Setup times 19
Table 1.3 Processing times. 34
Table 1.4 Sequence-dependent setup times 40
Table 1.5 Notations 46

Table 3.1 Processing times. 73
Table 3.2 Setup times 73
Table 3.3 Processing times. 77
Table 3.4 Sequence-dependent setup times 78
Table 3.5 Underlying graph problems and complexity 87

Table 5.1 ILP on small low-density instances. 110
Table 5.2 ILP on small medium-density instances. 111
Table 5.3 ILP on small high-density instances. 111
Table 5.4 Performance of heuristics H-BFSL and H-SFBL. 113
Table 5.5 Performance of heuristics H-BFSL and H-SFBL on big-size instances. 114
Table 5.6 Experiments with $\delta = 1$ 117
Table 5.7 Experiments with $\delta = 20$ 117
Table 5.8 Experiments with $\delta = 50$ 118
Table 5.9 Experiments with $\delta = 100$ 118
Table 5.10 Average probability distributions of neighborhood moves after the run of the proposed matheuristic 119
Table 5.11 Comparison between the matheuristic and the genetic algorithm of the literature 120
Table 5.12 Average approximate CPU times on A1-A15 instances (in seconds) 121
Table 5.13 Split-based ILS with different configurations: ILS(100,500) means 100 iterated local searches of 500 iterations each 123
Table 5.14 Split-based ILS with different configurations: ILS(50,500) means 100 iterated local searches of 1'000 iterations each 124

Table 5.15 Performance of split-based metaheuristic vs literature. 125
Table 5.16 Approximate CPU times of the algorithms with different configurations. 126
Table 5.17 Instances with null setup times 129
Table 5.18 Instances with low and high setup times 132
Table 5.19 Underlying graph problems and complexity 137

List of Algorithms

1 Dynamic programming sequencing algorithm 54
2 H-BFSL 56
3 Principle algorithm for insertion move 59
4 Principle algorithm for merger move 60
5 Principle algorithm for split move 61
6 Adaptive simulated annealing algorithm 63
7 Split algorithm for the RTLBP 72
8 Split for SDRALBP-1 76
9 Split for SDRALBP-2 81
10 split for SDRALBP-2 85
11 split-based ILS algorithm 94
12 Algorithm to build a giant sequence respecting precedence constraints 96
13 Pseudo-algorithm to compute a weakly compatible giant sequence . . 98
14 $\widetilde{Split}$ algorithm for the RTLBP 100
15 Algorithm to compute a compatible giant sequence from a weakly compatible giant sequence 101

Introduction

The assembly lines first appeared at the beginning of the 20th century in the factories of Ford Motor. They introduced an important innovation in how to organize the work in a workshop. Indeed, the operations are divided between the operators who are placed along a flow line. The economic interests of assembly lines are considerable, among which, an important increase in productivity. This allowed a rapid development of assembly lines all along the 20th century. These were often used for mass-production of inexpensive products. Even today, with the change in the economic model, assembly lines continue to adapt to new societal and industrial challenges.

Operational researchers and optimization scientists begin to take an interest in assembly lines early since the development of operational research. Salveson (1955) was the first to raise the issue of assembly line balancing, it was then defined as a combinatorial optimization problem by Baybars (1986) under the name: simple assembly line balancing problem (SALBP). This has given rise to several studies on the subject during the last decades. SALBP is concerned with assigning a set of operations linked with precedence constraints to a set of workstations placed along the flow line while minimizing some objective. Many SALBP extensions, that are of industrial relevance, were defined for which many optimization algorithms were suggested. Even though SALBP is now a well-established and well-investigated combinatorial optimization problem, the new industrial challenges give rise to new interesting extensions. Research on line balancing problems remains today a hot topic in operations research. Three extensions of SALBP are studied in the frame of this thesis.

During the last century, industrial revolutions have followed one another. The third industrial revolution was characterized by the use of electronics and IT to automate manufacturing. The fourth industrial revolution saw the rise of cyber-physical systems. A cyber-physical system (CPS) is a system of machines with

software and memories capable of executing programs, part of whose execution is characterized by movement in space, storage and analysis of data.

Nowadays, manufacturers are faced with a highly volatile market bringing a growing variety in demand. The modern manufacturing system should be able to be reconfigured to adapt to the new market demand within low cost and time. In this context, two issues must be tackled:

- The variability in production size: the ability of the production line to increase or decrease production size according to market demand. This property is also known as scalability Koren, Wang, and Gu (2017).
- The variability in the product specifications: the ability of the production line to adapt quickly and cost-effectively to a new product that belongs to the same product family Koren and Shpitalni (2010).

To address this issue, Koren et al. (1999) suggested in late 1990s the novel concept of Reconfigurable Manufacturing System (RMS). They give the following definition: *A Reconfigurable Manufacturing System (RMS) is designed at the outset for rapid change in structure, as well as in hardware and software components, in order to quickly adjust production capacity and functionality within a part family in response to sudden changes in market or in regulatory requirements.*

The RMS stands out from the two previous known production systems/paradigms both in terms of cost and efficiency:

- Dedicated Manufacturing Systems (DMS): DMS are production systems that are designed for mass-production. In this context, the production line is designed for a single product. Any change can require significant time and cost. The machines are rigidly sequenced and the production capacity is expected to be high. DML remain cost effective as long as demand is high and no change in product specifications is expected.
- Flexible Manufacturing Systems (FMS): FMS are basically designed for a variety of products. Different products with different volumes are to be manufactured within the same line. FMS use CNC (computer numerical controlled) machines equipped with a single tool that can change according to the product to be manufactured. FMS require high setup cost and their throughput is low. FMS can however quickly adapt to a new demand despite their relatively high cost.

We assist for years to the growing robotization of production lines. According to the international federation of robotics, a 14% increase is observed each year in terms of operational industrial robot jobs. Robots are replacing increasingly humans in repetitive manufacturing tasks. The World Economic Forum finds out that the total task hours performed by human operators is to drop by 13% by 2022. According to Oxford Economics, 20 million jobs in manufacturing will disappear for the benefit of robots by 2030. Robots offer many benefits in the context of manufacturing:

- Accuracy: the more and more manufacturing tasks require cutting edge precision beyond the reach of a human operator.
- Quality: robots allow better monitoring of quality to help reduce the quantity of waste and inconsistent products.
- Cost: despite higher setup cost, robots give a higher return on investment since their performance is much higher. Besides, robots do not require any training/learning cost.

However, we are less-likely going to see the total replacement of all human operators. In the modern factory, human operators work in collaboration with robots to perform operations with high added value. According to the International Federation of Robotics, the world's 345 million human operators are cooperating with some 1.7 million robots in factories all around to world. The same federation also reports that about 70% of workers see automation as an opportunity rather than a threat for their job.

In the frame of the thesis, we are interested with some assembly lines and corresponding problems that may arise with reconfigurability or robotization.

The first class of problems concerns Reconfigurable Transfer Lines and more precisely the Reconfigurable Transfer Line Balancing Problem (RTLBP). The objective is to study the balancing problem jointly with the sequencing problem arising with the sequence-dependent setup times. This problem has been defined about a decade ago. To the classical characteristics from Assembly Line Balancing Problems, two main characteristics are added:

- parallel workstations: the workstations can be equipped with several machines working in parallel in order to achieve better efficiency.
- industrial constraints: these constraints arise in the context of machining which is the the process of removing material from a blank part so as to obtain the desired shape and dimensions.

Two main arguments can be formulated to situate the RTLBP in the context of RMS:

- workstations can be equipped with multiple machines. This way, the production capacity can be easily monitored by adding or removing a machine in order to meet any change in demand.
- the considered machines are mono-spindle head CNC (Computer Numerical Control) machines equipped with tool magazines. The machines can handle a large set of operations at the cost of sequence-dependent setup times. A new product belonging to the same product family can then easily be processed by the system.

More details justifying the reconfigurability of the considered system can be found in Essafi (2010). We note that the problem is of industrial importance since it has been suggested by a French automotive company, more details could be found in the same reference.

The second class of problems concerns Robotic Assembly Lines where different robots are available to perform the operations. The durations of the operations depend on the type of robot selected for the workstation. Robotic Assembly Lines are situated at the heart of the robotization trend and therefore in Industry 4.0. Two problems are identified: the Robotic Assembly Line Balancing Problem-2 (RALBP-2) and the Sequence-Dependent Robotic Assembly Line Balancing Problem-2 (SDRALBP-2). The RALBP-2 raises the balancing and the (robot) selection decisions to be tackled jointly. The objective of minimizing the cycle time is considered. We consider the problem under two different assumptions:

- Many robots per type: each type of robot can be assigned to multiple workstations at the same time.
- One robot per type: each (type of) robot can be assigned to at most one workstation.

The Sequence-Dependent Robotic Assembly Line Balancing Problem-2 (SDRALBP-2) generalizes the RALBP-2 by considering the sequence-dependent setup times.

According to the problems, two or three decisions are considered: balancing, sequencing and robot selection decisions. To solve a combinatorial optimization problem involving several decisions or sub-problems, a natural and quite intuitive approach would be to solve each subproblem separately one after the other. Each subproblem is based on the solution of the subproblem solved before. Such a technique is known as a sequential approach and has been applied successfully to a wide range of optimization problems. We propose three main kinds of approaches:

- Integrated approaches: all the decisions are simultaneously considered.
- Balance-First Sequence-Last (BFSL) approaches: balancing decision is taken before the other decisions.
- Sequence-First Balance-Last (SFBL) approaches: sequencing decision is taken before the other decisions.

The manuscript is organized in five chapters.

Chapter 1: going from the Simple Assembly line Balancing Problem (SALBP), we define balancing problems in Reconfigurable Transfer Lines and in Robotic Assembly Lines. For each problem, we give its components, a short state of the art and an illustrative example. Mathematical models are proposed. Complexity and lower bounds are also discussed. Research gap and goals of the thesis are given as a conclusion.

Chapter 2: a Balance-First Sequence-Last (BFSL) approach for the RTLBP is suggested. At first, a two-step iterative method piloted by a constraint generation algorithm is described to compute a feasible solution. We prove that this method is a 2-approximation algorithm under some assumption. Then a matheuristic, combining simulated annealing and dynamic programming, is proposed to improve a feasible solution.

Chapter 3: before defining the Sequence-First Balance-Last (SFBL) approach we consider the particular case of a fixed *giant sequence* of operations. The balancing decision is modeled as a routing problem in an acyclic graph. The graphs and the routing problems are detailed for each balancing problem. In most cases, the routing problem and the corresponding balancing problem are solved thanks to a polynomial algorithm that we call *split* (inspired from vehicle routing problems).

Chapter 4: a Sequence-First Balance-Last approach (SFBL) approach for RALBP-2 and SDRALBP-2 is suggested. The split algorithm is used in an hybrid metaheuristic and in a branch and bound algorithm. Neighborhood and perturbation operators, and building of initial solution are detailed for the metaheuristic. Branching, exploration and bounding strategies are given for the branch and bound.

Chapter 5: an experimental study is conducted on literature instances. Exact methods are tested and compared with heuristics on small-size RTLBP instances. Heuristics and metaheuristics are tested on large-size RTLBP instances. Only split-based methods are experimented on RALBP-2 and SDRALBP-2 instances.

In the conclusion, we summarize contributions of this thesis and give perspective research directions.

Chapter 1

Assembly line balancing problems

Contents

1.1 Introduction **8**

1.2 An overview of assembly line balancing problems **9**

1.2.1 Line layouts and products 9

1.2.2 The Simple Assembly Line Balancing problem 12

1.3 Reconfigurable Transfer Lines **16**

1.3.1 Components of the problem 16

1.3.1.1 Sequence-dependent setup times 17

1.3.1.2 Parallel workstations 21

1.3.1.3 Industrial constraints 25

1.3.2 The Reconfigurable Transfer Line Balancing Problem . . 25

1.3.2.1 Illustrative example 27

1.3.2.2 State of the art 28

1.3.2.3 A new mathematical model for the RTLBP . . 29

1.3.2.4 Complexity and lower bounds 32

1.4 Robotic Assembly Lines **32**

1.4.1 The Robotic Assembly Line Balancing Problem 33

1.4.1.1 Illustrative example 34

1.4.1.2 State of the art 35

1.4.1.3 A new mathematical model for RALBP 36

1.4.1.4 Complexity and lower bounds 38

1.4.2 The Sequence-Dependent Robotic Assembly Line Balancing Problem 38

1.4.2.1 Illustrative example 39
1.4.2.2 An adapted mathematical model 39
1.4.2.3 Complexity and lower bounds 42
1.5 Research gap and goals of the thesis **43**

1.1 Introduction

The objective of this chapter is to introduce assembly line balancing problems in the context of Industry 4.0. We distinguish two types of lines: reconfigurable transfer lines and robotic assembly lines.

In the context of a reconfigurable transfer line (RTL), workstations can be equipped with multiple machines. The considered machines are homogeneous mono-spindle head CNC machines that can handle a large set of operations at the cost of sequence-dependent setup times.

In the context of robotic assembly lines (RAL), different heterogonoeus robots are available to perform the operations. The processing times of the operations and the sequence-dependent setup times depend on the type of robot selected for the workstation.

These 2 types of lines lead to the following balancing problems:

- The Reconfigurable Transfer Line Balancing Problem (RTLBP): it subsumes two decisions, namely, assigning the operations to the workstations (balancing decision) and sequencing the operations in each workstation (sequencing decision).
- The Robotic Assembly Line Balancing Problem (RALBP). It subsumes the decision of assigning the robots to the workstations (selection decision) in addition to the balancing decision.
- The Sequence-Dependent Robotic Assembly Line Balancing Problem (SDRALBP). It is an extension of the Robotic Assembly Line Balancing Problem where sequence-dependent setup times are considered. The sequencing decision is raised in addition to the balancing and selection decisions.

Since we deal with assembly line balancing problems, we first give a short taxonomy of these problems in the next section. Then, the RTLBP, the RALBP and the SDRALBP are detailed and situated in relation to the Simple Assembly Line Balancing Problem (SALBP) and some of its relevant extensions. For each of these problems, we give an illustrative example and a mathematical model. Novel mathematical models for the RTLBP and the RALBP are suggested. Complexity and lower bounds are discussed. Research gaps and goals of the thesis are given in the last section of the chapter.

1.2 An overview of assembly line balancing problems

Assembly lines are typically composed of a series of workstations connected by a material handling system. The product is processed in each workstation, then is moved to the next workstation thanks to a conveyor. A set of assembly operations is assigned to each workstation. The workload of some workstation designates the sum of the processing times of the operations assigned to it. The cycle time is often used as a KPI (Key Performance Indicator) of the assembly line, it represents the largest workload among all workstations. In other words, the cycle time is the duration separating the exit of two assembled products from the last workstation of the assembly line. The smaller the cycle time is, the more products are processed by the line. For this reason, the cycle time is considered as an indicator of efficiency. On the other hand, the higher the number of workstations is, the more expensive the line is to operate. The number of workstations is an indicator of cost. Balancing an assembly line refers to the decision problem of assigning the operations to the workstations. In practice, some constraints must be respected. Precedence constraints are frequently encountered. They link pairs of operations such that one operation must precede another, which means that it must be assigned either to a workstation placed before on the line or to the same workstation.

The objective of this subsection is not to give an exhaustive classification of assembly line balancing problems. Such a study can be found in Battaïa and Dolgui (2013). We only highlight on some classic characteristics of assembly lines that may affect the balancing problem.

1.2.1 Line layouts and products

Different assembly line layouts exist:

- Straight line: also called I-line. The workstations of the assembly line are placed one after the other within a serial straight line. The product being

assembled is processed by each workstation in the order it is placed in the line. A straight assembly line is represented in Figure 1.1. Operations 2,1 and 3 are performed in the first workstation, operations 4 and 5 are performed in the second workstation and operations 7 and 6 are performed in the third workstation.

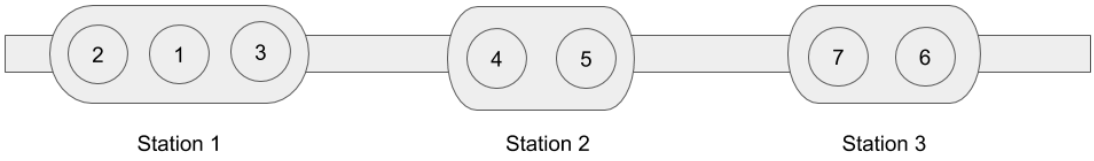

Figure 1.1: Straight assembly line.

- U-line: such assembly lines are arranged within the form of a "U". Workstations facing each other often share resources like human operators or robots: Miltenburg and Wijngaard (1994), Şahin and Kellegöz (2017). A U-assembly line is represented in Figure 1.2. The assembly product goes through workstation 1 (operations 2 and 1) then workstation 2 (operations 4 and 5) then returns to workstation 2 (operations 3 and 6) and finally workstation 1 (operation 7).

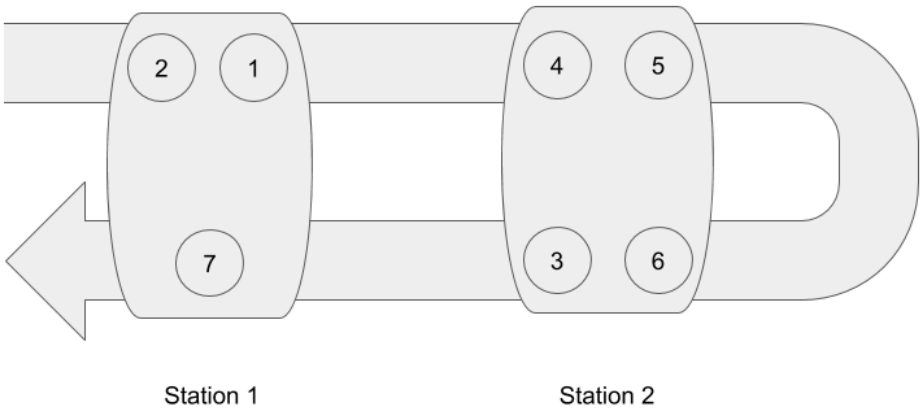

Figure 1.2: U-assembly line.

- Parallel workstations: In this context, some workstations are enhanced by duplicating the resources whether they are robots or human operators. Each time the resources are duplicated, the workload is divided by two: Vilarinho and Simaria (2002), Tiacci (2017). In Figure 1.3, the second workstation is duplicated.

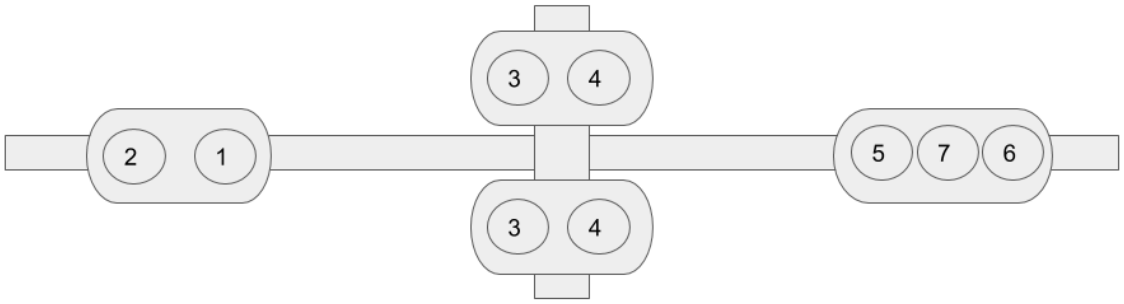

Figure 1.3: Parallel workstations.

- Parallel lines: Two or more lines could be parallelized to enhance efficiency. We call it crossover when two lines share the same workstation. The resources are placed between the lines, they are used either on a single line or alternately on one and the other. A product type is assigned to each line. Crossover can be useful for optimizing resource usage: Gökçen, Ağpak, and Benzer (2006), Özcan (2019).

We can distinguish three types of assembly line balancing problems according to the number of products being assembled within the line (Figure 1.4):

- Single model: A single type of product is considered.
- Multi model: Multiple products are considered. The products are processed sequentially: a number of required items of some product a is assembled then the line is setup to process another product b (van Zante-de Fokkert and de Kok (1997), Boysen, Fliedner, and Scholl (2008)).
- Mixed model: Multiple products are also considered but within the same product mix. Mixed model requires high similitude between products in order to process them within the same product mix: Merengo, Nava, and Pozzetti (1999), Vilarinho and Simaria (2006), Kara et al. (2011).

Three classes of problems can be derived based on the characteristics of the processing times:

- Deterministic times: processing times are known and given input of the problem.
- Stochastic times: processing times follow a stochastic distribution Sotskov, Dolgui, and Portmann (2006), Dolgui and Kovalev (2012).
- Dynamic times: processing times can change during the assembly process Cohen and Ezey Dar-El (1998), Toksarı et al. (2010).

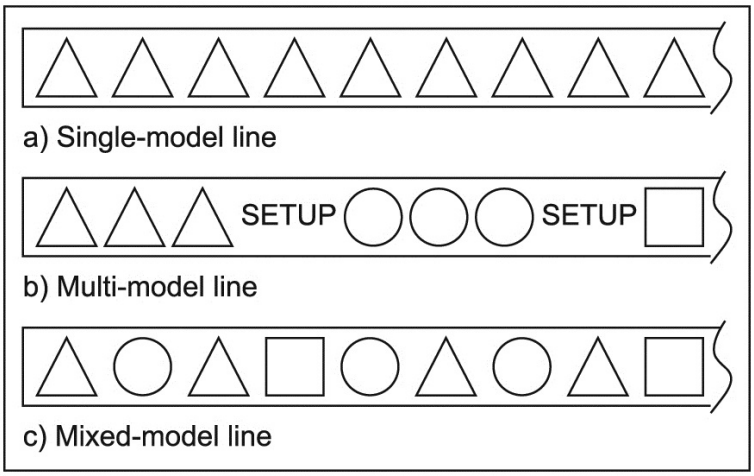

Figure 1.4: Single-model, multi-model and mixed-model lines (Becker and Scholl (2006))

We consider in the remainder of the thesis single model and deterministic times.

1.2.2 The Simple Assembly Line Balancing problem

The Simple Assembly Line Balancing Problem (SALBP) is a well-established combinatorial optimization problem. It was defined by Salveson (1955).

In the context of SALBP, we are given a set of operations each of which has a processing time and a set of workstations that are organized within a straight line. A set of precedence constraints is also given. It consists of a set of couples of operations (i, j) such that the workstation to which the operation i is assigned is not placed after the one of j. Precedence constraints are often represented by a precedence graph (Scholl and Klein (1999)).

SALBP consists in assigning each operation to a workstation while respecting precedence constraints. Several objectives are considered in literature:

- SALBP-1 is concerned with minimizing the number of workstations used while respecting the so-called cycle time constraint: for each workstation, the sum of the processing times of the operations assigned to this workstation does not exceed a given cycle time.
- SALBP-2 is about minimizing the cycle time (maximum workload among the workstations) while using no more than a given number of workstations.
- SALBP-E is concerned with minimizing the product of the cycle time and the number of workstations.
- SALBP-F is concerned with finding a feasible solution with given cycle time and maximum number of workstations.

The SALBP has been extensively studied during the last decades. It has been tackled with different kinds of methods: Integer Linear Programming, Metaheuristics, Branch and Bound...

The website: `https://assembly-line-balancing.de` contains all the well-known benchmark instances of the problem.

The branch and bound algorithm suggested in Sewell and Jacobson (2012), Morrison, Sewell, and Jacobson (2014) is a major breakthrough in the assembly line balancing literature. Indeed, the method is very efficient. It has solved some benchmark instances that has remained open for years. The method is developed for SALBP-1. The authors introduce a new operations research' method called "Branch and Bound and Remember". The suggested method has the usual structure of a branch and bound algorithm enhanced with some memory-based dominance rules (and remember). A node is pruned if it is dominated by a node that has already been processed. The method requires delicate management of memory since all the processed nodes are stored. Dolgui and Gafarov (2019) present some instances that cannot be solved by the method efficiently.

SALBP remains until today a hot topic in operations research on which many researchers are working around the world.

The SALBP-1 is stated as follows:

- **Data:**
  - A set N of n operations: $N = \{1, 2, \ldots, n\}$ and their processing times such that d_i denotes the processing time of operation i.
  - A set P of precedence relations: $(i, j) \in P$ means that i must be performed before j in the line.
  - A set S of s_{max} workstations: $S = \{1, 2, \ldots, s_{max}\}$. s_{max} is an upper bound of the number of workstations (for example n).
  - A cycle time C.
- **Decision:**
  - Balancing decision: assign the operations to the workstations.
- **Constraints:**
  - Respect precedence relations.
  - Respect the cycle time constraint: the sum of the processing times of the operations assigned to each workstation must not exceed the cycle time C.

- **Objectives:**
  - Minimize the number of workstations.

Illustrative example

In order to help the understanding, we illustrate the SALBP-1 by an instance:

- A product requires the execution of 7 operations numbered from 1 to 7 ($n = 7$).
- Precedence relations are represented in Figure 1.5.

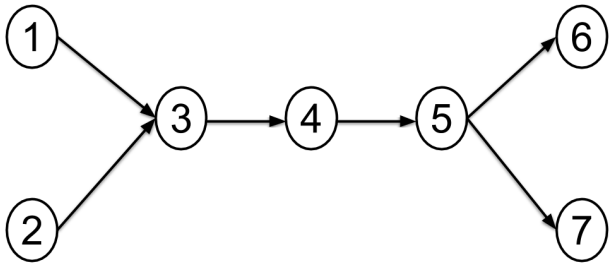

Figure 1.5: Precedence graph.

- Processing times are represented in Table 1.1.

i	1	2	3	4	5	6	7
d_i	1.5	1	3.5	1.5	2.5	3	1

Table 1.1: Processing times.

- A cycle time is given: $C = 9$

The solution depicted in Figure 1.6 represents a solution with three workstations. Operations 2 and 1 are assigned to workstation 1, operations 3 and 4 are assigned to workstation 2 and operations 5, 7 and 6 are assigned to workstation 3. The workloads are calculated as follows:

- Workload of workstation 1: $w_1 = d_2 + d_1 = 2.5$.
- Workload of workstation 2: $w_2 = d_3 + d_4 = 5$.
- Workload of workstation 3: $w_3 = d_5 + d_7 + d_6 = 6.5$.

The cycle time is respected since $w_1, w_2, w_3 \leq 9$, the number of workstations is 3.

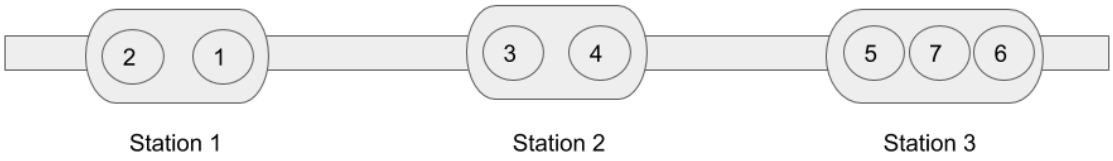

Figure 1.6: Example of feasible solution.

Mathematical model

A widespread and quite intuitive integer linear program for the SALBP-1 is that proposed by Baybars (1986), it uses the following binary variables:

$$x_{i,s} = \begin{cases} 1 & \text{If operation } i \text{ is assigned to workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$$y_s = \begin{cases} 1 & \text{If at least one operation is assigned to workstation } s \text{ (workstation } s \text{ is open).} \\ 0 & \text{Otherwise.} \end{cases}$$

The objective of minimizing the number of workstations can be formulated as follows:

$$Min \sum_{s=1}^{s_{max}} y_s$$

The constraints of the problem can be formulated as follows:

$$\sum_{s=1}^{s_{max}} x_{i,s} = 1, \forall i \in N \tag{1.1}$$

$$y_{s+1} \leq y_s, \forall s \in \{1, \ldots, s_{max} - 1\} \tag{1.2}$$

$$\sum_{s=1}^{s_{max}} s.x_{i,s} \leq \sum_{s=1}^{s_{max}} s.x_{i',s}, \forall (i, i') \in P \tag{1.3}$$

$$\sum_{i=1}^{n} d_i.x_{i,s} \leq C.y_s, \forall s \in S \tag{1.4}$$

The set of constraints (1.1) ensures that each operation is assigned to exactly one workstation, these are known as occurrence constraints. The set of constraints (1.2) ensures that no new workstation is opened unless all the preceding workstations are open. The set of constraints (1.3) represents the precedence constraints. The set of constraints (1.4) represents the cycle time constraints.

The previous model can easily be adapted for SALBP-2 by replacing the objective with: $Min\ C$.

As mentioned in Baybars (1986), some useless variables can be deleted from the model by computing:

- $e(i)$: A lower bound of the index of the workstation to which operation i can be assigned.
- $l(i)$: An upper bound of the index of the workstation to which operation i can be assigned.

The calculation of $e(i)$ and $l(i)$ is based on the precedence constraints and the cycle time. They are computed as follows:

$$e(i) = \left\lceil \frac{\sum_{i \in P_i} d_i}{C} \right\rceil$$

$$l(i) = s_{max} - \left\lceil \frac{\sum_{i \in S_i} d_i}{C} \right\rceil + 1$$

such that $P_i = \{i\} \cup \{j \in N; (j,i) \in P\}$ (i and its predecessors) and $S_i = \{i\} \cup \{j \in N; (i,j) \in P\}$ (i and its successors). In this way, all the variables $x_{i,s}$ such that $s \notin [e(i), l(i)]$ are set to 0.

1.3 Reconfigurable Transfer Lines

The particularity of the RTLBP is to consider simultaneously parallel machines, setup times and transfer line environment constraints. These three components are most often studied separately in literature. We first define each element independently and describe the related balancing problems by illustrating them with an example. Then we give an illustrative example for the RTLBP and a quick state of the art about reconfigurable transfer lines balancing. Finally, a novel mathematical model is suggested.

1.3.1 Components of the problem

We detail in this subsection sequence-dependent setup times, parallel workstations and industrial constraints that arise in the machining industry.

1.3.1.1 Sequence-dependent setup times

Sequence-dependent setups are encountered in many workshop problems (Allahverdi et al. (2008)). Setups are necessary to provide for tool change or product handling that can occur between two operations. Sequence-dependent setups in the context of assembly line were only considered recently (Andres, Miralles, and Pastor (2008)). Scholl, Boysen, and Fliedner (2013) cite some interesting situations where sequence-dependent setup times are encountered:

- In many industries where large products are manufactured, the operations are performed on different positions of the workpiece. This may require the operator to move between the positions where the consecutive operations are performed. Scholl, Boysen, and Fliedner (2013) reports that this walking distance can go up to 10 meters in the context of automotive industry. Alongside with the walking distances of the operators, time to withdraw a part from a container may be required. Scholl reports that for a major German car manufacturer, this walking and withdrawal time takes about 15% of the cycle time.
- Machine-tools often use a single tool at a time. Setup times depending on the sequence of operations are required to change the tool if two consecutive operations require different tools.
- In situations where operations require the workpiece to be put in a specific position, setup is needed for handling the piece between two consecutive operations requiring different positions.
- In many specific situations, a delay must be considered between two operations (for example after gluing or painting).

These last elements justify that the setup times cannot be neglected. Besides, the setup times very often depend on the sequence of operations.

Andres, Miralles, and Pastor (2008) define the Sequence-dependent simple assembly line balancing problem (SDSALBP). SDSALBP raises the decision of sequencing the operations in each workstation in addition to the balancing decision. The two decisions are to be addressed jointly.

Scholl, Boysen, and Fliedner (2013) argues that the sequence-dependent setup times must be differentiated between *forward* setup times and *backward* setup times. A forward setup time $t_{i,j}$ is considered when operation i is performed just before operation j in the same workstation within the same cycle. If operations i and j are respectively the last and the first operations assigned to some workstation, a

backward setup time $t_{i,j}$ is considered. The problem is very slightly different from SDSALBP and is called SUALBSP (Setup Assembly Line Balancing and Scheduling Problem). Even if we think that distinguishing between backward and forward setups may be realistic from a real world perspective, the latter is not made in the frame of the thesis for the sake of simplification.

Many recent studies consider sequence-dependent setup times in the context of assembly lines: Essafi, Delorme, and Dolgui (2010b), Martino and Pastor (2010), Janardhanan et al. (2019).

The Sequence-Dependent Simple Assembly Line Balancing Problem:

Compared to the SALBP-1, SDSALBP-1 has an additional data: the sequence-dependent setup times. They take the form of a matrix of size $n \times n$: $(t_{i,j})$ where $t_{i,j}$ is the setup time to be considered when operation i is performed just before operation j in some workstation. The calculation of the workloads of the workstations is slightly different since the induced setup times must be added to the processing times of the operations.

The SDSALBP-1 is stated as follows:

- **Data:**
  - As for the SALBP-1, a set N of n operations: $N = \{1, 2, \ldots, n\}$, their processing times (d_i), a set P of precedence relations, a set S of s_{max} workstations: $S = \{1, 2, \ldots, s_{max}\}$ and a given cycle time C.
  - The sequence-dependent setup times $(t_{i,j})$.
- **Decisions:**
  - Balancing decision: assign the operations to the workstations.
  - Sequencing decision: sequence the operations in each workstation.
- **Constraints:**
  - Respect precedence relations.
  - Respect the cycle time constraint: the sum of the processing times of the operations assigned to each workstation and the induced sequence-dependent setup times must not exceed the given cycle time C.
- **Objectives:**
  - Minimize the number of workstations.

Illustrative example

We illustrate the SDSALBP on a small example derived from the previous SALBP example (introduced in the previous section). Sequence-dependent setup times are shown in Table 1.2.

$t_{i,j}$	$j=1$	2	3	4	5	6	7
$i=1$	0	0.5	1	1	1	1	1
2	1	0	0.5	1	1	1	1
3	0.5	1	0	1	1	1	1
4	1	1	1	0	0.5	1	1
5	1	1	1	0.5	0	1	1
6	1	1	1	1	1	0	0.5
7	1	1	1	1	1.5	0.5	0

Table 1.2: Setup times

The workload of the workstations are calculated as follows for the solution shown in Figure 1.6:

- Workload of workstation 1: $w_1 = d_2 + t_{2,1} + d_1 + t_{1,2} = 4$.
- Workload of workstation 2: $w_2 = d_3 + t_{3,4} + d_4 + t_{4,3} = 7$.
- Workload of workstation 3: $w_3 = d_5 + t_{5,7} + d_7 + t_{7,6} + d_6 + t_{6,5} = 9$.

The cycle time is respected since $w_1, w_2, w_3 \leq 9$. We notice a non-negligible increase in the workloads of the workstations due to the consideration of sequence-dependent setup times. In workstation 3, a different sequence of operations, for example $(7, 5, 6)$, leads to a different workload, i.e. 9.5, which exceeds the cycle time.

Mathematical model

The following formulation of SDSALBP-1 is due to Andres, Miralles, and Pastor (2008). Compared to the formulation of the SALBP, Andres, Miralles, and Pastor (2008) add an additional dimension j to the variable $x_{i,s}$ in order to decide in which position (j) of the sequence of workstation s the operation i is assigned. Additional variables are also needed to account for sequence-dependent setup times.

The formulation uses the following binary variables:

$$x_{i,s,j} = \begin{cases} 1 & \text{If operation } i \text{ is assigned to workstation } s \text{ at the } j^{th} \text{ position of its sequence.} \\ 0 & \text{Otherwise.} \end{cases}$$

$$y_s = \begin{cases} 1 & \text{If at least one operation is assigned to workstation } s \\ 0 & \text{Otherwise.} \end{cases}$$

$$z_{i,i',s} = \begin{cases} 1 & \text{If operation } i \text{ is processed just before operation } i' \text{ at workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$$w_{i,s} = \begin{cases} 1 & \text{If operation } i \text{ is assigned to the last position of the sequence at workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

The objective of minimizing the number of workstations is considered:

$$Min \sum_{s=1}^{s_{max}} y_s$$

The constraints of SDSALBP are formulated as follows: (1.5-1.14).

$$\sum_{s=1}^{s_{max}} \sum_{j=1}^{n} x_{i,s,j} = 1, \forall i \in N \tag{1.5}$$

$$\sum_{i=1}^{n} x_{i,s,j} \leq 1, \forall s \in S, \forall j \in \{1, \ldots, n\} \tag{1.6}$$

$$\sum_{i=1}^{n} x_{i,s,j+1} \leq \sum_{i=1}^{n} x_{i,s,j}, \forall s \in S, \forall j \in \{1, \ldots, n-1\} \tag{1.7}$$

$$y_{s+1} \leq y_s, \forall s \in \{1, \ldots, s_{max} - 1\} \tag{1.8}$$

$$\sum_{s=1}^{s_{max}} \sum_{j=1}^{n} (n.(s-1)+j).x_{i,s,j} \leq \sum_{s=1}^{s_{max}} \sum_{j=1}^{n} (n.(s-1)+j).x_{i',s,j}, \forall (i,i') \in P \tag{1.9}$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} d_i.x_{i,s,j} + \sum_{i=1}^{n} \sum_{i'=1}^{n} t_{i,i'}.z_{i,i',s} \leq C.y_s, \forall s \in S \tag{1.10}$$

$$x_{i,s,j} + x_{i',s,j+1} \leq 1 + z_{i,i',s}, \forall (i,i') \in N^2, i \neq i', \forall j \in \{1, \ldots, n-1\}, \forall s \in S \tag{1.11}$$

$$x_{i,s,j} - \sum_{i' \in N; i' \neq i} x_{i',s,j+1} \leq w_{i,s}, \forall i \in N, \forall s \in S, \forall j \in \{1, \ldots, n-1\} \tag{1.12}$$

$$x_{i,s,n} \leq w_{i,s}, \forall i \in N, \forall s \in S \tag{1.13}$$

$$w_{i,s} + x_{i',s,1} \leq 1 + z_{i,i',s}, \forall (i,i') \in N^2, i \neq i', \forall s \in S \tag{1.14}$$

The set of constraints (1.5) ensures that each operation is assigned to exactly one workstation at a unique position of its sequence. The set of constraints (1.6) ensures that at each workstation at most one operation is assigned in each position of the sequence. The set of constraints (1.7) ensures that at each workstation no operation is assigned at a position $j + 1$ unless some operation is assigned at the position j. The set of constraints (1.8) ensures that no workstation is used unless its precedent workstation is also used. The set of constraints (1.9) ensures that precedence constraints are satisfied. The set of constraints (1.10) ensures that the cycle time is not exceeded at any workstation. The set of constraints (1.11) ensures that if operation i is followed by operation i' at workstation s then $z_{i,i',s}$ is set to 1. Constraints (1.12) and (1.13) ensure that $w_{i,s}$ is set to 1 whenever operation i is positioned at the last occupied position in the sequence of workstation s. The set of constraints (1.14) ensures that if operation i is positioned at the last occupied position in the sequence of workstation s and operation i' is positioned at the first position in the sequence of workstation s then $z_{i,i',s} = 1$ and consequently the setup time $t_{i,i'}$ is considered in (1.10).

1.3.1.2 Parallel workstations

In some situations, it is not possible to respect the cycle time constraint due to some operations' high processing times. Besides, the limitation in terms of workstations can make it impossible to reach a cycle time that is satisfactory from the decision-maker perspective. For the two previous reasons, Buxey (1974) introduced the Simple Assembly Line Balancing Problem with Parallel Workstations (SALBPPW). In the latter, paralleling workstations is adopted to achieve better cycle times. In this context, the layout of the line is organized within stages (Bukchin and Rubinovitz (2003)). Each stage can contain several workstations. The workstations within the same stage perform the same set of operations and operate on different units of the same product. This way, we can define the local cycle time of some stage as the workload of the workstations that constitute it divided by the number of workstations in the stage. The cycle time of the line is then the maximum cycle time among the stages. Having the ability to add a workstation in some stage reduces the local cycle time and therefore the cycle time of the line if the considered stage is bottleneck.

Wang and Koren (2012) and Koren, Wang, and Gu (2017) highlight on the benefits of parallel workstations in terms of reconfigurability. Indeed, they consider a Reconfigurable Manufacturing System (RMS) which takes the form of flow line (for example assembly line) composed of stages. Stages can contain several machines operating in parallel. The studies show that having the possibility to add/remove a machine in some workstation allows the decision-maker to monitor the production capacity (cycle time in the case of assembly lines) with higher granularity. The latter

allows to accommodate effectively with the changes in terms of market demand.

The Simple Assembly Line Balancing Problem with Parallel Workstations:

In the context of SALBPPW, the balancing decision stands for assigning the operations to the stages. Compared to the SALBP, SALBPPW introduces a new decision to address: the number of workstations in each stage. An additional data is often considered: the maximum number of workstations per stage (Bukchin and Rubinovitz (2003)).

The SALBPPW-1 is stated as follows (Bukchin and Rubinovitz (2003)):

- **Data:**
  - As for the SALBP, a set N of n operations: $N = \{1, 2, \ldots, n\}$, their processing times (d_i) and a set P of precedence relations.
  - A set S of s_{max} stages: $S = \{1, 2, \ldots, s_{max}\}$.
  - M_{max}: the maximum number of workstations in a stage.
  - A cycle time C.
- **Decisions:**
  - Balancing decision: assign the operations to the stages.
- **constraints:**
  - Respect precedence constraints.
  - Respect the maximum number of stages.
  - Respect the maximum number of workstations per stage.
  - Respect the cycle time constraint in each stage: the sum of the processing times of the operations assigned to the stage divided by the number of workstations hosted by the stage must not exceed the cycle time.
- **Objectives:**

  Different objectives can be considered for the SALBPPW:
  - Minimize the number of workstations or the number of stages or any aggregated function of both.

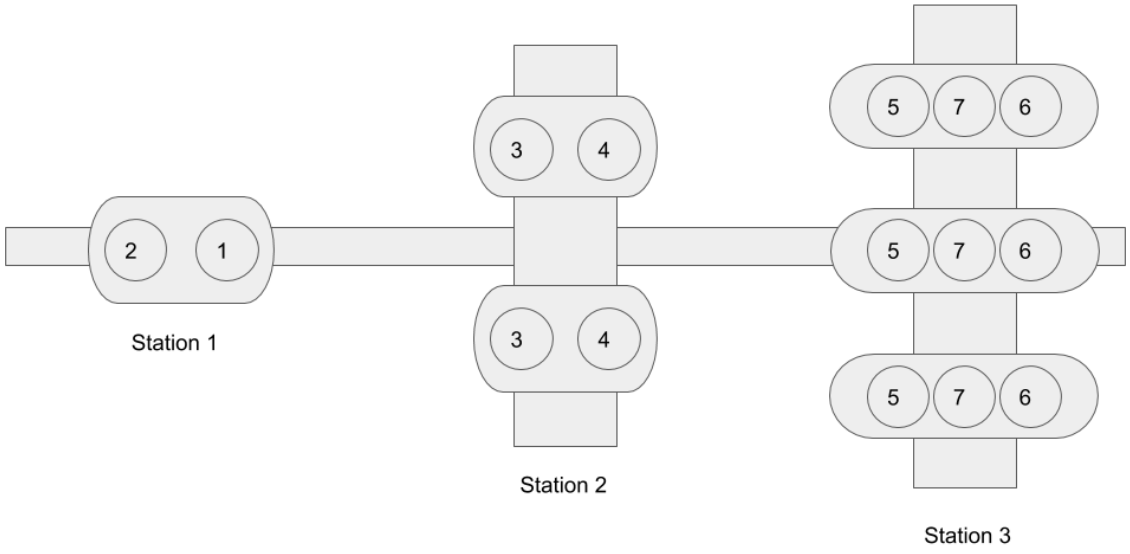

Figure 1.7: Example of feasible solution.

Illustrative example

We illustrate the SALBPPW-1 on a small example derived from the previous SALBP example (introduced in subsection 1.2.2).

For this example, we consider a fixed cycle time: $C = 3$. We note that to comply with the cycle time constraint, we have to consider more than one workstation at the stage where operation 3 is assigned. Indeed, the processing time of operation 3 is 3.5 which exceeds the cycle time.

Figure 1.7 illustrates a feasible solution. The solution has three stages which contain respectively one, two and three workstations.

The cycle times are calculated as follows:

- Cycle time of stage 1: $C_1 = d_2 + d_1 = 2.5$.
- Cycle time of stage 2: $C_2 = (d_3 + d_4)/2 = 2.5$.
- Cycle time of stage 3: $C_3 = (d_5 + d_7 + d_6)/3 \simeq 2.16$.

There are two bottleneck stages (stages 1 and 2). The cycle time constraint is respected since: $C_1, C_2, C_3 \leq C$.

Mathematical formulation

We present a mathematical model for the SALBPPW. The model is due to Bukchin and Rubinovitz (2003), it uses the following binary variables:

$$x_{i,s} = \begin{cases} 1 & \text{If operation } i \text{ is assigned to stage } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$$v_{s,k} = \begin{cases} 1 & \text{If there are } k \text{ workstations at stage } s. \\ 0 & \text{Otherwise.} \end{cases}$$

The objective of minimizing the number of workstations can be formulated as follows:

$$Min \sum_{s=1}^{s_{max}} \sum_{k=1}^{M_{max}} k.v_{s,k}$$

The constraints of the problem can be formulated as follows:

$$\sum_{s=1}^{s_{max}} x_{i,s} = 1, \forall i \in N \tag{1.15}$$

$$\sum_{k=1}^{M_{max}} v_{s,k} \leq 1, \forall s \in S \tag{1.16}$$

$$\sum_{s=1}^{s_{max}} s.x_{i,s} \leq \sum_{s=1}^{s_{max}} s.x_{i',s}, \forall (i,i') \in P \tag{1.17}$$

$$\sum_{i=1}^{n} d_i.x_{i,s} \leq C. \sum_{k=1}^{M_{max}} k.v_{s,k}, \forall s \in S \tag{1.18}$$

The constraints (1.15) ensure that each operation is assigned to exactly one stage. Constraints (1.16) ensure that at most a number of workstations is assigned to each stage. Constraints (1.17) represent the precedence constraints. Constraints (1.18) represent the cycle time constraints.

The model can easily be adapted for the objective of minimizing the cycle time. In this situation, it is relevant to add a variable y_s to decide whether to assign at least one operation to some stage s and a constraint $y_{s+1} \leq y_s$ to avoid empty stages at the middle of the solution. In this case, constraints (1.16) would be replaced by $\sum_{k=1}^{M_{max}} v_{s,k} \leq y_s, \forall s \in S$.

In the remainder of the thesis (and as it is usually done), we won't talk anymore about stages. We consider that a stage is a workstation. A stage with multiple workstations represents a workstation with multiple machines.

1.3.1.3 Industrial constraints

The RTLBP supports some additional constraints which are specific to the machining industry: inclusion, exclusion and accessibility constraints. Balancing problems considering those constraints are known as transfer line balancing problems. Firstly introduced in Dolgui, Guschinsky, and Levin (2000), some other authors consider those constraints like Battaïa et al. (2012). More references can be found in Battaïa and Dolgui (2013).

- Inclusion constraints link between two operations that must be assigned to the same workstation. As for precedence constraints, they are given in the form of set of couples that we denote I.

- Exclusion constraints consist in subsets (called exclusion subsets) of operations that must not be assigned all to the same workstation. This means that at least one operation is assigned to another workstation. We denote by E the set of all exclusion subsets. For example, suppose we have 4 operations denoted o_1, o_2, o_3, o_4 and an exclusion set $\{o_1, o_2, o_3\}$. In this case, it is not acceptable to have o_1, o_2 and o_3 all assigned to the same workstation. However, it is possible to have o_1 and o_2 assigned to the same workstation provided o_3 is assigned to a different workstation.

- Accessibility constraints are also considered. Each operation o_i has a subset Pos_i of possible part-fixing positions. An accessibility constraint is related to a workstation. It imposes that all the operations assigned to the same workstation must have at least one common part-fixing position. For example, suppose we have 4 operations denoted o_1, o_2, o_3, o_4 and 3 possible part-fixing positions $Pos = \{1, 2, 3\}$ such that:

$$Pos_{o_1} = \{1, 2\}, Pos_{o_2} = \{1, 2, 3\}, Pos_{o_3} = \{2, 3\}, Pos_{o_4} = \{3\}$$

  Operations $\{o_1, o_2, o_3\}$ could be assigned to the same workstation because the position 2 is shared by o_1, o_2 and o_3. However operations $\{o_1, o_2, o_4\}$ cannot be assigned to the same workstation because they share no position.

1.3.2 The Reconfigurable Transfer Line Balancing Problem

In the context of RTLBP, we are given a set of operations and a serial straight line of workstations. Each workstation is equipped with multiple identical machines working in parallel. Sequence-dependent setup times are considered. Each product goes through all the workstations of the line in the order. Each product is processed by only one machine at each workstation. At each cycle time, a new product arrives on the line on the first workstation, a processed product leaves the line from the last workstation and in each workstation a product leaves to the next workstation

and is replaced by a product coming from the precedent workstation. When a workstation is equipped with k machines, the product remains $k \times$ cycle time on a machine. The machines within the same workstation perform the same sequence of operations. A maximum number of machines per workstation and a maximum number of operations per workstation is considered.

The RTLBP is stated as follows:

- **Data:**
  - As for the SDSALBP-1, a set N of n operations: $N = \{1, 2, \ldots, n\}$, their processing times (d_i), a set P of precedence relations, a set S of s_{max} workstations: $S = \{1, 2, \ldots, s_{max}\}$ and the sequence-dependent setup times ($t_{i,j}$).
  - M_{max} and N_{max}, respectively the maximum number of machines and the maximum number of operations in a workstation.
  - s_{max}, maximum number of workstations.
  - C, a given cycle time.
  - I, a set of couples $(i, j) \in N \times N$ linked with an inclusion constraint.
  - E, a set of all subsets of operations that cannot be assigned to the same workstation.
  - Pos, a set of all possible part-fixing positions. Pos_i, the subset of possible part-fixing positions for operation i.
- **Decisions:**
  - Balancing decision: assign the operations to the workstations.
  - Sequencing decision: sequence the operations in each workstation:
- **Constraints:**
  - Precedence, inclusion, exclusion and accessibility constraints must be respected.
  - The cycle time constraint must be respected. For each workstation, the workload (the sum of the processing times and the setup times induced by the sequence allocated to the workstation) divided by the number of machines allocated to the workstation must not exceed the given cycle time.

  - The maximum number of machines per station, the maximum number of operations per station and the maximum number of workstations must be respected.

- **Objectives:**

  - Minimize the number of machines.

1.3.2.1 Illustrative example

In order to help the understanding, we illustrate the RTLBP thanks to a small example. The example is derived from the SDSALBP-1 example (subsection 1.3.1):

- At most 5 workstations can be used ($s_{max} = 5$).
- $N_{max} = 3$, maximum number of operations to be assigned to a workstation.
- $M_{max} = 3$, maximum number of machines to be hosted by a workstation.
- $C = 2.5$, cycle time.
- Inclusion and exclusion constraints are given by $I = \{(1, 2)\}, E = \{\{5, 6\}\}$.
- Accessibility constraints are given as follows: $Pos = \{1, 2, 3, 4\}$, $Pos_4 = \{1, 2\}$, $Pos_5 = \{3, 4\}$, $Pos_i = \{1, 2, 3, 4\}, \forall i \in \{1, 2, 3, 6, 7\}$

A feasible solution is represented in Figure 1.8. This solution uses 5 workstations:

- The sequence of operations in workstation 1 is $(1, 2)$: the workload is $d_1 + t_{1,2} + d_2 + t_{2,1} = 4$. The number of machines used is: $\lceil \frac{d_1+t_{1,2}+d_2+t_{2,1}}{C} \rceil = \lceil \frac{4}{2.5} \rceil = 2$.
- The sequence of operations in workstation 2 is (3): the workload is $d_3 = 3.5$. The number of machines used is: $\lceil \frac{3.5}{2.5} \rceil = 2$.
- The sequence of operations in workstation 3 is (4): the workload is $d_1 = 1.5$. The number of machines used is: $\lceil \frac{1.5}{2.5} \rceil = 1$.
- The sequence of operations in workstation 4 is (5): the workload is $d_5 = 2.5$. The number of machines used is: $\lceil \frac{2.5}{2.5} \rceil = 1$.
- The sequence of operations in workstation 5 is $(6, 7)$: the workload is $d_6 + t_{6,7} + d_7 + t_{7,6} = 5$. The number of machines used is: $\lceil \frac{5}{2.5} \rceil = 2$.

The number of machines is 8.

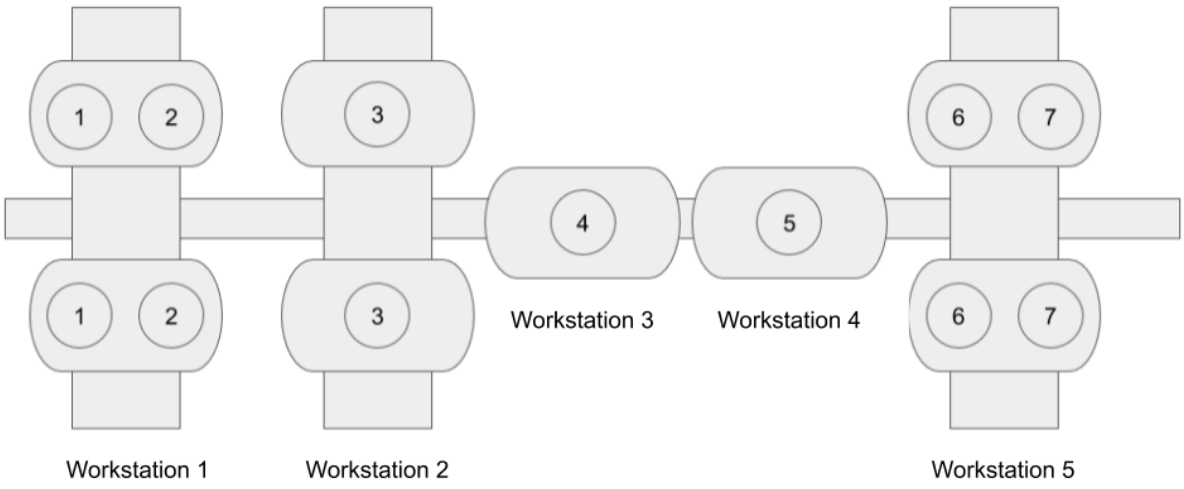

Figure 1.8: Example of feasible solution for the RTLBP.

1.3.2.2 State of the art

The particularity of the RTLBP is to consider simultaneously parallel machines, setup times and transfer line environment constraints. These three components are most often studied separately in literature. Few papers deal with the RTLBP, it has been first introduced in Essafi, Delorme, and Dolgui (2010b) where a MIP approach is suggested. The considered objective is to minimize the number of machines while respecting a given cycle time.

In Essafi et al. (2010), an ant colony optimization algorithm is considered: every ant creates a workstation and adds operations to it as long as it is possible. The ants should decide which operation to add to the current workstation. This is done thanks to the probability distribution given by pheromone trails.

In Essafi, Delorme, and Dolgui (2010a), a greedy construction algorithm is suggested to give a solution. There is no guaranty however that the algorithm will give a feasible solution. Its idea is to try to add operations from a list of candidate operations to the current workstation as long as it is possible. Once the workstation is full, another workstation is created, and the process is repeated until all operations are assigned. Since the greedy algorithm could output a solution that exceeds the number of workstations authorized, a second phase is developed to merge workstations if possible.

In Essafi, Delorme, and Dolgui (2012), a GRASP approach is introduced. Starting solutions are generated using the greedy random heuristic of Essafi, Delorme, and Dolgui (2010a) then a local search is performed using 4 neighborhood moves:

- Move a subset of operations from a workstation to another.
- Merge one or more workstations.
- Move operations from the most loaded workstations to the less loaded workstations.
- Optimize the subsequence of operations in each workstation (using heuristics and a MIP).

In Borisovsky, Delorme, and Dolgui (2013), a genetic algorithm is suggested. A chromosome is coded as a giant sequence of all operations and a heuristic decoder is suggested to build a solution. Classical crossover moves for the TSP are used as well as the following mutation moves:

- Swap 2 operations.
- Insert an operation into a different position.

An exact method is suggested in Borisovsky, Delorme, and Dolgui (2014): it uses a set partitioning model coupled with a constraint generation algorithm.

1.3.2.3 A new mathematical model for the RTLBP

The ILP of SDSALBP does not stand for the RTLBP due to the additional constraints and the possibility of having multiple machines in each workstation. We suggest a new formulation based on the SDSALBP model of Andres, Miralles, and Pastor (2008). The model has been published in Lahrichi et al. (2020a).

We note that there already exists a mathematical formulation for the RTLBP suggested by Essafi et al. (2010). It is significantly different from the model described in this subsection. The model of the literature is based on a variable $x_{i,j}$ (=1, if operation i is assigned to the j^{th} position of the sequence). The latter variable is used to build an overall sequence of operations. Another variable is used to decide the positions where to cut the sequence and therefore to determine the sequence of operations in each workstation. The model suggested here does not use this second variable since the variable $x_{i,j}$ is replaced by $x_{i,s,j}$ (=1, if operation i is assigned to the j^{th} position of the sequence of workstation s). This latter variable addresses simultaneously the balancing decision (decide whether i is assigned to workstation s) and the sequencing decision (decide whether i is placed in the j^{th} position of the sequence of workstation s).

The suggested formulation uses the following binary variables:

$$x_{i,s,j} = \begin{cases} 1 & \text{If operation } i \text{ is assigned to workstation } s \text{ at the } j^{th} \text{ position of its sequence.} \\ 0 & \text{Otherwise.} \end{cases}$$

$$y_s = \begin{cases} 1 & \text{If at least one operation is assigned to workstation } s \\ 0 & \text{Otherwise.} \end{cases}$$

$$v_{s,k} = \begin{cases} 1 & \text{If } k \text{ machines are assigned to workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$$z_{i,i',s} = \begin{cases} 1 & \text{If operation } i \text{ is processed just before operation } i' \text{ at workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$$w_{i,s} = \begin{cases} 1 & \text{If operation } i \text{ is assigned to the last position of the sequence at workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$$u_{s,a} = \begin{cases} 1 & \text{If position } a \text{ is chosen for workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

We consider the objective of minimising the number of machines used:

$$Min \sum_{s=1}^{s_{max}} \sum_{k=1}^{M_{max}} k.v_{s,k}$$

under the constraints: (1.19-1.33).

$$\sum_{s=1}^{s_{max}} \sum_{j=1}^{N_{max}} x_{i,s,j} = 1, \forall i \in N \tag{1.19}$$

$$\sum_{i=1}^{n} x_{i,s,j} \leq 1, \forall s \in S, \forall j \in \{1, \ldots, N_{max}\} \tag{1.20}$$

$$\sum_{i=1}^{n} x_{i,s,j+1} \leq \sum_{i=1}^{n} x_{i,s,j}, \forall s \in S, \forall j \in \{1, \ldots, N_{max} - 1\} \tag{1.21}$$

$$\sum_{k=1}^{M_{max}} v_{s,k} = y_s, \forall s \in S \tag{1.22}$$

$$y_{s+1} \leq y_s, \forall s \in \{1, \ldots, s_{max} - 1\} \tag{1.23}$$

$$\sum_{s=1}^{s_{max}} \sum_{j=1}^{N_{max}} (N_{max}.(s-1)+j).x_{i,s,j} \leq \sum_{s=1}^{s_{max}} \sum_{j=1}^{N_{max}} (N_{max}.(s-1)+j).x_{i',s,j}, \forall (i,i') \in P \tag{1.24}$$

$$\sum_{i=1}^{n}\sum_{j=1}^{N_{max}} d_i.x_{i,s,j} + \sum_{i=1}^{n}\sum_{i'=1}^{n} t_{i,i'}.z_{i,i',s} \leq C.\sum_{k=1}^{M_{max}} k.v_{s,k}, \forall s \in S \tag{1.25}$$

$$x_{i,s,j} + x_{i',s,j+1} \leq 1 + z_{i,i',s}, \forall (i,i') \in N^2, i \neq i', \forall j \in \{1,\ldots,N_{max}-1\}, \forall s \in S \tag{1.26}$$

$$x_{i,s,j} - \sum_{i' \in N; i' \neq i} x_{i',s,j+1} \leq w_{i,s}, \forall i \in N, \forall s \in S, \forall j \in \{1,\ldots,N_{max}-1\} \tag{1.27}$$

$$x_{i,s,N_{max}} \leq w_{i,s}, \forall i \in N, \forall s \in S \tag{1.28}$$

$$w_{i,s} + x_{i',s,1} \leq 1 + z_{i,i',s}, \forall (i,i') \in N^2, i \neq i', \forall s \in S \tag{1.29}$$

$$\sum_{s=1}^{s_{max}}\sum_{j=1}^{N_{max}} s.x_{i,s,j} = \sum_{s=1}^{s_{max}}\sum_{j=1}^{N_{max}} s.x_{i',s,j}, \forall (i,i') \in I \tag{1.30}$$

$$\sum_{i \in ES}\sum_{j=1}^{N_{max}} x_{i,s,j} \leq |ES| - 1, \forall ES \in E, \forall s \in S \tag{1.31}$$

$$\sum_{a \in Pos} u_{s,a} \leq 1, \forall s \in S \tag{1.32}$$

$$\sum_{j=1}^{N_{max}} x_{i,s,j} - \sum_{a \in Pos_i} u_{s,a} \leq 0, \forall i \in N, \forall s \in S \tag{1.33}$$

The set of constraints (1.19) ensures that each operation is assigned to exactly one workstation at a unique position of its sequence. (1.20) ensures that at each workstation at most one operation is assigned in each position of the sequence. (1.21) ensures that at each workstation no operation is assigned at a position $j+1$ unless some operation is assigned at the position j. (1.22) ensures that only one number of machines is chosen for every used workstation. (1.23) ensures that no workstation is used unless its precedent workstation is also used. (1.24) ensures that precedence constraints are satisfied. (1.25) ensures that the cycle time is not exceeded at any workstation. (1.26) ensures that if operation i is followed by operation i' at workstation s then $z_{i,i',s}$ is set to 1. Constraints (1.27) and (1.28) ensure that $w_{i,s}$ is set to 1 whenever operation i is positioned at the last occupied position in the sequence of workstation s. (1.29) ensures that if operation i is positioned at the last occupied position in the sequence of workstation s and operation i' is positioned at the first position in the sequence of workstation s then $z_{i,i',s} = 1$ and consequently the setup time $t_{i,i'}$ is considered in (1.25). (1.30) ensures that inclusion constraints are satisfied while (1.31) insures that exclusion constraints are satisfied. (1.32) and (1.33) ensure that accessibility constraints are satisfied.

1.3.2.4 Complexity and lower bounds

It is well-known that the SALBP is NP-Hard since Bin-Packing (which is NP-Hard in the strong sense) can be reduce to it: Wee and Magazine (1982), Scholl and Klein (1999).

SDSALBP is also NP-Hard since SALBP is reduced to it. It can be seen by reducing the Travelling Salesman Problem to the SDSALBP. Indeed, a SDSALBP with $s_{max} = 1$ is a TSP where cities represent operations and distances between cities represent setup times.

RTLBP is also NP-hard since it subsumes SDSALBP as a special case when $M_{max} = 1$, $I, E = \emptyset$ and $A_i = A, \forall i \in N$.

A well-known lower bound for SALBP can be given as follows (Scholl and Klein (1997)):

- Lower bound on the cycle time: $\left\lceil \frac{\sum_{i \in N} d_i}{s_{max}} \right\rceil$
- Lower bound on the number of workstations: $\left\lceil \frac{\sum_{i \in N} d_i}{C} \right\rceil$

We adapt the lower bounds for the RTLBP (and the SDSALBP) as follows (inspired from Essafi et al. (2010):

- Lower bound on the cycle time: $\left\lceil \frac{\sum_{i \in N} d_i + \lambda_{1+n-s_{max}}}{s_{max}} \right\rceil$
- Lower bound on the number of workstations: $\left\lceil \frac{\sum_{i \in N} d_i + \lambda_{1+n-s_{max}}}{C} \right\rceil$

where $\lambda_{1+n-s_{max},r}$ denotes the sum of the $1 + n - s_{max}$ smallest setup times if $n > s_{max}$ and 0 otherwise.

1.4 Robotic Assembly Lines

We have considered so far that there is only one type of resources available to perform the operations in the workstations.

The latter assumption cannot be considered realistic in the real world assembly lines. Indeed, in many situations, the decision-maker has to choose between different types of equipment or between workers with different skills to perform the operations: Battaïa and Dolgui (2013). Such a situation can occur in the context of the Robotic Assembly Line Balancing Problem (RALBP).

We describe in the two next subsections respectively the Robotic Assembly Line Balancing Problem (RALBP) and the Sequence-Dependent Robotic Assembly Line Balancing Problem (SDRALBP). For clarity, only the -2 variant of the balancing problem is presented in this section.

1.4.1 The Robotic Assembly Line Balancing Problem

The Robotic Assembly Line Balancing Problem (RALBP) is a combinatorial optimization problem that is concerned with simultaneously assigning a set of operations to a set of workstations placed among a serial assembly line and assigning to each workstation a type of robot. The processing time of an operation i depends on the type of robot r used and is denoted by d_i^r. The RALBP-2 can be stated as follows:

- **Data:**
  - As for SALBP, a set N of n operations: $N = \{1, 2, \ldots, n\}$, a set P of precedence relations and a set S of s_{max} workstations: $S = \{1, 2, \ldots, s_{max}\}$ such that s_{max} is a given maximum number of workstations.
  - A set $R = \{1, 2, \ldots, n_r\}$ of types of robots.
  - The processing times such that d_i^r is the processing time of operation i on a robot of type r.
- **Decisions:**
  - Balancing decision: As for the SALBP, assign the operations to the workstations.
  - (Robot) Selection decision: Select a type of robot to each workstation.
- **Constraints:**
  - Respect precedence relations.
  - Respect the maximum number of workstations constraint.
- **Objectives:**
  - Minimize the cycle time which is the maximum workload on a workstation.

1.4.1.1 Illustrative example

We illustrate the RALBP thanks to a small example. The example is built from the previous examples: the number of operations ($n = 7$) and the precedence relations do not change (Figure 1.5). We consider 4 types of robots. The processing times type per type are represented in Table 1.3.

We consider a unlimited number of available robots per type.

	r \ i	1	2	3	4	5	6	7
d_i^r	$r = 1$	1.5	1	3.5	1.5	2.5	3	1
	$r = 2$	1.5	0.5	3	2	1.5	2	1
	$r = 3$	1	1	3	2	1	1	0.5
	$r = 4$	2	2	4	1.5	2	1	1

Table 1.3: Processing times.

A solution is represented in Figure 1.9.

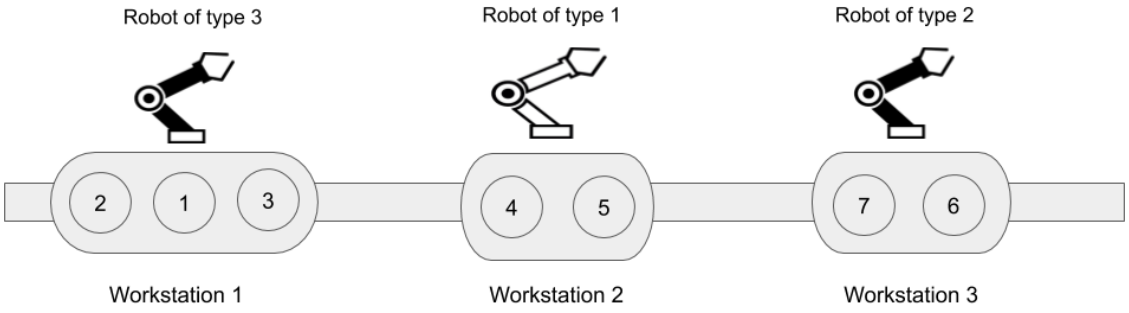

Figure 1.9: Example of feasible solution for the RALBP problem.

The solution uses 3 workstations. The cycle time is computed as follows:

- Workstation 1 is equipped with robot type 3. The workload is $d_2^3 + d_1^3 + d_3^3 = 3 + 1 + 1 = 5$.
- Workstation 2 is equipped with robot type 1. The workload is $d_4^1 + d_5^1 = 1.5 + 2.5 = 4$.
- Workstation 3 is equipped with robot type 2. The workload is $d_7^2 + d_6^2 = 2 + 1 = 3$.

From the above calculations we can deduce the cycle time: $C = Max(5, 4, 3) = 5$.

1.4.1.2 State of the art

The RALBP has been first introduced in Rubinovitz, Bukchin, and Lenz (1993). The RALBP has gained great importance due to its industrial relevance and due to the academic challenge it raises.

Two different assumptions are identified in literature:

- Many robots per type: in this assumption we are given a set of types of robots. Each type of robot can be assigned to multiple workstations without any limitation: Rubinovitz, Bukchin, and Lenz (1993), Yoosefelahi et al. (2012), Nilakantan et al. (2015), Çil, Mete, and Ağpak (2016), Borba, Ritt, and Miralles (2018).
- Only one robot per type: in this assumption we are given a set of robots each of which can be assigned at most to one workstation: Levitin, Rubinovitz, and Shnits (2006), Gao et al. (2009), Janardhanan et al. (2019).

The first assumption is the original assumption of RALBP as defined by Rubinovitz, Bukchin, and Lenz (1993). It is more studied in literature than the second assumption.

The second assumption can be of industrial relevance when the decision-maker has a limited fleet of robots. It is more constrained than the first assumption. Both assumptions are studied in the frame of the thesis.

The incompatibility between a robot and an operation can be addressed by considering that the corresponding processing time is infinite.

Even if the seminal paper (Rubinovitz, Bukchin, and Lenz (1993)) considers minimizing the number of workstations with a given cycle time (RALBP-1), most paper considers the objective of minimizing the cycle time: Levitin, Rubinovitz, and Shnits (2006), Nilakantan et al. (2015), Borba and Ritt (2014), Janardhanan et al. (2019). Some authors consider the latter objective jointly with minimizing the robots' cost (Yoosefelahi et al. (2012)) or the number of workstations (Çil, Mete, and Ağpak (2016)).

Many authors deal with the problem by means of metaheuristics: for example Levitin, Rubinovitz, and Shnits (2006) suggest a genetic algorithm for the RALBP-2, a solution is represented by three vectors: a vector that represents a sequence of operations, a vector that represents the workstations and a vector that represents the robots. However, only the first vector is involved in the genetic algorithm. An heuristic is suggested to build a solution from a sequence of operations. The same encoding/decoding procedure is adopted in Nilakantan et al. (2015)

The Assembly Line Worker Assignment and Balancing Problem

The Robotic Assembly Line Balancing Problem is similar to existing problems from the manual-manned assembly line literature where a double assignment of operations and workers to the workstations is encountered. There can be several reasons justifying that the two latter assignments must be tackled jointly. For example, the processing times of the operations may depend on the worker: in this particular context, the problem is (mathematically) equivalent to the RALBP. It is known as the Assembly Line Worker Assignment and Balancing Problem (ALWABP), it has been defined in Miralles et al. (2008). They consider the balancing problem alongside with the assignment of a set of workers with different characteristics (processing times) to the workstations. In the context of ALWABP, the worker is a limited resource: each worker can be assigned at most once.

A mathematical model is suggested in Miralles et al. (2007). The latter is similar to the model a model for RALBP once we consider that workers stand for robots. Other constraints which depend on the industrial context can be considered. In Miralles et al. (2007) and Miralles et al. (2008), the case of disabled workers is investigated. In this particular context, some workers cannot carry out certain operations. Besides, some a priori operation-worker and worker-station assignments must be considered due to therapeutic reasons. The authors consider minimizing the cycle time (ALWABP-2). Social benefit is also considered by making all workers having at least one operation to perform. Many recent papers have since considered ALWABP: Blum and Miralles (2011), Moreira et al. (2012), Vila and Pereira (2014), Borba and Ritt (2014), Akyol and Baykasoğlu (2019).

1.4.1.3 A new mathematical model for RALBP

Levitin, Rubinovitz, and Shnits (2006) suggested a mathematical formulation for the RALBP that uses a variable $x_{i,s}$ to decide whether operation i is assigned to workstation s and a second variable $v_{s,r}$ to decide whether robot r is selected for workstation s. This formulation has the disadvantage of not being linear, the cycle time is computed in some workstation s by: $\sum_i d_i^r . x_{i,s} . v_{s,r}$.

Borba, Ritt, and Miralles (2018) suggested a linear model using the same variables. The cycle time constraint in some workstation s is formulated as follows:

$$\sum_{i=1}^{n} d_i^r.x_{i,s} \leq C + M(1 - v_{s,r})$$

where M is a big number. This formulation has the disadvantage of using the big M technique which is known to be numerically unstable.

We suggested in the following a linear formulation, that uses no big M constraints. The formulation uses the following variables:

$$x_{i,s,r} = \begin{cases} 1 & \text{If the operation } i \text{ is assigned to the workstation } s \text{ and} \\ & \text{performed by a robot of type } r. \\ 0 & \text{Otherwise.} \end{cases}$$

$$v_{s,r} = \begin{cases} 1 & \text{If a robot of type } r \text{ is assigned to workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$$y_s = \begin{cases} 1 & \text{If at least one operation is assigned to workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$C =$ Cycle time

The constraints can be formulated as follows:

$$\sum_{s \in S} \sum_{r \in R} x_{i,s,r} = 1, \forall i \in N \tag{1.34}$$

$$\sum_{i \in N} x_{i,s,r} \leq v_{s,r}, \forall s \in S, \forall r \in R \tag{1.35}$$

$$\sum_{r \in R} v_{s,r} = y_s, \forall s \in S \tag{1.36}$$

$$y_{s+1} \leq y_s, \forall s \in S - \{s_{max}\} \tag{1.37}$$

$$\sum_{s \in S} \sum_{r \in R} s.x_{i,s,r} \leq \sum_{s \in S} \sum_{r \in R} s.x_{i',s,r}, \forall (i, i') \in P \tag{1.38}$$

$$\sum_{i \in N} \sum_{r \in R} d_i^r.x_{i,s,r} \leq C.y_s, \forall s \in S \tag{1.39}$$

The set of constraints (1.34) ensures that all operations must be assigned once and only once. (1.35) ensures that an operation is carried out by a robot of type r

on a workstation s only if the workstation s is equipped by a robot of type r and (1.36) ensures that no more than one type of robot can be selected for a workstation. (1.37) ensures that a workstation is only used if the previous workstations are also used. (1.38) ensures that the precedence constraints are respected. (1.39) ensures that the cycle time constraints are respected on all workstations.

The minimization of the cycle time can simply be formulated:

$$Min\ C$$

1.4.1.4 Complexity and lower bounds

RALBP is NP-hard since the SALBP can be reduced to the RALBP by assuming that there is only one type of robot ($n_r = 1$).

Remark 1. *The RALBP with the assumption (only one robot per type) is a generalization of the RALBP with many robots per type (basic RALBP). In other words, any algorithm than can solve the RALBP with this new assumption can also solve the basic RALBP. In terms of complexity theory, we say that the basic RALBP is reduced to the RALBP with only one robot per type. Indeed, given an instance of the basic RALBP, we can map it to an instance of the RALBP with the assumption "only one robot per type" where the set of robots is composed of all types of robots duplicated as many times as authorized workstations. The processing times on a robot are the processing times on the type of robots it belongs to.*

The previous lower bounds can easily be adapted for the RALBP (Borba and Ritt (2014)):

- Lower bound on the cycle time: $\left\lceil \frac{Min_{r \in R} \sum_{i \in N} d_i^r}{s_{max}} \right\rceil$
- Lower bound on the number of workstations: $\left\lceil \frac{Min_{r \in R} \sum_{i \in N} d_i^r}{C} \right\rceil$

1.4.2 The Sequence-Dependent Robotic Assembly Line Balancing Problem

The Sequence-Dependent Robotic Assembly Line Balancing Problem (SDRALBP) is the generalization of RALBP where sequence-dependent setup times are considered.

Sequence-dependent setup times $t^r_{i,i'}$ should be considered if operation i is performed just before operation i' in some workstation equipped by a robot of type r. We note that the setup times depend on the robot type. Compared to RALBP, sequence-dependent setup times raise an additional decision: three decisions must be addressed jointly:

- Balancing decision.
- (Robot) Selection decision.
- Sequencing problem.

Despite their industrial importance, sequence-dependent setup times have been rarely considered in literature in the context of RALBP. To the best of our knowledge, it has only been studied once in the literature by Janardhanan et al. (2019).

1.4.2.1 Illustrative example

The same example as the RALBP is considered. We give the sequence-dependent setup times in Table 1.4

Let us consider the same solution as the RALBP. The cycle time is computed as follows:

- Workstation 1 is equipped with robot type 3. The workload is $d^3_2 + t^3_{2,1} + d^3_1 + t^3_{1,3} + d^3_3 + t^3_{3,2} = 8.5$.
- Workstation 2 is equipped with robot type 1. The workload is $d_{4,1} + t^1_{4,5} + d_{5,1} + t^1_{5,4} = 4.75$.
- Workstation 3 is equipped with robot type 2. The workload is $d_{7,2} + t^2_{7,6} + d_{6,2} + t^2_{6,7} = 4$.

From the above calculations we can deduce the cycle time: $C = Max(8.5, 4.75, 4) = 8.5$.

1.4.2.2 An adapted mathematical model

To clarify the definition of the problem, we give a linear formulation based on the one of Janardhanan et al. (2019). The latter considers a limited number of robots per type of robot, it can be adapted for the case of an unlimited number of robots by type. The formulation of Janardhanan et al. (2019) is itself adapted from Andres, Miralles, and Pastor (2008).

We use i to index an operation, s to index a workstation, j to index a position in the sequence of operations assigned to a workstation and r to index a type of robot.

(a) Setup times of robots of type 1

	j	1	2	3	4	5	6	7
$t^1_{i,j}$	$i=1$	0	1	0.5	1.5	0.5	1	1
	$i=2$	1.5	0	0	2	0.5	0.5	1
	$i=3$	1	1	0	2	1	1	0.5
	$i=4$	0.5	0.5	0.5	0	0.5	1	1
	$i=5$	1.5	0.5	0.25	0.25	0	0.25	1
	$i=6$	1.5	0.5	0.5	2	0.5	0	1
	$i=7$	0.5	1	0.5	0.5	0.5	1	0

(b) Setup times of robots of type 2

	j	1	2	3	4	5	6	7
$t^2_{i,j}$	$i=1$	0	1	0.25	1.5	0.25	0.5	1
	$i=2$	1.5	0	0.25	0.25	0.25	0.25	1
	$i=3$	1	0.5	0	0.25	1	1	0.5
	$i=4$	0.25	2	4	0	2	1	1
	$i=5$	1	1	1	2	0	1	0.5
	$i=6$	1	1	1	2	1	0	0.5
	$i=7$	1.5	0.5	0.5	2	0.5	0.5	0

(c) Setup times of robots of type 3

	j	1	2	3	4	5	6	7
$t^3_{i,j}$	$i=1$	0	1	0.75	1.5	0.75	0.75	1
	$i=2$	0.75	0	0.75	2	1.5	0.75	1
	$i=3$	1	1	0	2	1	1	0.5
	$i=4$	2	0.5	0.75	0	0.75	1	1
	$i=5$	1	0.5	0.25	0.25	0	1	0.5
	$i=6$	1.5	0.5	0.5	2	0.5	0	1
	$i=7$	1	0.5	0.25	0.25	1	1	0

(d) Setup times of robots of type 4

	j	1	2	3	4	5	6	7
$t^4_{i,j}$	$i=1$	0	1	0.75	0.5	0.5	1	1
	$i=2$	1.5	0	3	2	1.5	2	1
	$i=3$	1	1	0	2	1	1	0.5
	$i=4$	2	1	0.5	0	2	1	1
	$i=5$	1.5	0.5	0.5	2	0	0.5	1
	$i=6$	1	0.5	0.25	0.25	1	0	0.5
	$i=7$	0.5	0.5	1	2	1	0.5	0

Table 1.4: Sequence-dependent setup times

The model has been published in Lahrichi et al. (2020c). The following variables are used:

$$x_{i,s,j,r} = \begin{cases} 1 & \text{If the operation } i \text{ is assigned to the workstation } s \text{ at the } j^{th} \\ & \text{position of its sequence and performed by a robot of type } r. \\ 0 & \text{Otherwise.} \end{cases}$$

$$y_s = \begin{cases} 1 & \text{If at least one operation is assigned to workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$$v_{s,r} = \begin{cases} 1 & \text{If a robot of type } r \text{ is assigned to workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$$z_{i,i',s,r} = \begin{cases} 1 & \text{If operation } i \text{ is performed just before operation } i' \text{ at} \\ & \text{workstation } s \text{ by a robot of type } r. \\ 0 & \text{Otherwise.} \end{cases}$$

$$w_{i,s} = \begin{cases} 1 & \text{If the operation } i \text{ is assigned to the last position in the} \\ & \text{sequence of station } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$C =$ Cycle time

The objective of minimizing the cycle time is considered: $Min\ C$

$$\sum_{s \in S} \sum_{j \in N} \sum_{r \in R} x_{i,s,j,r} = 1, \forall i \in N \tag{1.40}$$

$$\sum_{i \in N} \sum_{r \in R} x_{i,s,j,r} \leq 1, \forall s \in S, \forall j \in N \tag{1.41}$$

$$\sum_{i \in N} x_{i,s,j,r} \leq v_{s,r}, \forall s \in S, \forall j \in N, \forall r \in R \tag{1.42}$$

$$\sum_{r \in R} v_{s,r} = y_s, \forall s \in S \tag{1.43}$$

$$\sum_{i \in N} x_{i,s,j+1,r} \leq \sum_{i \in N} x_{i,s,j,r}, \forall s \in S, \forall j \in N - \{n\}, \forall r \in R \tag{1.44}$$

$$y_{s+1} \leq y_s, \forall s \in S - \{s_{max}\} \tag{1.45}$$

$$\sum_{s\in S}\sum_{j\in N}\sum_{r\in R}(n.(s-1)+j).(x_{i',s,j,r}-x_{i,s,j,r})\geq 0, \forall (i,i')\in P \quad (1.46)$$

$$\sum_{i\in N}\sum_{j\in N}\sum_{r\in R} d_i^r.x_{i,s,j,r}+\sum_{i\in N}\sum_{i'\in N}\sum_{r\in R} t_{i,i',r}.z_{i,i',s,r}\leq C.y_s, \forall s\in S \quad (1.47)$$

$$x_{i,s,j,r}+x_{i',s,j+1,r}\leq 1+z_{i,i',s,r}, i,i'\in N^2, i\neq i', j\in N-\{n\}, s\in S, r\in R \quad (1.48)$$

$$x_{i,s,j,r}-\sum_{i'\in N; i'\neq i} x_{i',s,j+1,r}\leq w_{i,s}, \forall i\in N, \forall s\in S, \forall j\in N-\{n\}, \forall r\in R \quad (1.49)$$

$$x_{i,s,n,r}\leq w_{i,s}, \forall i\in N, \forall s\in S, \forall r\in R \quad (1.50)$$

$$w_{i,s}+x_{i',s,1,r}\leq 1+z_{i,i',s,r} \forall i\in N, i'\in N, i\neq i', \forall s\in S, \forall r\in R \quad (1.51)$$

The set of constraints (1.40) ensures that all operations must be assigned once and only once. The set of constraints (1.41) ensures that at most one operation can be assigned to the same position. The set of constraints (1.42) ensures that an operation is carried out by a robot of type r on a workstation s only if the workstation s is equipped by a robot of type r. The set of constraints (1.43) ensures that no more than one type of robot can be assigned to a workstation. The set of constraints (1.44) ensures that a position is only occupied by an operation if all of its previous positions are also occupied. The set of constraints (1.45) ensures that a workstation is only used if the previous workstations are also used. The set of constraints (1.46) ensures that the precedence constraints are respected. The set of constraints (1.47) ensures that the cycle time constraints are respected on all workstations. The set of constraints (1.48) ensures that $z_{i,i',s,r}=1$ when i and i' follow each other on the workstation s (equipped by the robot r). The set of constraints (1.49) - (1.51) verifies that $z_{i,i',s,r}=1$ when i is the last operation assigned to the workstation s and i' the first operation assigned at the workstation s (equipped by the robot r).

1.4.2.3 Complexity and lower bounds

SDRALBP is NP-hard since the SALBP can be reduced to the SDRALBP by assuming that the setup times are null and that $n_r=1$ (only one robot).

We adapt the bounds for the SDRALBP:

- Lower bound on the cycle time: $\left\lceil \frac{Min_{r\in R}\sum_{i\in N} d_i^r+\lambda_{1+n-s_{max},r}}{s_{max}} \right\rceil$
- Lower bound on the number of workstations: $\left\lceil \frac{Min_{r\in R}\sum_{i\in N} d_i^r+\lambda_{1+n-s_{max},r}}{C} \right\rceil$

where $\lambda_{1+n-s_{max},r}$ denotes the sum of the $1+n-s_{max}$ smallest setup times on robot type r if $n>s_{max}$ and 0 otherwise.

1.5 Research gap and goals of the thesis

From an academic perspective, the motivation of the thesis is to study the balancing problem jointly with the sequencing and the robot selection problems. Those problems are of industrial relevance in the context of industrial 4.0 as previously explained.

We first study the balancing problem jointly with the sequencing problem: the problem is known in the literature as the Sequence-Dependent Simple Assembly Line Balancing Problem (SDSALBP). A review of the literature allowed us to identify the Reconfigurable Transfer Line Balancing Problem (RTLBP). This problem has the advantage of being a generalization of the SDSALBP. Besides, it considers additional industrial constraints and the problem fits into the context of reconfigurability which is fundamental in Industry 4.0. The RTLBP has been rarely considered in literature. We identified two research gaps:

- When dealing with joint resolution of two optimization problems (for example, the balancing and sequencing problems), a natural approach would be to investigate the sequential methods (Balance-First Sequence-Last and Balance-First sequence-Last methods). Such approaches are well known in combinatorial optimization and have been applied to a wide variety of problems (for example the vehicle routing problems Prins, Lacomme, and Prodhon (2014)). To the best of our knowledge, such a study has never been conducted for the RTLBP or more generally for the SDSALBP.
- In Borisovsky, Delorme, and Dolgui (2013) and Delorme, Malyutin, and Dolgui (2016), a useful case is identified: the case where an overall sequence of operations is imposed (the precedence graph is a path containing all the operations). This special case is very useful since in most metaheuristic resolution approaches, a sequence is used to encode a solution. No polynomial algorithm was known to solve optimally this special case. Borisovsky, Delorme, and Dolgui (2013) and Delorme, Malyutin, and Dolgui (2016) were content to use integer programming (exponential resolution) or heuristic to solve the problem. This could cause that the optimal solution is not reached (in the case of an heuristic decoder) or that the resolution time is prohibitive (in the case of an ILP decoder).

In a second time, we study the robot selection problem jointly with the balancing and sequencing problems which take us to the Sequence-Dependent Robotic Assembly Line Problem (SDRALBP). The problem fits into the context of the growing robotization of assembly lines and therefore in Industry 4.0. We consider the objective of minimizing the cycle time. Two research gaps are identified:

- Even if the Robotic Assembly Line Balancing Problem (RALBP) is well-established, the study of sequence-dependent setup times in the context of RALBP has rarely been addressed. To the best of our knowledge, there is only one study addressing the problem: Janardhanan et al. (2019). Besides, there is no comparative study between the two assumptions considered in RALBP (one or many robots per type).

- In many researches dealing with robotic assembly lines by means of meta-heuristics, a sequence of operations is used to encode a sequence. To decode the sequence, most authors (for example Nilakantan et al. (2015)) use a heuristic suggested by Levitin, Rubinovitz, and Shnits (2006). The greedy procedure is called consecutive assignment procedure. It is exponential and does not guaranty optimally. To the best of our knowledge no polynomial procedure was known in literature to solve this special case where the giant sequence is given. Many authors were content to use heuristic decoders which deprive them from accessing the optimal solution.

The thesis aims to fill the previous identified research gaps.

Figure 1.10 shows the positions of the RTLBP, the RALBP and the SDRALBP in relation to SALBP. These problems are generalizations of the SALBP. RTLBP is a generalization of the Sequence-Dependent Simple Assembly Line Balancing Problem (SDSALBP) since it considers further industrial constraints and the possibility of having several machines on the same workstation. SDRALBP is a generalization of SDSALBP and RALBP. It contains the three decisions: balancing, sequencing and selection.

To solve a combinatorial optimization problem involving several decisions or subproblems, a natural and quite intuitive approach would be to solve each subproblem separately one after the other. Each subproblem is based on the solution of the subproblem solved before. Such a technique is known as a sequential approach and has been applied successfully to a wide range of optimization problems: Vecchi et al. (2016). Sequential approaches have multiplied in recent years with the appearance of new optimization problems merging several subproblems. These problems are often very challenging to solve. An integrated approach that would attempt to solve the problem without taking advantage of the subproblems can be ineffective or time-consuming. A sequential approach makes use of the solutions of the subproblems to build up a solution for the master problem. The subproblems are often smaller than the master problem which makes them computationally solvable. The latter gives a major advantage of sequential approaches over integrated approaches. Nevertheless, we must highlight that breaking a combinatorial optimization problem into several

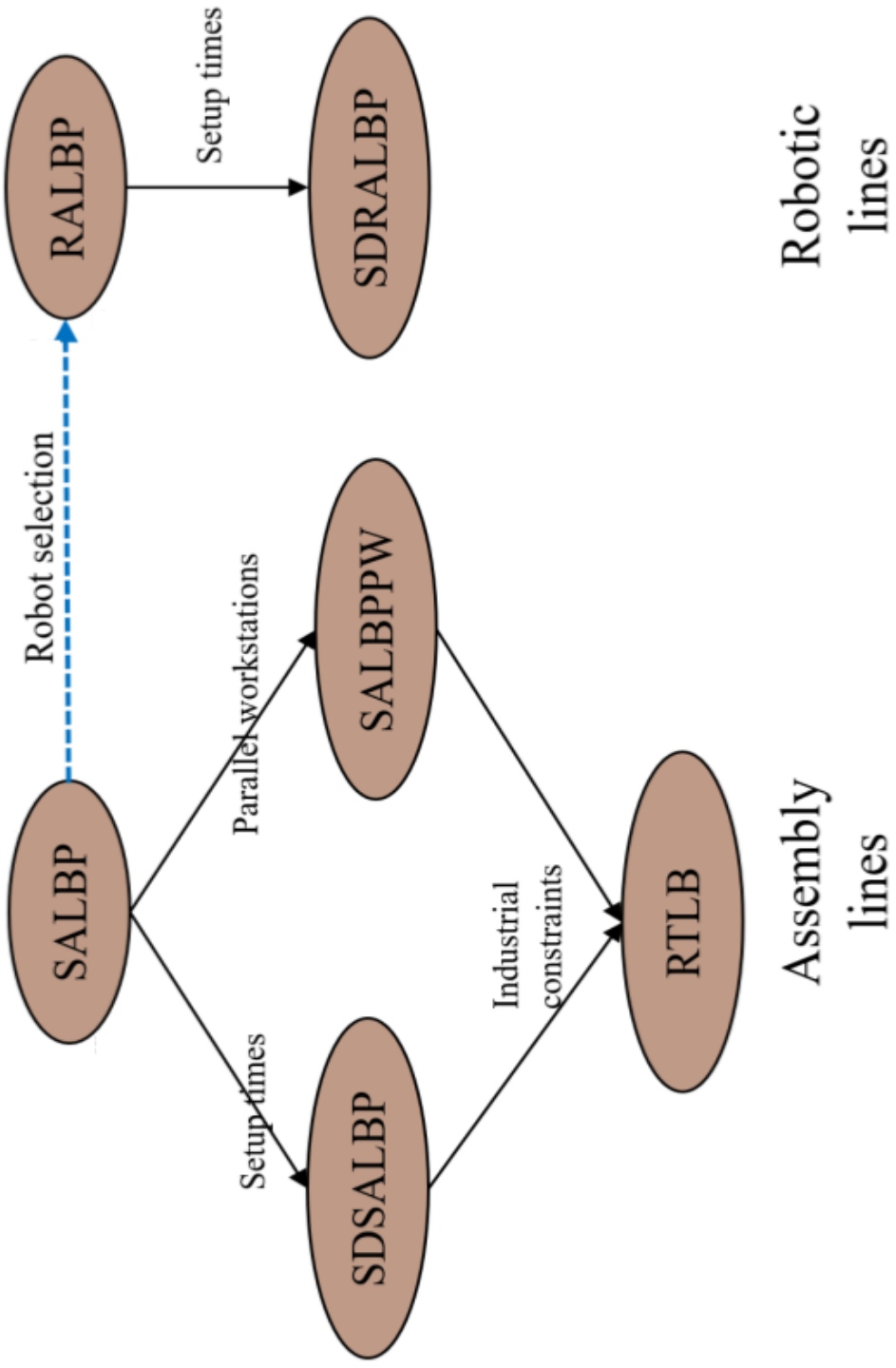

Figure 1.10: Position of the RTLBP and the SDRALBP in the assembly line balancing literature.

problems solved separately generally result in some optimality loss. A successful sequential approach would minimize this optimality loss.

Different sequential approaches are suggested in the remainder of the thesis. Next chapter is devoted to Balance-First Sequence-Last approach for the RTLBP. We emphasis afterwards on Sequence-First Balance-Last approch for the RTLBP and Sequence-First Balance-And-Select-Last approach for the RALBP and the SDRALBP.

We describe in Table 1.5 the notations used in the remainder of the thesis.

(a) General notations

n	Number of operations
N	Set of operations, indexed on $\{1, 2, \ldots, n\}$
s_{max}	Maximum number of workstations
S	Set of workstations, indexed on $\{1, 2, \ldots, s_{max}\}$
P	Set of couples $(i, j) \in N \times N$ such that i precedes j (also denoted $i << j$)
C	Cycle time
w_s	Workload of station s

(b) Specific notations to RTLBP

M_{max}	Maximum number of machines in a workstation
N_{max}	Maximum number of operations in a workstation
s_{max}	Maximum number of workstations/robots to be used
S	Set of workstations, indexed on $\{1, 2, \ldots, s_{max}\}$
d_i	Processing time of operation i.
$t_{i,j}$	Setup time to be considered when operation i is performed just before operation j in some workstation
I	Set of couples $(i, j) \in N \times N$ linked with an inclusion constraint
E	Set of subsets of operations that cannot be assigned to the same workstation
Pos	Set of all possible part-fixing positions.
Pos_i	Subset of possible part-fixing positions for operation i.

(c) Specific notations to SDRALBP

n_r	Number of robot types available
R	Set of robot types, indexed on $\{1, 2, \ldots, n_r\}$
$d_i^r, i \in N, r \in R$	processing time of operation i on a robot of type r
$t_{i,i'}^r, i, i' \in N, r \in R$	Setup time between operations i and i' on a robot of type r

Table 1.5: Notations

Chapter 2

A Balance-First Sequence-Last Approach for the RTLBP

Contents

2.1 Introduction **47**
2.2 Heuristic BFSL (H-BFSL): an approximation algorithm of type BFSL **49**
2.2.1 Balancing generation 50
2.2.2 Operation sequencing 53
2.2.3 Constraint generation scheme 55
2.3 M-BFSL: A matheuristic of type BFSL **57**
2.3.1 Solution' encoding 59
2.3.2 Neighborhood moves 59
2.3.3 Adaptive simulated annealing 62
2.4 Conclusion **63**

2.1 Introduction

As previously discussed, the RTLBP includes two decisions (the balancing and the sequencing decisions), two sequential approaches are then possible:

- **Balance-First Sequence-Last (BFSL):** The sequencing step is done after the balancing step.
- **Sequence-First Balance-Last (SFBL):** The balancing step is done after the sequencing step.

This chapter is dedicated to BFSL approach.

An algorithm of type BFSL works within two consecutive steps:

- Balancing step: the operations are assigned to the workstations. The sequence-dependent setup times are ignored (or underestimated). In this step, some constraints of the problem must be respected including precedence, inclusion, exclusion, accessibility and the maximum number of operations per workstation constraint. It is not relevant to consider the cycle time and the maximum number of machines per workstation constraint in the balancing step since the exact workloads of the workstations and the number of machines to be hosted in each workstation remain unknown at this level.
- Sequencing step: the operations are sequenced in each workstation. The balancing solution from the previous step is considered as input. The sequencing step must account for the precedence relations.

After performing the sequencing step, the workloads of each workstation can be computed. This allows to determine the number of machines to be hosted in each workstation so that the cycle time constraint is respected. We know that at this level only the maximum number of machines per workstation constraint remains to be verified, all the other constraints have already been taken into account in the balancing and sequencing steps. If we ever end up with a solution that violates the latter constraint, then either the balancing or the sequencing decision must be questioned. For this reason, designing an algorithm of type BFSL can be delicate in the case of such a constrained problem as the RTLB. Careful management of the constraints is required to yield a feasible solution of good quality.

To define properly an algorithm of type BFSL, one have to define the methods used for the balancing and sequencing steps as well as the technique used to ensure the feasibility and the quality of the obtained solution. This is the objective of the next sections.

We propose an heuristic of type BFSL (H-BFSL). We show that the algorithm approximates the optimal solution when the setup times are bounded by the processing times, an approximation ratio is given. The approximation algorithm is based on linear programming, constraint generation and dynamic programming. A matheuristic (M-BFSL) is then suggested, it is made of a constructive phase and an improvement phase. H-BFSL is used as a constructive phase and an adaptive simulated annealing algorithm hybridized with dynamic programming is used as an improvement phase. The following two sections are respectively dedicated to the

approximation algorithm H-BFSL and the matheuristic M-BFSL. This work has been published in Lahrichi et al. (2020b).

2.2 Heuristic BFSL (H-BFSL): an approximation algorithm of type BFSL

The approximation algorithm is a two-step iterative method piloted by a constraint generation algorithm:

- Balancing generation: The first step consists in solving the line balancing problem with only a partial consideration of the sequence-dependent setup times. Only a lower bound of the setup times is considered. In this step, the balancing decision is the only decision to make with the objective of minimizing the number of machines. We solve this problem thanks to an ILP.
- Operations sequencing: In the second step, we consider sequence-dependent setup times. This implies the decision of sequencing the operations in each workstation taking as input the solution of the first step. The sequencing problem is optimally solved thanks to a dynamic programming algorithm.

Due to the consideration of sequence-dependent setup times in the second step, the load of the workstations increases. This can result in exceeding the maximum number of machines authorized in some workstation. For this reason, constraints are iteratively generated and added to the model of the first step. Those constraints aim at forbidding solutions where the maximum number of machines authorized in the workstations is exceeded.

Each of these three points are explained separately in the following 3 subsections. The general scheme of H-BFSL is described in Figure 2.1.

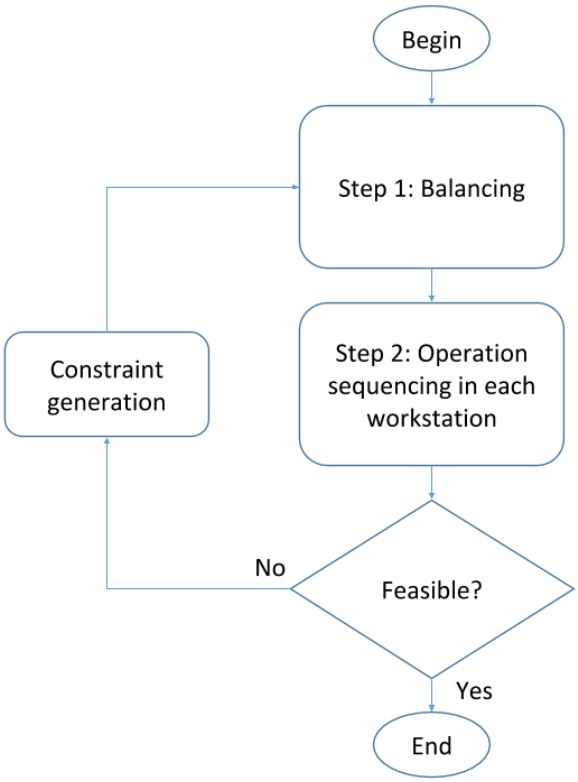

Figure 2.1: General scheme of H-BFSL algorithm

2.2.1 Balancing generation

The model of the RTLBP, described in chapter 1 (subsection 1.3.2.3), cannot be used to solve medium to large size instances. In order to limit the number of variables and constraints, we propose to use a relaxed version in which the sequence of operations in each workstations is not taken into account. As a consequence, variables $x_{i,s,j}$ of the RTLBP model become $x_{i,s}$ since we don't need to deal with the position in the sequence. Moreover, the variables $z_{i,i',s}$ and $w_{i,s}$ are no longer useful. All the constraints that use these variables are modified and we propose a simplified model described below. If the sequence-dependent setup times are not considered, the total workload computed for each workstation may be significantly inferior to the real value. In order to give a better lower bound, we consider for each operation i a setup time t_i^{min} equal to the smallest setup time to or from operation i:

$$t_i^{min} = Min(Min_{j \in N, j \neq i}\{t_{i,j}\}, Min_{j \in N, j \neq i}\{t_{j,i}\})$$

And so a better lower bound for workstation s is given for the total workload by the formula:

- If at least 2 operations are assigned to workstation s:

$$\sum_{i \in N} (d_i + t_i^{min}).x_{i,s}$$

- If only one operation is assigned to workstation s:

$$\sum_{i \in N} d_i.x_{i,s}$$

A setup time t_i^{min} is considered for an operation i in a workstation s only if operation i is assigned to s and if another operation is also assigned to s. For this reason, we consider a variable $\tilde{x}_{i,s}$ to decide whether an operation i is assigned to workstation s with another operation and a variable $\tilde{w}_s$ to decide if at least two operations are assigned to workstation s.

We use the following decision variables:

$$x_{i,s} = \begin{cases} 1 & \text{If operation } i \text{ is assigned to workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$$y_s = \begin{cases} 1 & \text{If at least one operation is assigned to workstation } s \\ 0 & \text{Otherwise.} \end{cases}$$

$$v_{s,k} = \begin{cases} 1 & \text{If } k \text{ machines are assigned to workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$$u_{s,a} = \begin{cases} 1 & \text{If position } a \text{ is chosen for workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$$\tilde{w}_s = \begin{cases} 1 & \text{If at least 2 operations are assigned to workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

$$\tilde{x}_{i,s} = \begin{cases} 1 & \text{If operation } i \text{ is assigned to workstation } s \text{ and } i \text{ is not the only} \\ & \text{operation assigned to workstation } s. \\ 0 & \text{Otherwise.} \end{cases}$$

We consider the objective of minimizing the number of machines used:

$$Min \sum_{s=1}^{s_{max}} \sum_{k=1}^{M_{max}} k.v_{s,k}$$

under the constraints:

$$\sum_{s=1}^{s_{max}} x_{i,s} = 1, \forall i \in N \tag{2.1}$$

$$\sum_{k=1}^{M_{max}} v_{s,k} = y_s, \forall s \in S \tag{2.2}$$

$$\sum_{i=1}^{n} x_{i,s} \leq y_s.N_{max}, \forall s \in S \tag{2.3}$$

$$y_{s+1} \leq y_s, \forall s \in S - \{s_{max}\} \tag{2.4}$$

$$\sum_{s=1}^{s_{max}} s.x_{i,s} \leq \sum_{s=1}^{s_{max}} s.x_{i',s}, \forall (i,i') \in P \tag{2.5}$$

$$2.\tilde{w}_s \leq \sum_{i=1}^{n} x_{i,s}, \forall s \in S \tag{2.6}$$

$$\sum_{i=1}^{n} x_{i,s} - 1 \leq \tilde{w}_s.N_{max}, \forall s \in S \tag{2.7}$$

$$x_{i,s} + \tilde{w}_s \leq \tilde{x}_{i,s} + 1, \forall i \in N, \forall s \in S \tag{2.8}$$

$$\tilde{x}_{i,s} \leq \sum_{i'=1, i \neq i'}^{n} x_{i',s}, \forall i \in N, \forall s \in S \tag{2.9}$$

$$\tilde{x}_{i,s} \leq x_{i,s}, \forall i \in N, \forall s \in S \tag{2.10}$$

$$\sum_{i=1}^{n} (d_i.x_{i,s} + t_i^{min}.\tilde{x}_{i,s}) \leq C. \sum_{k=1}^{M_{max}} k.v_{s,k}, \forall s \in S \tag{2.11}$$

$$\sum_{s \in S} s.x_{i,s} = \sum_{s \in S} s.x_{i',s}, \forall (i,i') \in I \tag{2.12}$$

$$\sum_{i \in ES} x_{i,s} \leq |ES| - 1, \forall ES \in E, \forall s \in S \tag{2.13}$$

$$\sum_{a \in Pos} u_{s,a} \leq 1, \forall s \in S \tag{2.14}$$

$$x_{i,s} - \sum_{a \in Pos_i} u_{s,a} \leq 0, \forall i \in N, \forall s \in S \tag{2.15}$$

Constraints (2.1) ensure that each operation is assigned to exactly one workstation. Constraints (2.2) ensure that only one number of machines is chosen for every used workstation. Constraints (2.3) ensure that the maximum number of operations to be allocated to a workstation is respected. Constraints (2.4) ensure that no workstation is used unless its precedent workstation is also used. Constraints (2.5) ensure that precedence constraints are satisfied. Constraints (2.6) and (2.7) ensure that $\tilde{w}_s = 1$ if and only if at least 2 operations are assigned to workstation s. Constraints (2.8) to (2.10) ensure that $\tilde{x}_{i,s} = 1$ if and only if at least 2 operations including i are assigned to workstation s. Constraints (2.11) ensure that the cycle time is not exceeded in any workstation. Constraints (2.12) ensure that inclu-

sion constraints are satisfied. Constraints (2.13) ensure that exclusion constraints are satisfied. Constraints (2.14) and (2.15) ensure that accessibility constraints are satisfied.

We denote by $\mathcal{ILP}(\mathcal{C})$ the ILP mentioned above under the set of constraints $\mathcal{C}$ ($\mathcal{C} = (2.1) - (2.15)$).

2.2.2 Operation sequencing

From balancing generation, we are given the assignment of operations to the workstations, we are now concerned with sequencing the operations in every workstation.

The sequencing problem is an ATSP (*Asymmetric Traveling Salesman Problem*) where operations represent cities and setup times distances between cities. However, we must consider precedence constraints within the same workstations, the problem induced is an ATSP with precedence constraints sometimes cited in the literature as the *Precedence Constrained Traveling Salesman Problem* (PCTSP) as stated in Bianco et al. (1994), Salii (2019).

To solve this problem we consider the well-known Held and Karp algorithm Held and Karp (1962). The resulting algorithm is of complexity within $O(n_s^2.2_s^n)$ where n_s is the number of cities. We apply this algorithm to every workstation. The number of operations in each workstation is limited to N_{max} therefore the complexity is within $O(N_{max}^2.2^{N_{max}})$. Even for large industrial instances N_{max} rarely exceeds 15 which make the approach reasonable to use (for $N_{max} = 15$ we have $N_{max}^2.2^{N_{max}} = 7,372,800$).

We suppose we are placed in some workstation s to which n_s operations are assigned. We number the operations from 1 to n_s. We want to construct a tour starting from operation 1 and returning to 1 passing by all the operations such that precedence constraints are respected and such that the total sum of setup times over the tour is minimized. We put $T = \{1, \ldots, n_s\}$. Given a subset $U \subset T$ such that $1 \in U$ and an operation i such that $i \in U$ and $i > 1$, the idea is to use the following dynamic programming formula:

$$c(U, i) = Min_{j \in U - \{i\}}(c(U - \{i\}, j) + t_{j,i}) \tag{2.16}$$

where $c(U, i)$ is the minimum cost of a path going from 1 to i and passing by each one of the other operations of U exactly once. The dynamic programming formula is initialized as follows:

$$c(U, 1) = +\infty; \forall U \subset T \text{ such that } 1 \in U \text{ and } |U| \geq 2 \tag{2.17}$$

$$c(\{1\}, 1) = 0 \tag{2.18}$$

To take precedence constraints into consideration, we must put:

$$c(U, i) = +\infty; \forall (U, i) \text{ such that } i \in U \text{ and } i \text{ has a successor in } U \tag{2.19}$$

(2.16)-(2.19) give the dynamic programming formula in order to compute all the $c(U, i), \forall (U, i)$ such that $i \in U$. We do not use a recursive function to avoid the computation of the same $c(U, i)$ several times. We compute $C(U, i), i \in U$ starting from the sets U such that $|U| = 2$ to $|U| = n_s$ (increasing cardinality). We then compute the cost of the optimal solution as follows:

$$c^* = Min_{j \in T - \{1\}}(c(T, j) + t_{j,1}) \tag{2.20}$$

This guaranties the optimality and the feasibility of the solution. The algorithm is depicted in Algorithm 1.

Algorithm 1 Dynamic programming sequencing algorithm

INPUT: An instance of the RTLBP. A subset T of n_s operations numbered from 1 to n_s.
OUTPUT: A sequence of the operations of T minimizing the sum of the setup times.

$c(\{1\}, 1) = 0$
for $k = 2$ **to** n_s **do**
  for each $U \subset T$ such that $|U| = k$ and $1 \in U$ **do**
    $c(U, 1) = +\infty$
    for $i \in U$ such that $i > 1$ **do**
      if i has a successor in U **then**
        $c(U, i) = +\infty$
      else
        $c(U, i) = Min_{j \in U - \{i\}}(c(U - \{i\}, j) + t_{j,i})$
      end if
    end for
  end for
end for
$c^* = Min_{j \in T - \{1\}}(c(T, j) + t_{j,1})$
Decode the solution of cost c^* to build an optimal sequence.

2.2.3 Constraint generation scheme

After solving the ILP from balancing generation, the sequence-dependent setup times are taken into consideration by running the dynamic programming algorithm from operations sequencing in each workstation. Since sequence-dependent setup times were not accounted for exactly in the balancing generation, we can end up with a solution exceeding the maximum number of machines allowed in some workstation after operations sequencing. To tackle this issue, constraints are iteratively added to $\mathcal{C}$. Those constraints aim at forbidding the assignment of the set of operations leading to the violation of the M_{max} constraint to the same workstation. More formally, let us denote by X the solution obtained after applying balancing generation and operations sequencing. If X is feasible then the algorithm terminates. If X is unfeasible, there must exist a workstation s where more machines than M_{max} are required to respect the cycle time. Let $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_{n_s})$ be the optimal sequence of the operations in s. We have:

$$d_{\sigma_1} + t_{\sigma_1,\sigma_2} + d_{\sigma_2} + t_{\sigma_2,\sigma_3} + \cdots + d_{\sigma_{n_s}} + t_{\sigma_{n_s},\sigma_1} > M_{max}.C$$

To obtain a feasible solution, we must make sure that the set of operations in σ is not authorized in any workstation. For this reason, we iterate through balancing generation by adding the following constraints to the ILP:

$$\sum_{u \in T} x_{u,s} - \sum_{u \in N-T} x_{u,s} \leq |T| - 1, \forall s \in S \qquad (C_T)$$

Two situations are possible:

- A proper subset of T is assigned to s:

$$\sum_{u \in T} x_{u,s} - \sum_{u \in N-T} x_{u,s} = \sum_{u \in T} x_{u,s} \leq |T| - 1$$

- A superset of T or a set containing at least one operation outside T is assigned to s:

$$\sum_{u \in T} x_{u,s} - \sum_{u \in N-T} x_{u,s} < \sum_{u \in T} x_{u,s} \leq |T| \implies \sum_{u \in T} x_{u,s} - \sum_{u \in N-T} x_{u,s} \leq |T| - 1$$

All in all, H-BFSL is a constraint generation algorithm. It is depicted in Algorithm 2. In the beginning only constraints (2.1)-(2.15) are considered. If we end up with an unfeasible solution, then constraints C_T cutting the unfeasible solution are iteratively generated:

Theorem 2.2.1. *H-BFSL terminates and outputs a feasible solution within a finite number of iterations (provided the instance admits at least a feasible solution).*

Algorithm 2 H-BFSL

INPUT: An instance of RTLBP.
OUTPUT: X, A feasible solution.

1: Initialize the set of constraints: $\mathcal{C} := (2.1) - (2.15)$
2: **repeat**
3: Solve the ILP of balancing generation: $X = solve[\mathcal{ILP}(\mathcal{C})]$
4: Run Algorithm 1 of operations sequencing in each workstation (X is modified)
5: **for all** T set of operations assigned to a workstation where the M_{max} constraint is violated **do**
6: $\mathcal{C} = \mathcal{C} \cup C_T$
7: **end for**
8: **until** X is feasible

Proof. In the worst case, H-BFSL forbids all sets T that leads to exceeding the maximum number of machines M_{max}. It has then at most as many iterations as the number of sets T it forbids plus one (this number is bounded by the number of combinations of at most N_{max} elements among n). The solution outputted is feasible since all the constraints except M_{max} constraint were taken into consideration in the balancing generation step and the M_{max} constraint is considered in the constraint generation. □

Theorem 2.2.2. *H-BFSL is a 2-approximation algorithm if we assume that:*

$$t_{i,j} \leq min(d_i, d_j) \quad \forall (i,j) \in N^2 \tag{2.21}$$

Proof. The solution outputted by the algorithm is feasible and its overall cost (denoted c) is given by the cost of the solution outputted by the balancing generation (denoted c_1) plus the number of machines added following operations sequencing due to the consideration of setup times (denoted k), i.e $c = c_1 + k$. Besides we have: $c_1 \leq c^*$ where c^* denotes the optimal solution of the RTLBP. Thanks to (2.21) we have $k \leq c_1$ because the number of setup times for each workstation is less or equal to the number of operations. Then the workload induced by the setup times in each workstation is less or equal to the workload induced by the operations times. Those two inequations ($c_1 \leq c^*$, $k \leq c_1$) finally give:

$$c \leq 2.c^*$$

which shows the approximation ratio.

□

Even if both subproblems (balancing and sequencing) are solved to optimality in H-BFSL, the given solution is not guaranteed to be optimal because the two

subproblems are not solved jointly. It is a heuristic with guaranteed performance that can be used to build a first feasible solution for iterative improvement methods such as the one described in the next section.

2.3 M-BFSL: A matheuristic of type BFSL

Matheuristics are efficient optimization procedures that take simultaneously advantage of the accuracy of exact methods and the scalability of heuristics. We suggest in this section a matheuristic of type BFSL called M-BFSL. Its general scheme is depicted in Figure 2.2.

From a feasible initial solution computed thanks to H-BFSL and denoted by X, the matheuristic improves the solution using a simulated annealing coupled with three different neighborhood moves.

The neighborhood moves first perturb the balancing solution (*balancing perturbation*) then performs optimal sequencing of the operations within each workstation (*operations sequencing*). The three neighborhood moves only differ by the *balancing perturbation*. The *operations sequencing* is always done thanks to the optimal dynamic programming algorithm described in subsection 2.2.2. In this way, a neighboring solution Y is computed.

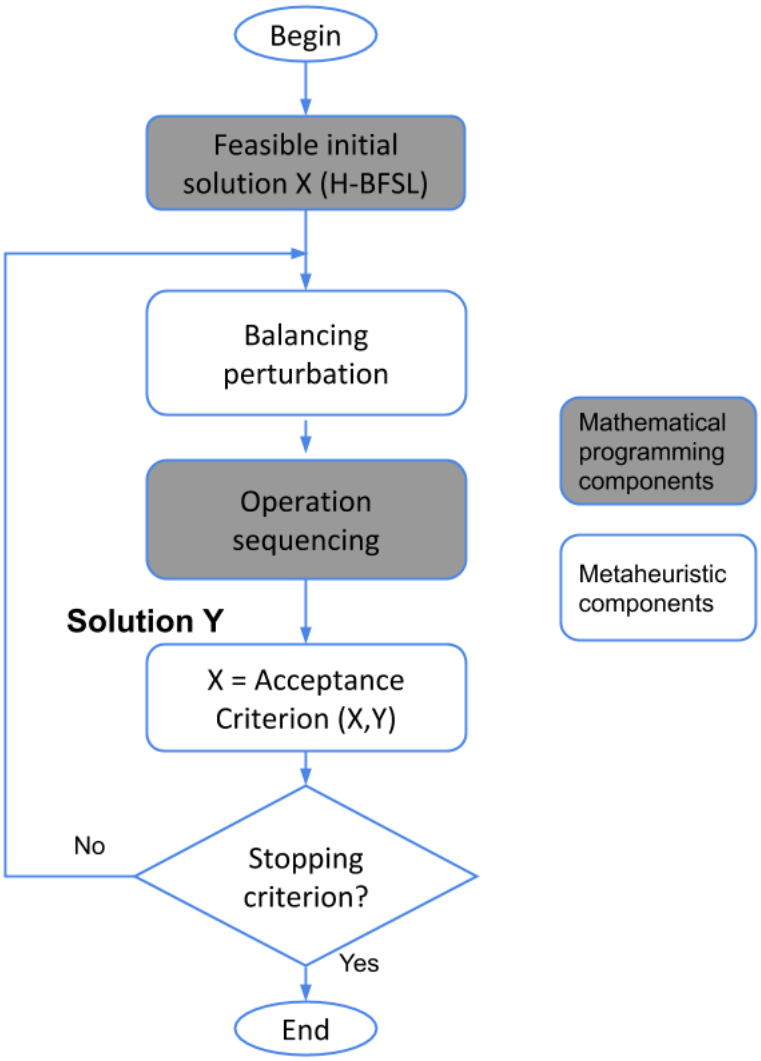

Figure 2.2: General scheme of the proposed matheuristic

An acceptance criterion is defined in order to choose which of X or its neighbor Y will be kept for the next iteration.

Moreover, we indicate in Figure 2.2 which components come from the field of mathematical programming, and which ones come from the field of metaheuristics.

We describe next the building blocks of the matheuristic, namely the encoding scheme, the neighborhood moves and the metaheuristic scheme.

2.3.1 Solution' encoding

A solution X is encoded as a list of sequences:

$$X = [\sigma_1, \sigma_2, \ldots, \sigma_p]$$

where σ_s is the sequence of operations allocated to workstation s. If n_s operations are allocated to workstation s, then σ_k is represented as follows:

$$\sigma_k = (\sigma_k^1, \sigma_k^2, \ldots, \sigma_k^{n_s})$$

where σ_k^i is the operation assigned to the i^{th} position in the sequence of workstation s.

2.3.2 Neighborhood moves

We use 3 neighborhood moves:

- V_1 Insertion (of an operation from a workstation to another workstation).
- V_2 Merger (of two workstations).
- V_3 Split (of one workstation to two workstations).

Insertion move V_1 aims to move an operation from a workstation to another. The source and the destination workstations are re-sequenced optimally using Algorithm 1. To respect precedence constraints, the destination workstation must be chosen carefully. We may need to displace more than one operation in order to respect inclusion constraints. The insertion move is described in Algorithm 3.

Algorithm 3 Principle algorithm for insertion move

INPUT: X, a feasible solution of the RTLBP.
OUTPUT: Y, a random neighbor of X with respect to insertion move

1: Select randomly and uniformly an operation i (the workstation of i is called the source workstation).
2: Select randomly and uniformly a destination workstation. This workstation is chosen between the last workstation containing an operation that precedes i and the first workstation containing an operation that succeeds i.
3: Move the operation i and all the operations that are linked by an inclusion constraint with operation i from the source workstation to the destination workstation.
4: Run the sequencing algorithm (Algorithm 1) in the source and destination workstations.

Example 2.3.1. *Let's consider the following solution with 8 operations and 3 workstations:* $X = [(3,5),(6,1,2),(4,7,8)]$. *We suppose that operations 1 and 2 are linked with an inclusion constraint. We give a random neighbor* $Y \in V_1(X)$ *as follows:* $Y = [(3,5),(6),(4,7,2,8,1)]$. *Operations 1 and 2 have been moved from workstation 2 to workstation 3. It is assumed that* $(4,7,2,8,1)$ *is the optimal sequencing of operations 1, 2, 4, 7 and 8, and that precedence constraints are respected.*

Merger move V_2 aims to merge two workstations into one. The number of workstations is reduced by one. It is equivalent to move all the operations of some source workstation to a destination workstation. The destination workstation must be chosen so as the precedence constraints are respected. The resulting workstation (destination) is re-sequenced optimally using Algorithm 1. The merger move is described in Algorithm 4.

Algorithm 4 Principle algorithm for merger move

INPUT: X, A feasible solution of the RTLBP.
OUTPUT: Y, A random neighbor of X with respect to merger move

1: Select randomly and uniformly a source workstation.
2: Select randomly and uniformly a destination workstation. This workstation is chosen between the last workstation containing an operation that precedes an operation of the source workstation and the first workstation containing an operation j that is preceded by an operation from the source workstation.
3: Move all the operations of the source workstation to the destination workstation.
4: Run the sequencing algorithm (Algorithm 1) in the destination workstation.

Example 2.3.2. *Let's consider the following solution with 8 operations and 4 workstations:* $X = [(3,5),(6,1,2),(4,7),(8)]$. *We give a random neighbor* $Y \in V_2(X)$ *as follows:* $Y = [(3,7,5,4),(6,1,2),(8)]$. *The first and the third workstations have been merged into the first workstation. It is assumed that* $(3,7,5,4)$ *is the optimal sequencing of operations 3, 4, 5 and 7, and that precedence constraints are respected.*

Split move V_3 aims to split one workstation into two consecutive workstations. An operation from the split workstation is taken as *pivot* to cut the workstation. Two resulting workstations are obtained, they are re-sequenced optimally using Algorithm 1. The split move is described in Algorithm 5.

Algorithm 5 Principle algorithm for split move

INPUT: X A feasible solution of the RTLBP.
OUTPUT: Y A random neighbor of X with respect to split move

1: Select randomly and uniformly an operation i (The workstation of i is called W).
2: Partition the workstation W into 2 workstations W_1, W_2: if an operation j is placed before i, it is placed with all its inclusive operations in W_1. All the remaining operations form W_2.
3: Run the sequencing algorithm (Algorithm 1) in the two resulting workstations (W_1 and W_2).

Example 2.3.3. *Let's consider the following solution with 8 operations and 2 workstations:* $X = [(3, 5, 6, 1, 2), (4, 7, 8)]$. *We give a random neighbor* $Y \in V_3(X)$ *as follows:* $Y = [(3, 5), (6, 2, 1), (4, 7, 8)]$. *The first workstation has been split into two workstations taking* 6 *as pivot.*

V_1, V_2 and V_3 have been chosen so that the accessibility property is achieved as shown by the following theorem:

Theorem 2.3.4. *(Accessibility) Given two solutions* X *and* Y *of the RTLBP,* Y *is accessible from* X *by applying a succession of moves within* V_1, V_2 *and* V_3.

Proof. To access $Y = [\sigma'_1, \sigma'_2, \ldots, \sigma'_q]$ from $X = [\sigma_1, \sigma_2, \ldots, \sigma_p]$, we can apply the following:

- If $p = q$, for s going from 1 to q, transform σ_s to σ'_s as follows:
  - Move the operations within $\sigma'_s - \sigma_s$ to σ_s by applying V_1.
  - Displace the operations within $\sigma_s - \sigma'_s$ from σ_s by applying V_1.
- If $p < q$, apply V_3 to build new workstations.
- If $p > q$, apply V_2 to delete workstations.

□

The previous theorem shows that the optimal solution can be accessible from any starting solution.

We note that the moves are performed in such way that:

- For V_1, V_2 and V_3: precedence and inclusion constraints are respected.
- For V_3: exclusion constraints are respected.

If the obtained solution does not respect the other constraints then a new solution is generated (Algorithm 6).

2.3.3 Adaptive simulated annealing

Simulated annealing is a widespread metaheuristic that has been applied successfully to a wide range of optimization problems (Van Laarhoven and Aarts (1987)). Besides, it is simple and offers enough diversification and intensification: two fundamental ingredients in a successful metaheuristic.

More precisely, we use an adaptive homogeneous simulated annealing algorithm to improve the solution returned by H-BFSL. The general scheme of the improvement method is depicted in Algorithm 6. In the latter, $H(X)$ stands for the number of machines used by the solution X.

The adaptive aspect is derived from the following: a random neighborhood move is chosen among $\{V_1,\ V_2\ ,V_3\}$ according to the following probability distribution $\{p_1 = \frac{w_1}{w_1+w_2+w_3}, p_2 = \frac{w_2}{w_1+w_2+w_3}, p_3 = \frac{w_3}{w_1+w_2+w_3}\}$ where w_1, w_2 and w_3 are weights associated respectively with V_1, V_2 and V_3. Each time a neighborhood move V_i gives a strictly improving solution, w_i is relatively increased.

The initialization of the different parameters (temperature and weights), the temperature cooling procedure, the weights update, and the stopping conditions are discussed in the experimentation chapter (Chapter 5).

Algorithm 6 Adaptive simulated annealing algorithm

INPUT: X A feasible solution of the RTLBP given by H-BFSL.
OUTPUT: $Best$ A local optimal solution of the RTLBP.

$Best := X$
Initialise the temperature T and the weights w_1, w_2 and w_3
while Stopping condition of outer loop is not met **do**
  while Stopping condition of inner loop is not met **do**
    repeat
      Select randomly a neighbourhood move $V_i, i \in \{1, 2, 3\}$ with probability $p_i = \frac{w_i}{w_1+w_2+w_3}$
      Generate randomly and uniformly a candidate solution Y: $Y = V_i(X)$
    until Y is feasible
    if $H(Y) \leq H(X)$ **then**
      Accept: $X = Y$
      if $H(Y) < H(Best)$ **then**
        $Best := X$
        Increase the weight w_i of V_i
      end if
    else
      Accept with probability $exp(-\frac{H(Y)-H(X)}{T})$: $X = Y$
    end if
  end while
  Decrease the temperature T
end while

2.4 Conclusion

H-BFSL, an heuristic algorithm of type Balance-First Sequence-Last has been suggested in this chapter. The algorithm relies on linear programming, constraint generation and dynamic programming. The balancing subproblem is solved thanks to an ILP, then the sequencing subproblem is solved thanks to dynamic programming. The two steps are piloted thanks to a constraint generation algorithm. An approximation ratio of 2 is proven.

M-BFSL, a matheuristic of type Balance-First Sequence-Last has been suggested in this chapter. H-BFSL is used to compute an initial feasible solution. The solution is improved thanks to an hybridization of an adaptive simulated annealing with dynamic programming. Three neighborhood moves are used. They perform a balancing perturbation followed by operations sequencing in the workstations.

The reverse approach : Sequence-First Balance-Last (for the RTLBP) and Sequence-First Balance-And-Select-Last (for the RALBP and the SDRALBP) methods, is investigated next for both reconfigurable transfer lines and robotic assembly lines. This approach is based on a novel algorithm called split solving the polynomial case where the giant sequence is given. To the best of our knowledge, this latter result is new. All the next chapter will be devoted to it. Split will be later used as a decoding procedure in a metaheuristic and in a branch and bound.

Chapter 3

A new polynomial split approach for line balancing problems

Contents

3.1 Introduction **66**
3.2 Split approach: an illustrative example **66**
3.3 RTLBP **68**
3.3.1 Auxiliary graph 69
3.3.2 A constrained shortest path 70
3.3.3 Illustrative example 71
3.4 RALBP **75**
3.4.1 RALBP-1/SDRALBP-1 75
3.4.1.1 Auxiliary graph 76
3.4.1.2 A shortest path 76
3.4.1.3 Illustrative example 77
3.4.2 RALBP-2/SDRALBP-2 79
3.4.2.1 Auxiliary graph 79
3.4.2.2 A constrained minmax path 79
3.4.2.3 Illustrative example 80
3.4.2.4 Extension to one robot per type 83
3.5 Conclusion **87**

3.1 Introduction

In Borisovsky, Delorme, and Dolgui (2013), Delorme, Malyutin, and Dolgui (2016), Levitin, Rubinovitz, and Shnits (2006) and Nilakantan et al. (2015), a particular case is identified: the case where an overall sequence of operations is imposed.

This chapter is intended to solve this particular case of a given giant sequence thanks a novel polynomial algorithm called split.

Split is inspired to us from an homonym method developed by Beasley (1983) for a particular case of the Vehicle Routing Problem (VRP). The VRP is concerned with assigning a set of costumers to a set of vehicles initially located within a depot. The objective is to minimize the total distance traveled by the vehicles. Beasley considers a fixed "giant tour" of costumers. He proves that finding an optimal solution respecting a given giant tour is polynomial. It is equivalent to a shortest path problem in some auxiliary graph. Prins (2004) has then used split as decoding procedure in a metaheuristic which led to the best known method for the VRP for many years. Similar researches were conducted afterwards: Prins, Lacomme, and Prodhon (2014).

Split is an efficient method in a Sequence-First Cluster-Second approach, because it allows to optimally determine the clusters, given an initial giant tour. We propose here an original way to adapt ideas from VRP to balancing problems. In this chapter, split is used to solve the particular case of a given giant sequence for various balancing problems. Then, it is used in approaches of type Sequence-First Balance-Last and Sequence-First Balance-And-Select-Last in the next chapter.

In the next sections, split is first described for SALBP-2 then it is derived for the considered balancing problems. For each of these problems, we describe the auxiliary graph, give a polynomial algorithm to solve the path problem and give a small illustrative example.

3.2 Split approach: an illustrative example

We consider the SALBP-2 to illustrate the split approach in this section. SALBP-2 is concerned with minimizing the cycle time while respecting a maximum number of workstations (s_{max}). Before presenting the general scheme of split, we give some definitions.

Definition 3.2.1. *(Giant sequence) Given an instance with n operations, a giant sequence is a permutation of all the operations:* $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ *where* σ_i *is the operation at position i of the sequence.*

Definition 3.2.2. *(A solution satisfying a giant sequence) A solution X is said to satisfy a giant sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ if for all σ_i, σ_j such that $i < j$ either σ_i and σ_j are assigned to the same workstation in X or the workstation to which σ_i is assigned, is situated before the workstation to which σ_j is assigned.*

Definition 3.2.3. *(Compatible giant sequence) A giant sequence σ is said to be compatible with respect to an instance if there exists at least one feasible solution satisfying σ.*

Given a giant sequence σ and an instance $\mathcal{I}$, the optimal solution satisfying σ can be obtained thanks to the search of an optimal path in an appropriate auxiliary graph denoted $\mathcal{H}_{\mathcal{I}}(\sigma)$. We describe next the construction of $\mathcal{H}_{\mathcal{I}}(\sigma)$.

Auxiliary graph

The graph $\mathcal{H}_{\mathcal{I}}(\sigma) = (V, A)$ is a directed graph, composed of the set of nodes V and the set of arcs A. The set of nodes is given by the set of operations plus an additional node corresponding to a fictitious operation, i.e. $V = N \cup \{0\}$. An arc (i, j) from operation i to operation j refers to a workstation to which the subsequence $(i+1, i+2, \ldots, j)$ is assigned. A is composed of all the arcs (i, j) such that $i < j$. Besides, $\mathcal{H}_{\mathcal{I}}(\sigma)$ is weighted. For the SALBP-2, the weight of the arc (i, j) is given by:

$$c_{i,j} = \sum_{k=i+1}^{j} d_k$$

$c_{i,j}$ corresponds to the time required to perform the subsequence $(i+1, \ldots, j)$, i.e., the workload of the corresponding workstation.

Optimal path

In the graph thus constructed, a path from the node 0 to the node σ_n corresponds to a solution of the problem.

Indeed, a path $\{(0, \sigma_{o_1}), (\sigma_{o_1}, \sigma_{o_2}), (\sigma_{o_2}, \sigma_{o_3}), \ldots, (\sigma_{o_{k-1}}, \sigma_{o_k}), (\sigma_{o_k}, \sigma_n)\}$ corresponds to the solution where the subsequence $(\sigma_1, \sigma_2, \ldots, \sigma_{o_1})$ is assigned to the first workstation, $(\sigma_{1+o_1}, \ldots, \sigma_{o_2})$ is assigned to the second workstation,... and $(\sigma_{1+o_k}, \ldots, \sigma_n)$ is assigned to the last workstation ($k + 1^{th}$ workstation). More precisely, there is a one-to-one correspondence between the space of feasible solutions satisfying the giant sequence σ and the paths from 0 to σ_n in $\mathcal{H}_{\mathcal{I}}(\sigma)$. Besides, the objective value of a solution (the cycle time for SALBP-2) matches the maximum weight of an arc in the corresponding path in $\mathcal{H}_{\mathcal{I}}(\sigma)$. Consequently, an optimal solution respecting a giant sequence σ can be obtained by computing the path from 0 to σ_n in $\mathcal{H}_{\mathcal{I}}(\sigma)$

containing an arc with maximum weight and using no more than s_{max} arcs. This problem corresponds to the search for a minmax path (Chechik et al. (2016)), it can be solved thanks to polynomial algorithm described later.

Remark 2. *To solve SALBP-1 for a given giant sequence, the following algorithm is optimal:*

(1) $s = 1$ *(open an initial workstation)*

(2) For $i = 1$ *to* n

(a) If $(w_s + d_{\sigma_i} \leq C)$ *(workload of* s *+ processing time of* $i \leq$ *cycle time* C*):*

i. Add the operation σ_i *to* s

(b) else:

i. Open a new workstation: $s = s + 1$

(3) EndFor

The latter algorithm is not applicable for SALBP-2 or any type 2 problem since the cycle time remains unknown for those problems. Besides, it is no more optimal for type 1 problems considering either sequence-dependent setup times, robot selection or any industrial case. In any of those cases, the greedy heuristic can be used as an heuristic as it is the case in Borisovsky, Delorme, and Dolgui (2013). We suggest in this chapter an optimal polynomial algorithm to solve the problem that is applicable for all these situations.

Remark 3. *Throughout the remainder of this chapter, we supposed that a giant sequence of operations* σ *is given. To simplify the description of the algorithms, we suppose without loss of generality that* $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n) = (1, 2, \ldots, n)$*. We also suppose that* σ *respects precedence constraints. If* σ *is compatible, split returns an optimal solution respecting* σ*, otherwise split does not return a solution. Thus, split can be used to test the compatibility of a giant sequence respecting precedence constraints.*

Each of the following subsections is dedicated to the adaptation of split to a different problem.

3.3 RTLBP

We adapt in this section split for the RTLBP with the calculation of the arcs weight in the auxiliary graph and the resolution of the underlying optimization problem.

3.3.1 Auxiliary graph

$\mathcal{H}_{\mathcal{I}}(\sigma)$ is a weighted directed graph with the same set of nodes than SALBP-2 (set of operations plus an additional fictitious vertex 0).

The weight of an arc (i, j) is given by:

$$c_{i,j} = \begin{cases} \left\lceil \frac{\sum_{k=i+1}^{j} d_k + (\sum_{k=i+1}^{j-1} t_{k,k+1}) + t_{j,i+1}}{C} \right\rceil & \text{If } j > i+1 \text{ (at least 2 operations)} \\ \left\lceil \frac{d_{i+1}}{C} \right\rceil & \text{If } j = i+1 \text{ (one operation: no setup)} \end{cases}$$

which could be interpreted as the number of machines necessary to perform the subsequence $i+1,\ldots,j$ such that the given cycle time is respected.

The set of arcs (i, j) is composed of all the arcs (i, j) such that $i < j$. However, some of the arcs must be deleted if the constraints are violated. All the cases are listed below:

- **The maximum number of operations per workstation is exceeded:** An arc (i, j) violating the maximum number of operations per workstation constraint can be detected by the following condition:

$$j - i > N_{max}$$

- **The maximum number of machines per workstation is exceeded:** Likewise, an arc (i, j) violating the maximum number of machines per workstation constraint can be detected by the following condition:

$$c_{i,j} = \left\lceil \frac{\sum_{k=i+1}^{j} d_k + (\sum_{k=i+1}^{j-1} t_{k,k+1}) + t_{j,i+1}}{C} \right\rceil > M_{max}$$

- **Inclusion constraints are not satisfied:** An arc (i, j) violating the inclusion constraints can be detected by the following condition:

$$\exists (a, b) \in I \ / \ \mathbb{1}_{\{i+1,\ldots,j\}}(a)^1 + \mathbb{1}_{\{i+1,\ldots,j\}}(b) = 1$$

- **Exclusion constraints are not satisfied:** An arc (i, j) violating the exclusion constraints can be detected by the following condition:

$$\exists ES \in E \ / \ \sum_{a \in ES} \mathbb{1}_{\{i+1,\ldots,j\}}(a) = |ES|$$

[1] $\mathbb{1}_X(a)$ **is the indicator function, equals 1 if** $a \in X$ **and 0 otherwise.**

- **Accessibility constraints are not satisfied:** An arc (i, j) violating the accessibility constraints can be detected by the following condition:

$$\cap_{k\in\{i+1,\dots,j\}} Pos_k = \emptyset$$

3.3.2 A constrained shortest path

Finding an optimal balancing respecting the giant sequence σ is equivalent to finding the shortest path between 0 and σ_n in the graph $\mathcal{H}_{\mathcal{I}}(\sigma)$, of length lower than (or equal to) s_{max} arcs (i.e. the shortest path constrained not to exceed s_{max} arcs). This is because an arc stands for a workstation and the solution cannot use more than s_{max} workstations. The existence of such a path between the starting node and the ending node implies the compatibility of the giant sequence. If there is no path having s_{max} arcs or less between 0 and σ_n in $\mathcal{H}_{\mathcal{I}}(\sigma)$, then there exists no feasible balancing solution respecting the giant sequence σ. We note that the more the problem is constrained, the less arcs are contained in the graph and the easier is the problem to solve.

We propose Algorithm 7 in order to compute a shortest path constrained not to exceed s_{max} arcs in the graph $\mathcal{H}_{\mathcal{I}}(\sigma)$. The algorithm uses labels on nodes to encompass information on the system state. The labels keep track of the number of machines and the number of arcs (workstations). A label l is representative of a partial solution, it has two components:

$$l = (a, b)$$

where a and b denote respectively the number of machines and the number of workstations used so far by the partial solution.

We define a set of labels L_i for each node i. Every label in L_i corresponds to a path (partial solution) between 0 and i.

The algorithm uses a dominance rule in order to avoid combinatorial explosion:

Definition 3.3.1. *(Dominance rule)* (a, b) *is dominated by* (a', b') *if:*

$$a' \leq a \text{ and } b' \leq b$$

Algorithm 7 starts with fictitious node 0 labeled $\{(0, 0)\}$ (i.e. $L_0 := \{(0, 0)\}$) and continues with the other nodes following the order of the giant sequence. For every node t and every label $(a_t, b_t) \in L_t$, the algorithm explores every outgoing arc (t, i) and tries to propagate it (i.e. add a label to the set of labels of node i denoted L_i) if:

$(a_t + c_{t,i}, b_t + 1)$ is not *dominated* by a label of L_i (Propagation rule)

If so, the label $(a_t + c_{t,i}, b_t + 1)$ is added to L_i and all labels dominated by $(a_t + c_{t,i}, b_t + 1)$ are deleted from L_i. The shortest path cost is stored in a^*.

Lemma 3.3.2. *The dominance rule limits the number of labels per node to s_{max}.*

Proof. Suppose by contradiction that we have more than s_{max} labels for some node i. Then, necessarily there must exist two labels $(a, b), (a', b) \in L_i$ (i.e. with same number of workstations), because for every label (x, y) we have:

$$y \in \{1, 2, \ldots, s_{max}\}$$

The coexistence of (a, b) and (a', b) is impossible due to the dominance rule.

□

Theorem 3.3.3. *The algorithm runs in $O(n^4)$ where n is the number of operations.*

Proof. The algorithm performs dominance tests for each 3-uplet (label of origin node, arc, label of destination node). Thus, it runs in $O(m.s_{max}^2)$ where m is the number of arcs in the graph. Since $s_{max} \leq n$ and $m \leq n + (n-1) + ... + 1 = \frac{n(n+1)}{2}$, split runs in $O(n^4)$. □

Remark 4. *The tests of lines 11 and 13 can both be done in a single scan of the list L_i by exploiting the transitivity of the dominance relation: we browse the list L_i and as soon as a label is dominated by (a_i, b_i) we can conclude that the condition of line 11 is satisfied and thus add (a_i, b_i) to L_i and delete all the remaining labels of L_i that are dominated by (a_i, b_i).*

3.3.3 Illustrative example

We recall the RTLBP example introduced in the first chapter:

- A part requires the execution of 7 operations numbered from 1 to 7 ($n = 7$).
- At most 4 workstations can be used ($s_{max} = 4$).
- Precedence relations are represented in Figure 3.1.

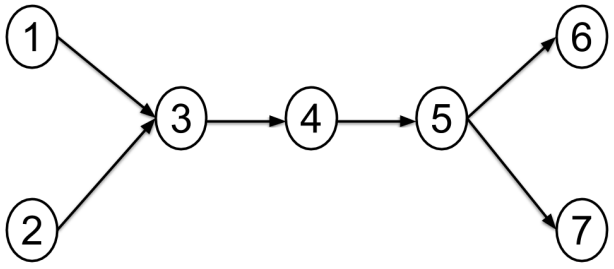

Figure 3.1: Precedence graph.

Algorithm 7 Split algorithm for the RTLBP

INPUT ($\mathcal{I}$,σ) where $\mathcal{I}$ is an instance of the RTLBP and σ is a giant sequence respecting precedence constraints.
OUTPUT X: An optimal solution (with the minimal number of machines) respecting σ if there exists a feasible solution

Build the graph $\mathcal{H}_{\mathcal{I}}(\sigma)$
$L_0 := \{(0,0)\}$
for t=1 **to** n **do**
  $L_t := \emptyset$
end for
for t=0 **to** n-1 **do**
  for all $(t,i) \in A$ *(Propagate labels from L_t)* **do**
    for all $(a_t, b_t) \in L_t$ **do**
      if ($b_t < s_{max} - 1$ or $i = n$) **then**
        $(a_i, b_i) := (a_t + c_{t,i}, b_t + 1)$
        if (a_i, b_i) is not dominated by an element of L_i **then**
          $L_i := L_i \cup \{(a_i, b_i)\}$
          if (a_i, b_i) dominates some elements $(a'_i, b'_i) \in L_i$ **then**
            $L_i := L_i \backslash \{(a'_i, b'_i)\}$
          end if
        end if
      end if
    end for
  end for
end for
if $L_n \neq \emptyset$ **then**
  $a^* := Min_{(a_i,b_i)\in L_n}(a_i)$
  Decode the path of length a^* to build X
end if

- $N_{max} = 3$, maximum number of operations to be assigned to a workstation.
- $M_{max} = 3$, maximum number of machines to be hosted by a workstation.
- The processing times and setup times are given respectively in Tables 3.1, 3.2.

i	1	2	3	4	5	6	7
d_i	1.5	1	3.5	1.5	2.5	3	1

Table 3.1: Processing times.

$t_{i,j}$	$j = 1$	2	3	4	5	6	7
$i = 1$	0	0.5	1	1	1	1	1
2	1	0	0.5	1	1	1	1
3	0.5	1	0	1	1	1	1
4	1	1	1	0	0.5	1	1
5	1	1	1	0.5	0	1	1
6	1	1	1	1	1	0	0.5
7	1	1	1	1	1	0.5	0

Table 3.2: Setup times

- $C = 2.5$, cycle time.
- Inclusion and exclusion constraints are given by $I = \{(1, 2)\}, E = \{\{5, 6\}\}$.
- Accessibility constraints are given as follows: $Pos = \{1, 2, 3, 4\}$, $Pos_4 = \{1, 2\}$, $Pos_5 = \{3, 4\}$, $Pos_i = \{1, 2, 3, 4\}, \forall i \in \{1, 2, 3, 6, 7\}$

We consider the giant sequence $\sigma = (1, 2, 3, 4, 5, 6, 7)$. An illustration of the graph $\mathcal{H}_{\mathcal{I}}(\sigma)$ is given in Figure 3.2.

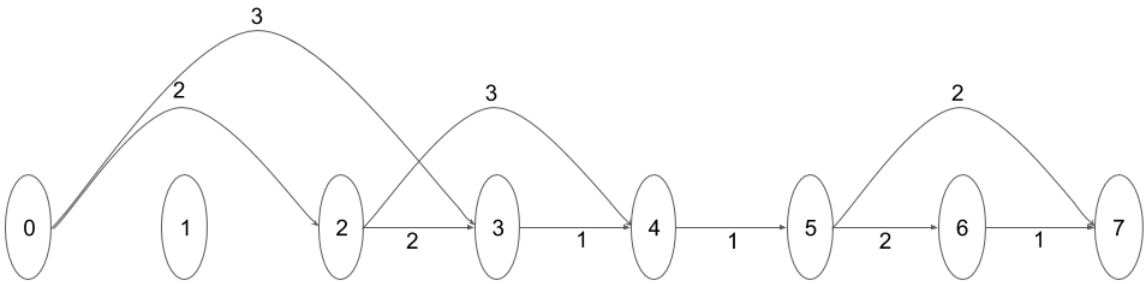

Figure 3.2: Auxiliary graph

Arcs violating the considered constraints have been deleted. For example, arcs (0,1), (1,2) and (1,3) are deleted due to the violation of the inclusion constraint "(1,2)". The arc (3,5) is deleted due to the violation of the accessibility constraint: operations 4 and 5 do not share a common position. Arc (4,6) is deleted due to the violation of the exclusion constraint "{5,6}". Arcs (0,4), (0,5), (0,6) and (0,7) are deleted due to exceeding the maximum number of machines (M_{max}) and operations (N_{max}) per workstation.

Remark 5. *The graph could be disconnected despite the existence of a balancing solution: in the example of Figure 3.2, the disconnectivity comes from the fact that operations 1 and 2 must be assigned to the same workstation due to the inclusion constraint.*

In Figure 3.3, we represent in red the shortest path not exceeding 4 workstations. The corresponding optimal solution is given in the figure. The path is of length 7, i.e. 7 machines and has 4 arcs, i.e. 4 workstations.

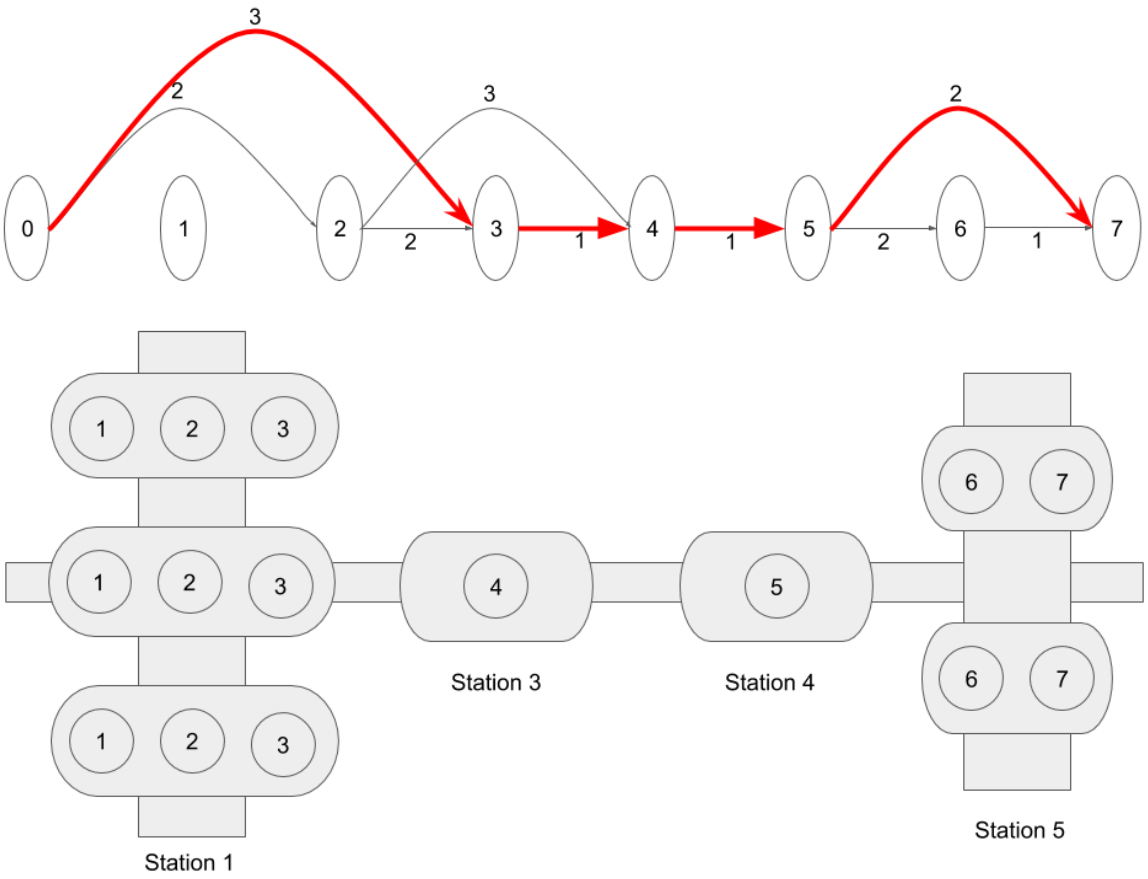

Figure 3.3: Constrained shortest path and corresponding solution.

3.4 RALBP

The cases of RALBP-1 and RALBP-2 are respectively similar to SDRALBP-1 and SDRALBP-2. It suffices to consider that the setup times are null. For this reason, we are content to describe SDRALBP-1 and SDRALBP-2.

3.4.1 RALBP-1/SDRALBP-1

We consider in this subsection SDRALBP-1. Many types of robots are available to perform the operations. The same type of robot can be selected for multiple workstations without any limitation. The cycle time C is fixed and the number of workstations is to be minimized.

3.4.1.1 Auxiliary graph

We remove from A all the arcs (i, j) such that the workload induced by the subsequence $(i + 1, i + 2, \ldots, j)$ on all the types of robots exceeds the given cycle time. In other words, we remove all the arcs $(i, j), i < j$ satisfying the following condition:

$$Min_{r\in R} \sum_{k=i+1}^{j} d_k^r + \sum_{k=i+1}^{j-1} t_{k,k+1}^r + t_{j,i+1}^r > C$$

Besides, all the weights of the arcs are equal to 1: $\forall\ (i, j) \in A,\ c_{i,j} = 1$.

Theorem 3.4.1. *The construction of the auxiliary graph can be done within a time complexity of $O(n^3.n_r)$.*

Proof. For a couple of operations (i, j) such that $i < j$, the workload $Min_{r\in R} \sum_{k=i+1}^{j} d_k^r + \sum_{k=i+1}^{j-1} t_{k,k+1}^r + t_{j,i+1}^r$ can be computed in $O(n.n_r)$ and there are $O(n^2)$ couples. □

3.4.1.2 A shortest path

Since we want to minimize the number of workstations, the underlying problem is a shortest path problem. The shortest path from 0 to σ_n can be obtained thanks to Algorithm 8 (it can be seen as a Bellman algorithm), it runs in $O(|A|) = O(n^2)$ (without taking into account the construction of the graph).

Algorithm 8 Split for SDRALBP-1

INPUT ($\mathcal{I}$,σ) where $\mathcal{I}$ is an instance of the SDRALBP problem and σ is a giant sequence respecting precedence constraints.
OUTPUT X: An optimal solution among solutions satisfying σ.

```
Build the graph H_I(σ)
l_0 := 0
for t = 1 to n do
    l_t := ∞
end for
for t = 0 to n − 1 do
    for all (t, i) ∈ A (Propagate arcs from t) do
        l_i := Min(l_i, l_t + 1)
    end for
end for
Decode the path of length l_n to build up X.
```

Algorithm 8 stores the distance from node 0 to the node i in l_i. The initialization is done in lines 2-5. Then for each node t, following the order of the giant sequence, and for each outgoing arc (t, i), l_i is updated if $l_t + 1 < l_i$. As a result of this, l_n stores the length of the shortest path from 0 to σ_n. The instruction of line 11 can easily be done by firstly retrieving the path of length l_n then by allocating the subsequence $(i + 1, \ldots, j)$ corresponding to the k^{th} arc of the shortest path to the k^{th} workstation. Finally, the type of robot that minimizes the workload is chosen for each workstation.

3.4.1.3 Illustrative example

We illustrate the split for SDRALBP-1 thanks to a small example. We consider an instance with 5 operations ($n = 5$) and 4 types of robots ($n_r = 4$).

The precedence relations, the processing times and the sequence-dependent setup times are represented respectively in Figure 3.4, Table 3.4 and Table 3.4.

The given cycle is $C = 6$.

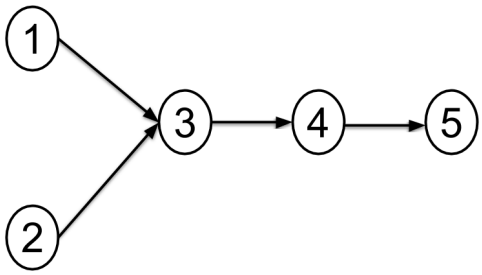

Figure 3.4: Precedence graph.

	i	1	2	3	4	5
d_i^r	$r = 1$	1.5	1	3.5	1.5	2.5
	$r = 2$	1.5	0.5	3	2	1.5
	$r = 3$	1	1	3	2	1
	$r = 4$	2	2	4	1.5	2

Table 3.3: Processing times.

(a) Setup times on robots of type 1

	i	1	2	3	4	5
$t^1_{i,j}$	$j=1$	0	1	0.5	1.5	0.5
	$j=2$	1.5	0	0	2	0.5
	$j=3$	1	1	0	2	1
	$j=4$	0.5	0.5	0.5	0	0.5
	$j=5$	1.5	0.5	0.25	0.5	0

(b) Setup times on robots of type 2

	i	1	2	3	4	5
$t^2_{i,j}$	$j=1$	0	0.5	0.5	1	0.25
	$j=2$	1.5	0	0.25	0.25	0.25
	$j=3$	0.25	1	0	0.5	1
	$j=4$	0.25	2	4	0	2
	$j=5$	1	1	1	2	0

(c) Setup times on robots of type 3

	i	1	2	3	4	5
$t^3_{i,j}$	$j=1$	0	1	0.5	1.5	0.75
	$j=2$	0.75	0	0.5	2	1.5
	$j=3$	1	1	0	2	1
	$j=4$	2	0.5	0.75	0	0.5
	$j=5$	1	0.5	0.25	1.5	0

(d) Setup times on robots of type 4

	i	1	2	3	4	5
$t^4_{i,j}$	$j=1$	0	1	0.75	0.5	0.5
	$j=2$	1.5	0	3	2	1.5
	$j=3$	1	1	0	2	1
	$j=4$	2	1	0.5	0	2
	$j=5$	1.5	0.5	0.5	2	0

Table 3.4: Sequence-dependent setup times

We consider the fixed sequence $\sigma = (1, 2, 3, 4, 5)$. It respects precedence constraints. $\mathcal{H}_{\mathcal{I}}(\sigma)$ is represented in Figure 3.5. In this graph, the following arcs have been deleted because they correspond to violated constraints:

$$\{(0, 4), (0, 5), (1, 4), (1, 5), (2, 5)\}$$

For all these arcs, the minimum workload exceeds the given cycle time. A shortest path is shown in red on the figure. It represents a solution with 2 workstations.

Subsequence $(1, 2, 3)$ is assigned to the first workstation and $(4, 5)$ is assigned the second workstation.

The type of robot assigned to the first workstation is given by $Argmin_{r\in R}\ d^r_1 + t^r_{1,2} + d^r_2 + t^r_{2,3} + d^r_3 + t^r_{3,1} = 2$. The type of robot assigned to the second workstation is given by $Argmin_{r\in R}\ d^r_4 + t^r_{4,5} + d^r_5 + t^r_{5,4} = 3$.

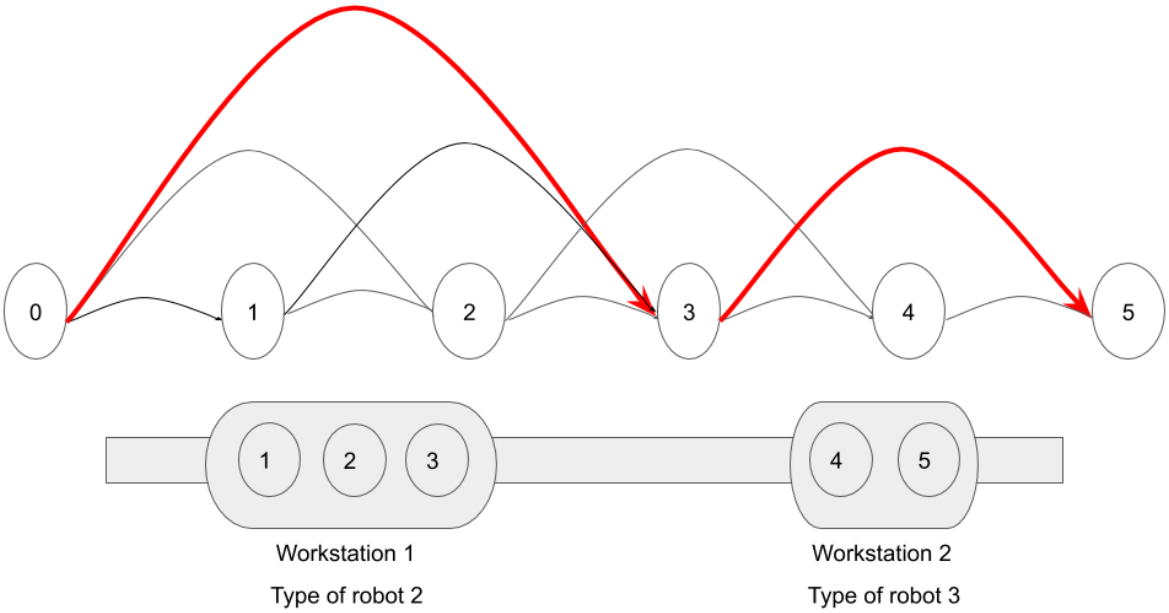

Figure 3.5: Auxiliary graph and optimal solution for SDRALBP-1

3.4.2 RALBP-2/SDRALBP-2

We adapt split to SDRALBP-2. In this case, we have a given maximum number of workstations s_{max} and the cycle time C is to be minimized. The RALBP-2 case can be derived by considering that the setup times are null.

3.4.2.1 Auxiliary graph

We only detail the steps that differ from SDRALBP-1. The set of nodes V of the graph $\mathcal{H}_{\mathcal{I}}(\sigma) = (V, A)$ does not change. The set of arcs A is given by all the arcs (i, j) such that $i < j$. Besides, $\mathcal{H}_{\mathcal{I}}(\sigma)$ is weighted. The weight of the arc (i, j) is given by:

$$c_{i,j} = Min_{r \in R} \{ \sum_{k=i+1}^{j} d_k^r + \sum_{k=i+1}^{j-1} t_{k,k+1}^r + t_{j,i+1}^r \}$$

$c_{i,j}$ corresponds to the minimum time required to perform the subsequence $(i + 1, \ldots, j)$. We point out that we could take any robot that minimizes the workload. The auxiliary graph can be constructed in $O(n^3.n_r)$.

3.4.2.2 A constrained minmax path

An optimal solution respecting the giant sequence σ can be obtained by computing a path from 0 to σ_n in $\mathcal{H}_{\mathcal{I}}(\sigma)$ that minimizes the maximum weight of an arc and

that uses no more than s_{max} arcs. In graph theory, the problem is known as a *bottleneck path* or *min-max path* (Chechik et al. (2016)) which consists in finding a path between a pair of vertices such as the weight of an arc of maximum weight is minimized. This problem is polynomial since a shortest path algorithm can be adapted to compute a min-max path. We are dealing with a constrained min-max path because the path should not exceed s_{max} arcs.

We propose Algorithm 9 in order to compute a shortest path constrained not to exceed s_{max} arcs in the graph $\mathcal{H}_{\mathcal{I}}(\sigma)$. The algorithm is based on labels that keep track of the cycle time and the number of arcs. As for the RTLBP, a label $l = (a, b)$ represents a partial solution. In the case of the SDRALBP-2, a and b denote respectively the number of machines and the number of workstations used so far by the partial solution.

The same dominance rules as for RTLBP are used here. Only the propagation rule differs. The propagation of label (a_t, b_t) through the arc (t, i) gives the label $(Max(a_t, c_{t,i}), b_t + 1)$: the maximum between a_t and $c_{t,i}$ gives the cycle time, the number of workstations b_t is incremented by 1.

We can prove similarly that the algorithm runs in $O(n^4)$.

3.4.2.3 Illustrative example

We illustrate the split for SDRALBP-2 thanks to a small example. We consider the same instance as the previous subsection. Since we deal with SDRALBP-2, we must give the maximum number of workstations: $s_{max} = 3$.

We consider the same fixed giant sequence $\sigma = (1, 2, 3, 4, 5)$. $\mathcal{H}_{\mathcal{I}}(\sigma)$ is represented in Figure 3.6.

Algorithm 9 Split for SDRALBP-2

INPUT ($\mathcal{I}$,σ) where $\mathcal{I}$ is an instance of the SDRALBP-2 problem and σ is a giant sequence respecting precedence constraints.

OUTPUT X: An optimal solution (with the minimal cycle time C^*) respecting σ if there exists a feasible solution

Build the graph $\mathcal{H}_{\mathcal{I}}(\sigma)$
$L_0 := \{(0,0)\}$
for t=1 **to** n **do**
  $L_t := \emptyset$
end for
for t=0 **to** n-1 **do**
  for all $(t,i) \in A$ *(Propagate labels from L_t)* **do**
    for all $(a_t, b_t) \in L_t$ **do**
      if ($b_t < s_{max} - 1$ or $i = n$) **then**
        $(a_i, b_i) := (Max\{a_t, c_{t,i}\}, b_t + 1)$
        if (a_i, b_i) is not dominated by an element of L_i **then**
          $L_i := L_i \cup \{(a_i, b_i)\}$
          if (a_i, b_i) dominates some element $(a'_i, b'_i) \in L_i$ **then**
            $L_i := L_i \backslash \{(a'_i, b'_i)\}$
          end if
        end if
      end if
    end for
  end for
end for
if $L_n \neq \emptyset$ **then**
  $C^* := Min_{(a_i,b_i)\in L_n}(a_i)$
  Decode the path of cost C^* to build X
end if

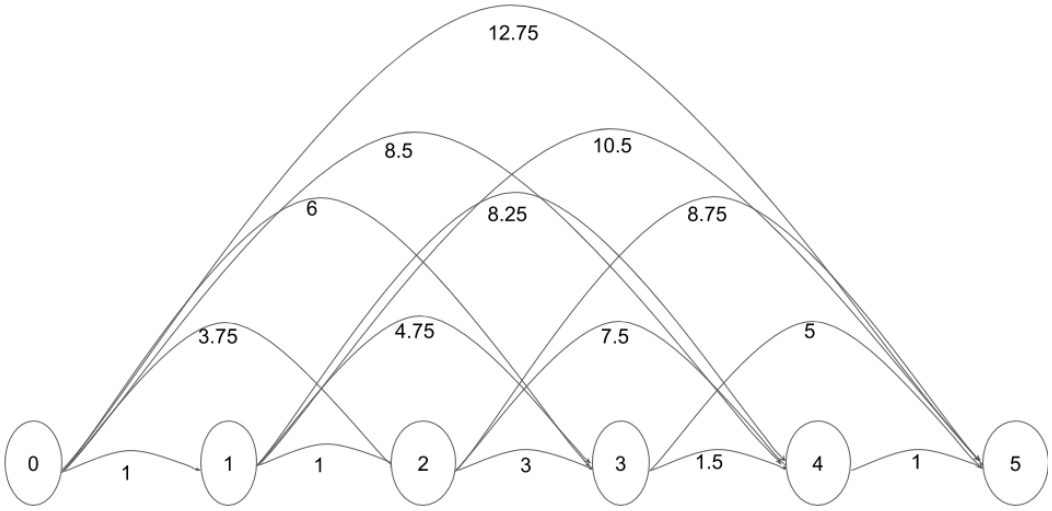

Figure 3.6: Auxiliary graph for SDRALBP-2

A min-max path not exceeding 3 arcs is shown in red in Figure 3.7. It represents a solution with 3 workstations. Subsequence $(1, 2)$ is assigned to the first workstation, (3) is assigned the second workstation and $(4, 5)$ is assigned the third workstation.

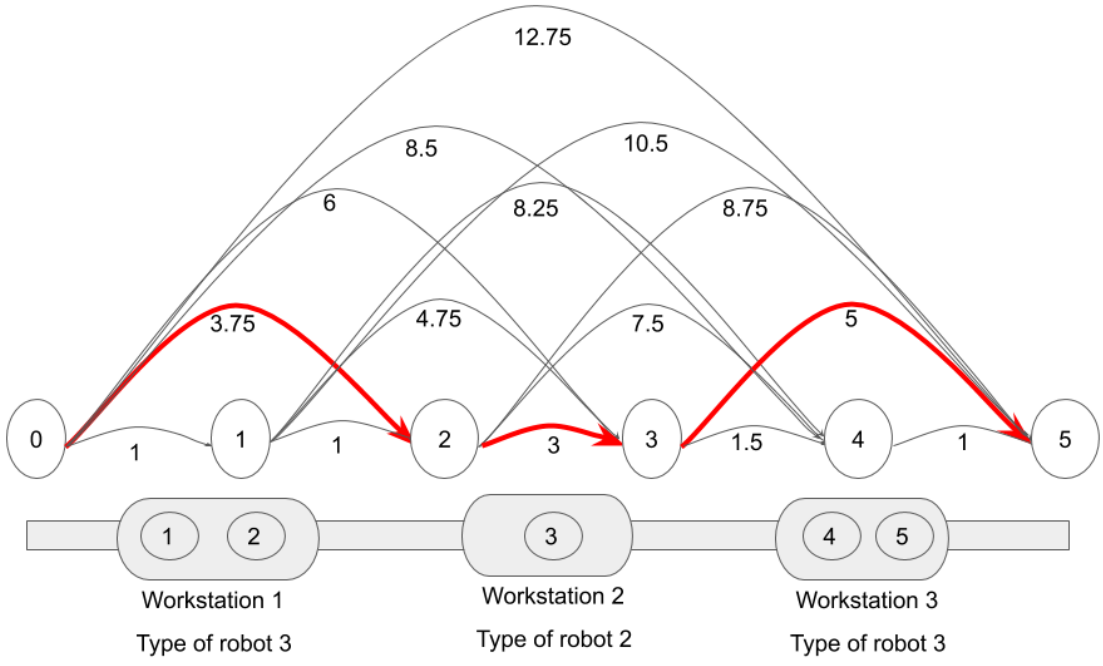

Figure 3.7: Min-max path not exceeding 3 arcs and corresponding solution.

The type of robot assigned to the three workstations are given respectively by $Argmin_{r\in R}\ d_1^r + t_{1,2}^r + d_2^r + t_{2,1}^r = 3$, $Argmin_{r\in R}\ d_3^r = 2$ and $Argmin_{r\in R}\ d_4^r + t_{4,5}^r + d_5^r + t_{5,4}^r = 3$.

We notice that this solution is no more feasible if we suppose the assumption "only one robot per type" because robot type 3 is used twice (in workstation 1 and 3). We'll see in the next subsection how to adapt split to this assumption.

3.4.2.4 Extension to one robot per type

We adapt in this subsection split for the assumption "one robot per type". We assume that each robot can be assigned to at most one workstation. In other words, we no longer allow the possibility of assigning several robots of the same type to different workstations. In this context, the input is slightly changed. Instead of having a set of types of robots, each of which can be assigned to multiple workstations, we now have a set of robots, each of which can be assigned at most to one workstation. We keep the same notations R and n_r respectively for the set of robots and the number of robots. All other components of the SDRALBP (input, output, decision to be maid) are preserved. This assumption is more general than the previously studied and make the problem more difficult as seen in the complexity section in the first chapter (section 1.4.1.4). We will see in this subsection that *split* can be adapted for this assumption. But since it is harder, the simple graph approach is no longer relevant and we have to consider multi-graphs to model the problem. Besides, the polynomiality of split is lost. For this reason, heuristics and bounds are used to speed up split.

Auxiliary graph

$\mathcal{H}_{\mathcal{I}}(\sigma) = (V, A)$ is a directed weighted multi-graph defined by its set of vertices V, its multiset[2] of arcs A. The set of vertices is given by $V = N \cup \{0\}$, 0 being a fictitious operation and N being the set of operations numbered from 1 to n. As it is a multi-graph, the same arc (same source and same destination) can be present more than once in the graph. A contains n_r arcs (i, j) such that $i < j$, the same occurrences of an arc (i, j) are labeled from 1 to n_r and we denote $(i, j)_r \in A$ the occurrence number r of arc (i, j), i.e, the occurrence representing the robot r. The arc $(i, j)_r$ represents a workstation to which the operations from $i + 1$ to j are assigned and are performed by the robot r. The weight of an arc $(i, j)_r$ is given by:

$$w((i, j)_r) = \sum_{k=i+1}^{j} d_k^r + \sum_{k=i+1}^{j-1} t_{k,k+1}^r + t_{j,i+1}^r$$

[2]In a multiset, an element can be present more than once, the multiplicity defines the number of copies of an element in the set

which represents the time required to perform the sub-sequence $(i+1,\ldots,j)$ on robot r.

A constrained shortest path

In this way, a path from 0 to n in $\mathcal{H}_{\mathcal{I}}(\sigma)$ is representative of a feasible solution provided the path does not pass through two arcs $(i,j)_r$, $(i',j')_{r'}$ such that $r = r'$ to respect the constraint raised by the assumption "only one robot per type". The related problem becomes a constrained min-max path. This latter consists in finding a path from 0 to n in $\mathcal{H}_{\mathcal{I}}(\sigma)$ that minimizes the maximum weight of an arc constrained not to:

- Exceed s_{max} arcs.
- Go through two arcs $(i,j)_r$, $(i',j')_{r'}$ such that $r = r'$.

We suggest Algorithm 10 to compute such a path.

The labels used in Algorithm 10 differ from the previous versions of split. Each node i of the graph $\mathcal{H}_{\mathcal{I}}(\sigma)$ is associated with a list of labels L_i. A label $(a_i, b_i, \delta_i) \in L_i$ represents a path between the fictitious node 0 and the node i modeling a partial solution of b_i workstations and a cycle time of a_i. $\delta_i : R \longrightarrow \{0,1\}$ is a function that defines the robots used in the path, i.e.:

$$\delta_i(r) = \begin{cases} 1 & \text{If robot } r \text{ is used on the path} \\ 0 & \text{Otherwise} \end{cases}$$

Thus, by using the third label δ_i, we can ensure that each robot is used at most once. The propagation and domination rules must be slightly adapted:

- The label $(a_t, b_t, \delta_t) \in L_t$ is propagated through the arc $(t,i)_r$ if $\delta_t(r) = 0$ by adding to L_i the label $(Max[a_t, w((t,i)_r)], b_t + 1, \delta'_t)$ such that δ'_t is defined as follows:

$$\delta'_t(r') = \begin{cases} \delta_t(r') & \text{If } r' \neq r \\ 1 & \text{If } r' = r \end{cases}$$

- We say that the label (a,b,δ) dominates the label (a',b',δ') if:

$$(a \leq a' \text{ and } b \leq b' \text{ and } \delta \leq \delta'^{3})$$

- We suggest Algorithm 10 in order to compute a constrained min-max path.

[3]Let $\delta : R \longrightarrow \{0,1\}$ (a,b,δ), $\delta' : R \longrightarrow \{0,1\}$ (a,b,δ) two applications. The notation $\delta \leq \delta'$ means that $\forall r \in R, \delta(r) \leq \delta'(r)$ and the notation $\delta < \delta'$ means that $\delta \leq \delta'$ and that $\exists r \in R, \delta(r) < \delta'(r)$.

Algorithm 10 split for SDRALBP-2

INPUT ($\mathcal{I}$, σ) where $\mathcal{I}$ is an instance of the SDRALBP-2 problem and σ is a giant sequence respecting precedence constraints. We suppose without loss of generality that $\sigma = \{1, 2, ..., n\}$
OUTPUT X: optimal solution with respecting σ

Build the graph $\mathcal{H}_{\mathcal{I}}(\sigma)$
$L_0 := \{(0, 0)\}$
for t = 1 **to** n **do**
  $L_t := \emptyset$
end for
for t = 0 **to** n-1 **do**
  for all $(t, i)_r \in A$*(Propagate labels from* L_t*)* **do**
    for all $(a_t, b_t, \delta_t) \in L_t$ **do**
      if ($b_t < s_{max} - 1$ or $i = n$) and $\delta_t(r) = 0$ **then**
        $\delta'_t := \delta_t$
        $\delta'_t(r) := 1$
        $(a_i, b_i, \delta_i) := (Max[a_t, w((t, i)_r)], b_t + 1, \delta'_t)$
        if (a_i, b_i, δ_i) is not dominated by an element of L_i **then**
          $L_i := L_i \cup \{(a_i, b_i, \delta_i)\}$
          if (a_i, b_i, δ_i) dominates some element $(a'_i, b'_i, \delta'_i) \in L_i$ **then**
            $L_i := L_i \backslash \{(a'_i, b'_i, \delta'_i)\}$
          end if
        end if
      end if
    end for
  end for
end for
Decode the label within L_n to build X.

The algorithm starts with the label $(0, 0, \mathbf{0})$ at the node 0 such that $\mathbf{0}$ is the application that equals 0 everywhere on R. The algorithm then propagates labels node by node and arc by arc using dominance tests.

Lemma 3.4.2. *The number of labels on each node is limited by $n.2^{n_r}$ where n_r is the number of robots.*

Proof. There are at most $s_{max} \leq n$ possibilities for the second component of the label and at most 2^{n_r} possibilities for the third component of the label. So using the dominance rule, we have $|L_i| \leq n.2^{n_r}$ for each node i. □

Theorem 3.4.3. *Algorithm 10 runs in $O(n^3.n_r.2^{n_r})$.*

Proof. Algorithm 10 performs dominance tests for all the couples (label, arc). The number of labels being bounded by $n.2^{n_r}$ (following the previous lemma) and the number of arcs being less than $n^2.n_r$, the complexity of the algorithm is within $O(n^3.n_r.2^{n_r})$. □

Speeding up the split algorithm by means of heuristics and bounds

When dealing with the assumption "only one robot per type", split gets exponential. In this condition, split cannot be integrated into metaheuristics that require the execution of split several times. A solution to this issue is to speed up split.

We aim in this section at introducing a technique in order to speed up split. We suppose in the remainder of this section that a maximum number of workstations s_{max} is given and that the cycle time C is to be minimized. Its principle is as follows:

- Compute an upper bound C_{ub} of the cycle time for the problem thanks to an heuristic yielding a solution respecting the given giant sequence σ.
- Use the previously obtained upper bound twice for:
  - Eliminating arcs whose weights exceed the upper bound at the time of the construction of the graph, i.e., all arcs $(i, j)_r$ such that $w((i, j)_r)(= \sum_{k=i+1}^{j} d_k^r + \sum_{k=i+1}^{j-1} t_{k,k+1}^r + t_{j,i+1}^r) > C_{ub}$.
  - During the execution of the algorithm, a label (a, b, δ) is pruned if $a > C_{ub}$.

We note that the speeding up technique preserves the optimality of split for the following two reasons:

- All the arcs deleted during construction of the graph cannot be part any optimal solution.
- All the labels deleted during the execution of split cannot yield any optimal solution.

We introduce next an heuristic in order to compute C_{max}. The described algorithm could be seen as an heuristic to obtain a solution respecting a giant sequence. The heuristic is made up of several steps described below:

(1) Compute an optimal balancing solution S respecting σ not necessarily respecting the assumption "one robot per type". It can be obtained by applying the split algorithm from the previous subsection.

(2) If there are robots being assigned more than once, transform the solution S to a solution respecting the assumption "one robot per type":

   (a) $S_v = \emptyset$

   (b) For each robot being assigned more than once, add all the workstations where this robot is assigned to S_v expect the workstation with the maximum workload.

   (c) Build a permutation $\tilde{S}_v$ of all the workstations in S_v.

   (d) For each workstation $s \in S_v$ following the order given by $\tilde{S}_v$: assign to s a robot that is not already assigned and that minimizes the workload of the workstation.

   (e) Repeat step (d) with a permutation $\tilde{S}_{v'}$ obtained through an insertion move while it leads to an improving solution.

3.5 Conclusion

We suggested in this chapter a polynomial algorithm to solve the particular case where the giant sequence is given. The split algorithm is suggested to solve this polynomial case. split is applied for SALBP-2, RALB-1/SDRALBP-1, RALB-2/SDRALBP-2 and RTLBP. Table 3.5 summarizes the underlying graph problems and time complexities. split was inspired to us by the homonym approach from Vehicle Routing Problems. This establishes a link between two different problems but which are of the same nature since both routing problems and balancing problems can be considered as "packing problems". This rapprochement is yet to investigate in order to make the balancing problems benefit from the cumulative experience on VRP.

Problem	Underlying graph problem	Time complexity
SALBP-2	Constrained min-max path	$O(n^4)$
RTLBP	Constrained shortest path	$O(n^4)$
RALBP-1/SDRALBP-1	Shortest path	$O(n^2)$
RALBP-2/SDRALBP-2	Constrained min-max path	$O(n^4)$

Table 3.5: Underlying graph problems and complexity

split has many advantages among which its low complexity and its simplicity of implementation. We adapted it for the assumption "one robot per type". Under this particular assumption, the polynomiality of split is lost. Optimal algorithms are obtained in $O(n^2.n_r.2^{n_r})$ and $O(n^3.n_r.2^{n_r})$ respectively for RALB-1/SDRALBP-1 and RALBP-2/SDRALBP-2. The latter complexities are still reasonable if n_r is not too big. However, in order to tackle big-size instances we suggest a speeding technique based on heuristics and bounds. The latter preserves the optimality of split.

The split algorithm integrates very well in methods of type Sequence-First Balance-Last or Sequence-First Balance-And-Select-Last intended respectively for reconfigurable transfer lines and robotic assembly lines. Indeed, the second step can be solved optimally thanks to split in both approaches. Therefore, all that remains to design such approaches is the sequencing part. Two different schemes are suggested in the next chapter for the sequencing part:

- **Approximate:** a giant sequence of good-quality is obtained thanks to a metaheuristic that explores the space of giant sequences and uses split as a decoding procedure.
- **Exact:** the best giant sequence, and therefore the best solution, is obtained thanks to a branch and bound algorithm. The latter explores a tree representing all the giant sequences. Split is used to compute a lower bound on the nodes (subsequences).

Chapter 4

Sequence-First Balance-Last approaches using split approach

Contents

4.1 Introduction **89**
4.2 A split-based metaheuristic **90**
4.2.1 General scheme 90
4.2.1.1 Encoding-decoding scheme 92
4.2.1.2 Metaheuristic scheme 93
4.2.2 Local search and perturbation moves 95
4.2.3 Computing a compatible giant sequence 95
4.2.3.1 RALBP-2 and SDRALBP-2 96
4.2.3.2 RTLBP 96
4.3 A split-based branch and bound algorithm **101**
4.3.1 Branching strategy 102
4.3.2 Exploration strategy 102
4.3.3 Split-based lower bounding technique 104
4.4 Conclusion **106**

4.1 Introduction

We introduce in this chapter split-based resolution methods for the RTLBP, the RALBP-2 and the SDRALBP-2. This approach can be seen as the reverse approach to the Balance-First Sequence-Last approach introduced in chapter 2. The objective

is first to build a sequence and secondly to determine an optimal balancing for this sequence. A split-based metaheuristic and a split-based branch and bound are suggested. For both methods, split is used as a building block.

In regards to the split-based metaheuristic, a giant sequence is used to encode a solution and split is used as a decoding procedure. The space of giant sequences is explored using a metaheuristic. This split-based encoding-decoding scheme significantly reduces the search space. Split acts as an aggressive and efficient guidance technique which allows, for a given sequence, to solve in polynomial time all the other decision sub-problems. Indeed, approximate resolution methods from literature use either an heuristic or a exponential algorithm to decode a giant sequence (Borisovsky, Delorme, and Dolgui (2013), Nilakantan et al. (2015)) or use a complete encoding of the solution including robot assignment vector and/or operations to workstations assignment vector (Janardhanan et al. (2019)) which yields a larger search space. The general scheme of the split-based metaheuristic is independent from the considered problem, only the algorithm to compute a compatible giant is problem-dependent.

In regards to the split-based branch and bound resolution methods, split is used as a procedure to compute a lower bound in a branch and bound algorithm. It is still in development phase, partial results are given in the experimental section.

The two next subsections are devoted respectively to the split-based mataheuristic and the split-based branch and bound.

4.2 A split-based metaheuristic

This section is devoted to the split-based metaheuristic. We describe in the following subsections the general scheme of the method, the local search and perturbation moves and finally the algorithm to compute a compatible giant sequence.

4.2.1 General scheme

The general scheme of the split-based metaheuristic is depicted in Figure 4.1.

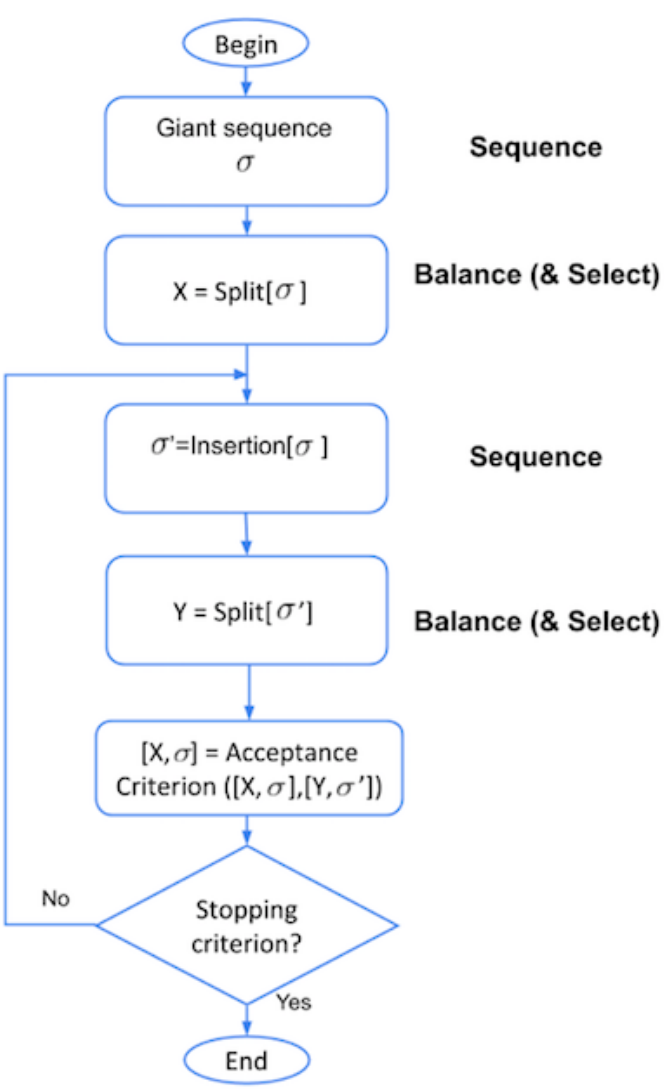

Figure 4.1: Scheme of the split-based metaheuristic

An initial giant sequence σ is computed first, the corresponding (optimal) solution is given by split $X = split[\sigma]$. A neighboring giant sequence σ' of σ is computed thanks to an insertion move (described later in the section). The corresponding (optimal) solution is given by split $Y = split[\sigma']$. An acceptance criterion, corresponding to the solution with the lowest objective value, is considered to choose which giant sequence to take for the next iteration.

We describe next the encoding-decoding scheme and some of its properties.

4.2.1.1 Encoding-decoding scheme

Sequence-first methods work within two steps. The first step aims to solve the sequencing problem which can be done by giving a sequence of all operations (giant sequence). The second step aims to solve the balancing problem and the balancing and the selection problem (if relevant) while respecting the giant sequence. For this reason, we call sequence-first methods for the RTLBP: *Sequence-First Balance-Last (SFBL)* methods, that of RALBP *Sequence-First Select-Last (SFSL)* methods and that of SDRALBP: *Sequence-First Balance-And-Select-Last (SFBSL)* methods. In those methods, split is used as a black-box algorithm (Figure 4.2) to evaluate a giant sequence.

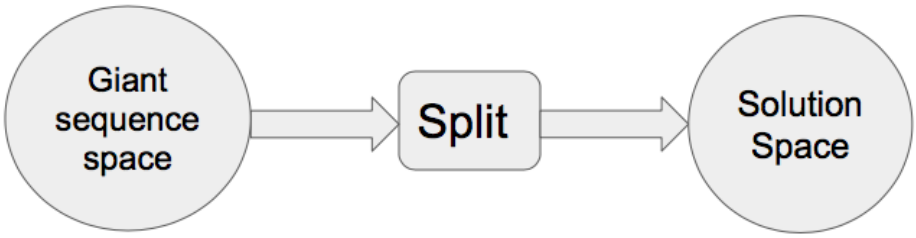

Figure 4.2: Split is a decoding procedure, mapping each sequence into a solution of the problem.

Many metaheuristics resolution approaches in literature use a giant sequence to encode a solution. Only heuristics or exponential algorithms have been used so far to decode a giant sequence (Borisovsky, Delorme, and Dolgui (2013), Nilakantan et al. (2015)). In other publications, additional sequences are used together with the giant sequence to encode a solution (complete encoding) which significantly enlarges the search space (Janardhanan et al. (2019)). To design a split-based metaheuristic, we use a giant sequence to encode a solution and split to decode it. We denote this encoding-decoding scheme by $[\sigma, split]$. This way, the search space is significantly reduced while preserving the optimal solution:

Theorem 4.2.1. *The encoding-decoding technique:* $[\sigma, split]$ *preserves an optimal solution.*

Proof. Any optimal solution X can be represented by a giant sequence σ_X. The execution of split on σ_X yields either X or any other solution with equivalent objective value since it has been shown in the previous chapter that split returns an optimal solution corresponding to a giant sequence. □

The previous theorem shows that the set of optimal solutions has non-empty intersection with the set of solutions given by split (Figure 4.3).

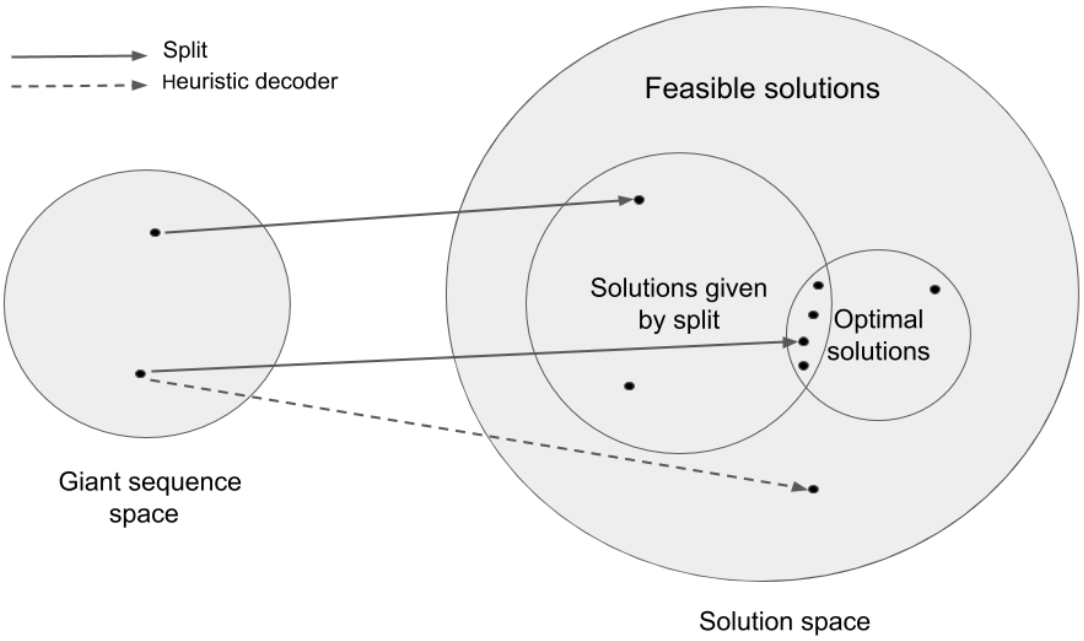

Figure 4.3: Split versus heuristic decoder

In Figure 4.3, the space of giant sequences is voluntarily represented smaller than the space of solutions. Split gives an optimal solution when the giant sequence encode an optimal solution. An heuristic decoder does not guarantee the same. All what remains is searching for a "good" giant sequence by means of metaheuristics. The following subsection explains the metaheuristic scheme chosen.

4.2.1.2 Metaheuristic scheme

Unlike local search that is trapped in a local optimum, iterated local search (ILS) introduces diversification by applying a perturbation to the local optimum and iterating the local search. It exploits the idea that performing local search from a pretty good solution (perturbed local optimum) is more effective than starting from a random solution. ILS has proven to be efficient for various combinatorial optimization problems: Lourenço, Martin, and Stützle (2010), Stützle (2006), Dong, Huang, and Chen (2009).

Once having computed a starting giant sequence, we perform an iterated local search in the space of the giant sequences, the split is used to evaluate each giant sequence. The general scheme is depicted in Algorithm 11. In this algorithm, $H(X)$ denotes the cost of solution X, i.e., the number of machines of RTLBP or the cycle time for RALBP-2 and SDRALBP-2.

Algorithm 11 split-based ILS algorithm

INPUT: An instance of SDRALBP or RTLBP.
OUTPUT: X^*, best found solution.

1: Compute a compatible giant sequence: σ
2: Perform split: $X = split(\sigma)$
3: Record the best known sequence: $(X^*, \sigma^*) = (X, \sigma)$
4: **while** Stopping criterion ILS is not met **do**
5: **while** Stopping criterion LS is not met **do**
6: Choose a random neighbor of σ: σ'
7: Perform split: $X' = split(\sigma')$
8: **if** $H(X') \leq H(X)$ **then**
9: $(X, \sigma) = (X', \sigma')$
10: **end if**
11: **end while**
12: **if** $H(X) \leq H(X^*)$ **then**
13: $(X^*, \sigma^*) = (X, \sigma)$
14: **end if**
15: $(X, \sigma) = Perturbation[(X^*, \sigma^*)]$
16: **end while**

We start with computing a compatible giant sequence σ (giant sequence corresponding to a feasible solution). Then a local search is performed from σ (lines 5-11): a random neighbor σ' is selected and evaluated thanks to split. σ is updated if $H(X') \leq H(X)$ where X and X' denote the solutions corresponding respectively to σ and σ' (lines 8-10). The local search terminates when the stopping criterion of the local search is met. The best known solution X^* is updated after each local search (lines 12-14), this acceptance criterion is called Better Walk in Lourenço, Martin, and Stützle (2010). A perturbation is applied from the giant sequence yielding the best known solution σ^* (line 15). The procedure (local search + perturbation) is iterated while the stopping criterion of the ILS is not met. We consider a maximum number of iterations as stopping criterion for the iterated local search and the local search. The giant sequence yielding the overall best solution across the ILS is returned by the algorithm.

Each ingredient of the ILS is detailed next: the local search move, the perturbation operator and the method to build a compatible giant sequence are described next.

4.2.2 Local search and perturbation moves

Neighborhood move

We use a simple insertion neighborhood that inserts an operation in a different position of the giant sequence. This neighborhood is applied in such way that the precedence constraints are respected. Given the giant sequence $\sigma = (\sigma_1, \ldots, \sigma_n)$, a random neighbor is obtained by selecting a random operation $\sigma_i, 1 \leq i \leq n$. Once this operation selected, two operations must be identified $\sigma_{l(i)}$ and $\sigma_{f(i)}$ such that

$l(i) = max\{j; j < i \text{ and } (\sigma_j, \sigma_i) \in P\}$, the position of the last predecessor of σ_i in σ

and

$f(i) = min\{j; i < j \text{ and } (\sigma_i, \sigma_j) \in P\}$, the position of the first successor of σ_i in σ

Then a random position is selected between positions $l(i)$ and $f(i)$ (uniform selection in $\{l(i)+1, \ldots, f(i)-1\}$) to (re)insert operation σ_i.

Remark 6. *During the local search process, a new auxiliary graph must be built for each new neighbor. To accelerate the process, only parts of the graph that are subject to change are reprocessed. Since we use an insertion move, when an operation at position i is moved to some position j:*

- *If $j > i$: Arcs (σ_a, σ_b) such that $(b \leq i-1)$ or $(a \geq j)$ or $(a \geq i$ and $b \leq j-1)$ remain unchanged.*
- *If $i > j$: Arcs (σ_a, σ_b) such that $(b \leq j-1)$ or $(a \geq i)$ or $(a \geq j$ and $b \leq i-1)$ remain unchanged.*

Perturbation move

The perturbation operator in the iterated local search stands for applying the neighborhood operator 3 times. Each of the three neighborhood moves is repeated until a compatible giant sequence is found. In other words, after the application of each operator, we test the compatibility of the resulting giant sequence by applying split: if it is not compatible, we reject it and apply the operator again.

4.2.3 Computing a compatible giant sequence

We recall that a compatible giant sequence is a giant sequence for which there exist a feasible solution (split returns a solution in this case). A compatible giant sequence necessarily respects precedence constraints.

The procedure for computing a compatible giant sequence is problem-dependent. The following subsection is devoted to the case of RALBP-2 and SDRALBP-2, then the case of RTLBP is detailed separately.

4.2.3.1 RALBP-2 and SDRALBP-2

Theorem 4.2.2. *A compatible giant sequence with respect to an instance of RALBP-2 and SDRALBP-2 is a giant sequence respecting precedence constraints.*

Proof. Given a giant sequence respecting precedence constraints, a feasible solution satisfying this giant sequence can be obtained by allocating all the giant sequence to a single workstation. □

A compatible giant sequence respecting precedence constraints can be obtained thanks to Algorithm 12. An alternative approach to Algorithm 12 would be to organize the operations within successive levels. The operations within each level do not have any predecessors in the next levels.

Algorithm 12 Algorithm to build a giant sequence respecting precedence constraints

INPUT: An instance of SDRALBP-2 or RALBP-2.
OUTPUT: A giant sequence σ respecting precedence constraints.

1: $\sigma = \emptyset$
2: **while** $|\sigma| < n$ **do**
3: Select randomly and uniformly i an operation without predecessor or such that all predecessors have already been included in σ.
4: Append i to the end of σ: $\sigma := (\sigma; i)$
5: **end while**

4.2.3.2 RTLBP

If no inclusion, exclusion, accessibility, M_{max}, N_{max} or s_{max} constraints are taken into consideration, then it is easy to compute a compatible giant sequence. Indeed, any giant sequence respecting the precedence constraints is compatible in this case. Yet, if we consider some combination of those constraints, any sequence respecting the precedence constraints is no longer guarantied to be compatible.

Theorem 4.2.3. *Given an instance $\mathcal{I}$ of the RTLBP, the problem of finding a compatible giant sequence is NP-Hard.*

Proof. We show that the TSP (Traveling Salesman Decision Problem) is reduced to the considered problem. Indeed, the TSP is the task of deciding whether there exists a "tour" visiting a given set of n_c cities and returning to the initial city such that the total distance traveled is less or equal to a given constant B. Then, an instance of the TSP could be mapped with an instance of the RTLBP where operations represent cities, the setup times between operations represent distances between cities and such that $I = N, E = \emptyset, Pos = \{0\}, Pos_i = \{0\}$ $(\forall i \in N), s_{max} = 1, M_{max} = 1, N_{max} = n_c, d_i = 0$ $(\forall i \in N)$ and $C = B$. This is a polynomial time reduction which justifies the NP-Hardness of our problem.

□

Corollary 4.2.4. *The problem of finding a feasible solution for the RTLBP is NP-Hard.*

Definition 4.2.5. *(Weakly compatible giant sequence) We define a weakly compatible giant sequence as a giant sequence for which there exists a balancing solution that is feasible with relaxing the maximum number of workstations constraint (s_{max}).*

Theorem 4.2.6. *Given an instance $\mathcal{I}$ of the RTLBP, the problem of finding a weakly compatible giant sequence is NP-Hard.*

Proof. The same reduction as Theorem 4.2.3 can be performed without consideration of s_{max}.

□

In order to compute a compatible giant sequence, we proceed in two steps:

- Computing a weakly compatible giant sequence.
- Repairing the giant sequence to make it compatible.

Computing a weakly compatible giant sequence

Even the problem of determining a weakly compatible giant sequence being NP-Hard, we are content to propose an exponential algorithm (but quite fast in practice) to obtain such a sequence (Algorithm 13). Then, a repair procedure is suggested to obtain a compatible giant sequence from a weakly compatible giant sequence (Algorithm 15).

Algorithm 13 works in three steps:

Step 1 Gather the operations according to inclusion constraints: we check easily that the inclusion constraints define an equivalence relation (denoted by Inc). Thus, we can compute the quotient set:

$$N/Inc = \{S_1, S_2, \ldots, S_k\}$$

In other words, S_i are groups of operations, built such that if there exists an inclusion relation between two tasks, then they are in the same S_i.

Step 2 Sequence operations on each set: optimally sequence the operations in the subsets $S_i, 1 \leq i \leq k$ while respecting precedence constraints: we solve an Asymmetric Traveling Salesman problem (ATSP) where operations represent cities and setup times represent distances between cities. This is performed thanks to a dynamic program adapted from Held and Karp (1962) (Algorithm 1 from the previous chapter). In this way, the subsets S_i become subsequences denoted by $\tilde{S}_i$.

Step 3 Build the weakly compatible giant sequence respecting the precedence constraints by concatenation of the subsequences.

Remark 7. *Since Algorithm 13 uses the dynamic programming algorithm from Held and Karp (1962) in step 2, it is exponential in the size of the subsets $S_i, 1 \leq i \leq k$. However, the size of every subset is bounded by N_{max} which is usually much smaller than the number of operations. In practice, Algorithm 13 is quite efficient both in computation time and memory usage.*

Algorithm 13 Pseudo-algorithm to compute a weakly compatible giant sequence

INPUT: An instance of RTLBP.
OUTPUT: σ giant sequence that is hopefully weakly compatible.

Step 1 Gather in subsets the operations according to inclusion constraints: $\{S_1, S_2, \ldots, S_k\}$

Step 2 Sequence the operations in each each subset S_i (we use Algorithm 1 from the previous chapter: Held and Karp (1962)).

Step 3 Build a weakly compatible giant sequence σ respecting the precedence constraints:

(a) $\sigma = \emptyset$.

(b) **While** $(|\sigma| < n)$

i. Select randomly a subsequence $\tilde{S}_i$ such that all predecessors of operations in $\tilde{S}_i$ are already contained in σ.

ii. Append $\tilde{S}_i$ to σ.

(c) **End while**

Definition 4.2.7. *(Triangular inequality) We say that the setup times respect the triangular inequality if:*

$$t_{i,j} + t_{j,k} \leq t_{i,k}, \forall (i,j,k) \in N \times N \times N$$

Theorem 4.2.8. *Given a feasible instance $\mathcal{I}$ of the RTLBP where the setup times respect the triangular inequality, Algorithm 13 returns a weakly compatible giant sequence.*

Proof. Let us consider the solution where each subsequence $\tilde{S}_i, 1 \leq i \leq k$ is allocated to a different workstation. This solution respects the giant sequence outputted by Algorithm 13. Besides, it is easy to see that this solution is feasible with respect to inclusion, exclusion, accessibility, precedence and N_{max} constraints. It remains to verify that the M_{max} constraint is satisfied. To do this, we must check that for each $\tilde{S}_i$, $W^1{}_{\tilde{S}_i} \leq C.M_{max}$. Since all the operations of each subsequence $\tilde{S}_i$ are linked with inclusion constraints, they should all be included in the same workstation in any feasible solution. In other words, for each subsequence $\tilde{S}_i$, there must exist a subsequence $\tilde{S}'_i$ such that:

(1) $\tilde{S}_i \subset \tilde{S}'_i$.

(2) $W_{\tilde{S}'_i} \leq C.M_{max}$.

Since the triangular inequality is satisfied, (1) implies that:

$$W_{\tilde{S}_i} \leq W_{\tilde{S}'_i}$$

Therefore, (2) implies that $W_{\tilde{S}_i} \leq C.M_{max}$. □

If the setup times do not respect the triangular inequality, Algorithm 13 remains a good heuristic for computing a weakly compatible giant sequence.

Remark 8. *The solution described in the proof of Theorem 4.2.8 can be obtained by a modified version of split, we denote it by $\widetilde{split}$. $\widetilde{Split}$ is described in Algorithm 14, it allows solutions exceeding s_{max} workstations. If feasible solutions with at most s_{max} workstations exist, $\widetilde{split}$ has the same behavior as split. Otherwise, it minimizes the number of workstations.*

[1] $W_{\tilde{S}}$ where $\tilde{S}$ is a subsequence of operations defines the sum of the processing times and the setup times induced by the subsequence. For example, if $\tilde{S} = (1,2,3)$, $W_{\tilde{S}} = d_1 + t_{1,2} + d_2 + t_{2,3} + d_3 + t_{3,1}$.

Algorithm 14 $\widetilde{Split}$ algorithm for the RTLBP

INPUT ($\mathcal{I}$,σ) where $\mathcal{I}$ is an instance of the RTLBP and σ is a giant sequence respecting precedence constraints.
OUTPUT X: a feasible solution respecting σ that minimizes the number of machines if such a solution exists, a solution violating the maximum number of workstations constraint that minimizes the number of workstations otherwise.

Build the graph $\mathcal{H}_{\mathcal{I}}(\sigma)$
$L_0 := \{(0,0)\}$
for t=1 **to** n **do**
  $L_t := \emptyset$
end for
for t=0 **to** n-1 **do**
  for all $(t,i) \in A$ *(Propagate labels from L_t)* **do**
    for all $(a_t, b_t) \in L_t$ **do**
      $(a_i, b_i) := (a_t + c_{t,i}, b_t + 1)$
      if (a_i, b_i) is not dominated by an element of L_i **then**
        $L_i := L_i \cup \{(a_i, b_i)\}$
        if (a_i, b_i) dominates some elements $(a'_i, b'_i) \in L_i$ **then**
          $L_i := L_i \backslash \{(a'_i, b'_i)\}$
        end if
      end if
    end for
  end for
end for
if $L_n \neq \emptyset$ **then**
  if $Min_{(a_i,b_i)\in L_n}(b_i) \leq s_{max}$ **then**
    $C^* := Min_{(a_i,b_i)\in L_n; b_i \leq s_{max}}(a_i)$
  else
    $C^* := a_i^*$ such that $(a_i^*, b_i^*) = Argmin_{(a_i,b_i)\in L_n}(b_i)$
  end if
  Decode the path of length C^* to build X
end if

Computing a compatible giant sequence from a weakly compatible giant sequence

After obtaining a weakly compatible giant sequence with Algorithm 13, an heuristic (Algorithm 15) is designed to obtain a compatible giant sequence. The heuristic works as follows: starting from a weakly compatible giant sequence, a local search

with insertion neighborhood (introduced in the previous chapter) is applied to obtain a compatible giant sequence. A neighbor is accepted if its split uses less workstations. In Algorithm 15, $WS(X)$ denotes the number of workstations in solution X.

Algorithm 15 Algorithm to compute a compatible giant sequence from a weakly compatible giant sequence

INPUT: σ, a weakly compatible giant sequence and $\mathcal{I}$ an instance of the RTLBP.
OUTPUT: σ, a (hopefully) compatible giant sequence.

while (Stopping condition is not met) and (σ is not compatible) **do**
  $X = \widetilde{split}(\mathcal{I}, \sigma)$
  Choose a random neighbor of σ: σ'
  if $\widetilde{split}(\mathcal{I}, \sigma')$ returns a solution **then**
    $X' = \widetilde{split}(\mathcal{I}, \sigma')$
    if $WS(X') \leq WS(X)$ **then**
      $[X, \sigma] = [X', \sigma']$
    end if
  end if
end while

We just described an heuristic intended to compute a compatible giant sequence which consists in building a weakly compatible sequence and then repairing it (Algorithm 13 + Algorithm 15). By coupling it with split, the method builds a feasible solution. It will be called "H-SFBL" in the remainder of the manuscript.

4.3 A split-based branch and bound algorithm

A brand and bound algorithm searches the solutions space using a tree structure. Each node of the tree represents a set of partial solutions. All the nodes should be explored in order to compute an optimal solution. Thankfully, pruning schemes detect unpromising nodes and limit the search process. When an unpromising node is detected, all the descendant nodes are pruned. Pruning schemes relies on an upper bound computed at the beginning of the algorithm thanks to some heuristic. Pruning schemes also relies on an efficient lower bound computed at each node in order to evaluate the children solutions of the current node. The branching technique describes the construction and the search of the graph.

In order to define a branch and bound algorithm, the three following steps must be described:

- Branching strategy: it defines the nodes, their signification and the structure of the search tree. It defines in particular the paternity relationship between the nodes (branching).
- Exploration (or search) strategy: it defines the priority according to which the nodes are visited.
- Bounding strategy: it defines the computation of the lower bound in each node.

A complete taxonomy of branch and bound algorithms can be found in Morrison et al. (2016).

4.3.1 Branching strategy

The suggested algorithm exploits the idea that the problem becomes polynomial once having fixed a giant sequence. It enumerates all possible giant sequences using a tree structure.

Each node in the tree structure represents a subsequence of operations. From a root node representing an empty subsequence, we perform branching to determine an operation to add to the subsequence. Then in each node, we branch on all nodes that have not yet been visited in the the path between the root and the node.

The branch and bound algorithm is more likely to be efficient when it starts with the most promising nodes. We use the previously described split-based approximate method as an upper bound. It is computed once at the beginning of the algorithm. We use the giant sequence relative to the initial upper bound to guide the search in the tree. This giant sequence is denoted by $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n)$. The branching is done with respect to the sequence σ (Figure 4.2).

Each leaf represents a sequence of all operations (giant sequence). The tree contains $1 + n + n(n-1) + ... + n!$ nodes and $n!$ leafs. Its height is of n.

4.3.2 Exploration strategy

Different search strategies exist in the branch-and-bound literature, they are described in Morrison et al. (2016). In this preliminary study, we use a depth-first search.

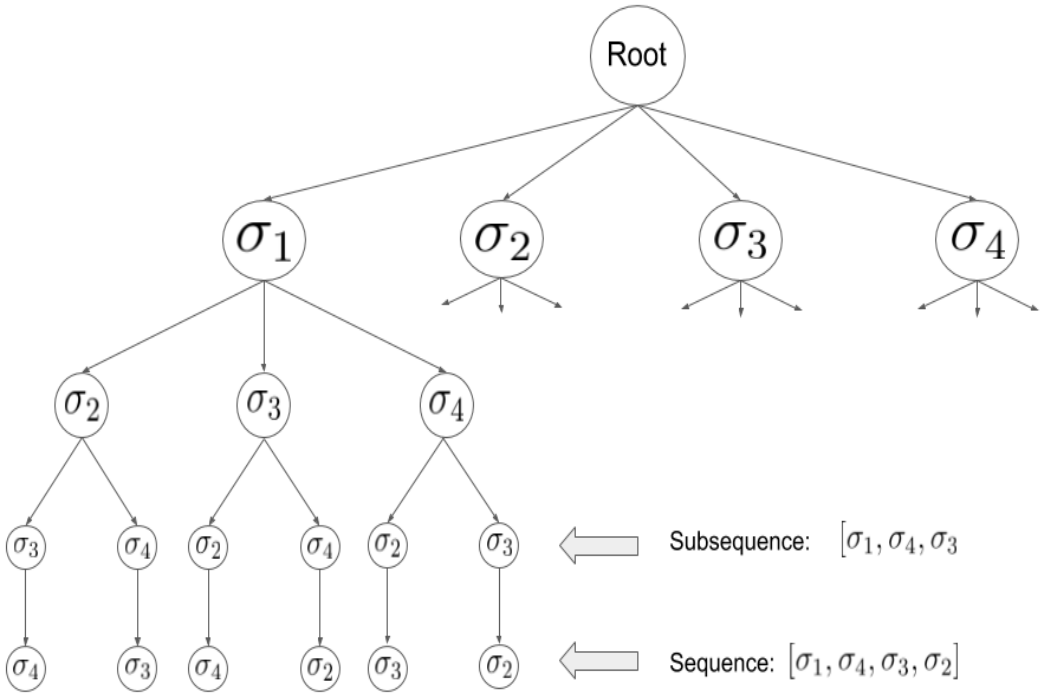

Figure 4.4: Tree structure for an instance of 4 operations. The subsequence represented by a node and the sequence represented by a leaf are explicited.

There are 3 scenarios where a node is pruned:

(1) The actual node is a leaf: in this case the node is representative of a giant sequence and split is applied. The upper bound is updated if split returns a better solution.

(2) The lower bound in the actual node is greater than the upper bound: in this case, this node (and all its descendants) is pruned. We cannot improve the upper bound solution from this node.

(3) The actual node is unfeasible: the subsequence cannot be respected, it can be detected thanks to the split-based lower bound procedure (Theorem 4.3.2)

4.3.3 Split-based lower bounding technique

Each node is representative of a subsequence of operations. If the node is of depth h, the subsequence contains h operations.

Let's consider a node of depth h. We denote the subsequence of operations at this node by :

$$\delta = (\delta_1, \delta_2, ..., \delta_h)$$

In order to compute a lower bound, we complete this subsequence with fictious operations $\gamma_{h+1}, ..., \gamma_n$ that are not included in the operations set. As a result of this, we get the giant sequence:

$$\delta^h = (\delta_1, \delta_2, ..., \delta_h, \gamma_{h+1}, ..., \gamma_n)$$

Then, the lower bound is computed by performing the following steps :

(1) Build up the graph $\widetilde{\mathcal{H}}_{\mathcal{I}}(\delta^h)$ as follows:

$\widetilde{\mathcal{H}}_{\mathcal{I}}(\delta^h)$ has the same set of vertices and the same set of arcs than $\mathcal{H}_{\mathcal{I}}(\delta^h)$, the arc weights are however modified as follows:

$$\tilde{c}_{i,j} = \begin{cases} c_{i,j} & \text{If } j \leq h \\ \left\lceil \frac{\sum_{k=i+1}^{h} d_k + \sum_{k=i+1}^{h-1} t_{k,k+1} + min\ t_{h,\cdot} + min\ t_{\cdot,i+1} + dt_{min}(j-h)}{C} \right\rceil & \text{If } j > h \text{ and } j > i+1 \\ \left\lceil \frac{min_{\delta}\ d_{\cdot}}{C} \right\rceil & \text{If } j > h \text{ and } j = i+1 \end{cases}$$

such that:

- $min\ t_{h,\cdot} = min_{k \in \mathcal{N} - \{h\}} t_{h,k}$

- $min\ t_{.,i+1} = min_{k \in \mathcal{N} - \{i+1\}} t_{k,i+1}$
- $dt_{min}(j-h) = min_{\delta}^{j-h}\ d_{.} + min_{\delta}^{j-h-1}\ t_{.,.}$ such that:
  - $min_{\delta}\ d_{.} = min_{k \in \mathcal{N} - \{\delta_1, \delta_2, ..., \delta_h\}} d_k$
  - $min_{\delta}^{j-h}\ d_{.}$ is the sum of the $j-h$ smallest completion times among $\mathcal{N} - \{\delta_1, \delta_2, ..., \delta_h\}$.
  - $min_{\delta}^{j-h-1}\ t_{.,.}$ is the sum of the $j-h-1$ smallest setup times among $\{t_{a,b}; a, b \in \mathcal{N} - \{\delta_1, \delta_2, ..., \delta_h\}\}$.

(2) Run the split algorithm in $\widetilde{\mathcal{H}}_{\mathcal{I}}(\delta^h)$

Inclusion, exclusion and accessibility constraints are considered between operations within $\{\delta_1, \delta_2, ..., \delta_h\}$. No inclusion, exclusion or accessibility constraints involving operations within $\{\gamma_{h+1}, ..., \gamma_n\}$ are considered

Theorem 4.3.1. *Let s be the solution cost obtained when applying split in*

$$\widetilde{\mathcal{H}}_{\mathcal{I}}(\delta_1, \delta_2, ..., \delta_h, \gamma_{h+1}, ..., \gamma_n)$$

then s is a lower bound for all solutions obtained from node n.

Proof. Let f be a leaf node obtained from the node n. Let ω be the giant sequence that f is representative of. Necessarily, ω starts with $\delta_1, \delta_2, ..., \delta_h$.

We put:

$$\omega = \delta_1, \delta_2, ..., \delta_h, \delta_{h+1}, ..., \delta_n$$

The graphs $\mathcal{H} = \widetilde{\mathcal{H}}_{\mathcal{I}}(\delta_1, \delta_2, ..., \delta_h, \gamma_{h+1}, ..., \gamma_n)$ and $\mathcal{H}' = \mathcal{H}_{\mathcal{I}}(\omega)$ are isomorphic. Besides, since edge weight is a non-decreasing function with respect to processing times and setup times, for any edge (i, j) in $\mathcal{H}$ and its corresponding edge (i', j') in $\mathcal{H}'$, we necessarily have:

$$\tilde{c}_{i,j} \leq c_{i',j'}$$

Consequently any path in $\mathcal{H}$ has a smaller cost than its corresponding path in $\mathcal{H}'$ and the application of split in $\mathcal{H}$ gives a lower bound.

□

Theorem 4.3.2. *Let s be the solution cost obtained when applying split in*

$$\mathcal{H}_{\mathcal{I}}(\delta_1, \delta_2, ..., \delta_h, \gamma_{h+1}, ..., \gamma_n)$$

then: $s = +\infty$ if and only if the subsequence $\delta_1, \delta_2, ..., \delta_h$ is incompatible with respect to $\mathcal{I}$

Proof. It is already known from the split algorithm that : $s = +\infty$ if and only if the subsequence $\delta_1, \delta_2, ..., \delta_h, \gamma_{h+1}, ..., \gamma_n$ is incompatible with respect to $\mathcal{I}$. Since operations $\gamma_{h+1}, ..., \gamma_n$ does not intervene in any inclusion, exclusion, accessibility or precedence constraints; we necessarily have $(\delta_1, \delta_2, ..., \delta_h, \gamma_{h+1}, ..., \gamma_n)$ incompatible with respect to $\mathcal{I}$ if and only if $(\delta_1, \delta_2, ..., \delta_h)$ is incompatible with respect to $\mathcal{I}$.

□

Remark 9. *Split is applied in each node of the tree to obtain a lower bound. A new auxiliary graph must be built in each node. To accelerate the process, only parts of the graph that are subject to change are reprocessed.*

4.4 Conclusion

We have described in this chapter, two methods based on split to solve the RTLBP, the RALBP-2 and the SDRALBP-2:

- An split-based mataheuristic: a giant sequence is used to decode a solution while split is used as decoding procedure in a metaheuristic. We proved that this encoding-decoding technique significantly reduces the search space and preserves an optimal solution. This method will be investigated in the experimentation section for the RTLBP, the RALBP2 and for the SDRALBP-2. It has been published in (Lahrichi et al. (2020a)) and (Lahrichi et al. (2020c)).
- An split-based branch and bound: split is used to compute a lower bound in a branch and bound algorithm. It is still a preliminary work. First results for the RTLBP are given next.

Next chapter is devoted to computer experiments. The suggest methods in this chapter as well as BFSL (chapter 2) and the ILPs (chapter 1) are tested on instances taken from literature. The focus will be made on the split-based metaheuristic. We consider it is the major contribution of this thesis since it introduces a rupture in relation to literature where only heuristics or exponential decoders are used to decode a giant sequence.

Chapter 5

Experimental results

Contents

5.1 Introduction **108**
5.2 Reconfigurable Transfer Lines **108**
5.2.1 Instances 109
5.2.1.1 Small-size instances 109
5.2.1.2 Large-size instances 109
5.2.2 Exact methods 110
5.2.3 Heuristics 112
5.2.4 Metaheuristics 115
5.2.4.1 M-BFSL 115
5.2.4.2 Split-based ILS 121
5.3 Robotic Assembly lines **127**
5.3.1 RALBP-2 127
5.3.1.1 Instances 127
5.3.1.2 Split-based ILS 127
5.3.2 Sequence-dependent Robotic Assembly Line Balancing Problem-2 130
5.3.2.1 Instances 131
5.3.2.2 Split-based ILS 131
5.4 Conclusion **133**

5.1 Introduction

We conduct in this chapter experiments in order to evaluate the methods suggested in the thesis. We note that our code is available online at the following link: `https://github.com/YoussefLahrichi`.

The chapter is organized as follows:

- We first consider reconfigurable transfer lines:
  - The literature instances are presented. There are small-size and big-size instances.
  - The ILP and the branch and bound algorithm are evaluated on small-size instances.
  - The heuristics H-BFSL and H-SFBL both intended to build a feasible solution are evaluated and compared.
  - The matheuristic M-BFSL and the split-based metaheuristic are evaluated and compared with a genetic algorithm from literature on big-size instances.
- Robotic Assembly Lines:
  - The literature instances are presented. There are instances without setup times (RALBP-2) and instances with non-null setup times (SDRALBP-2).
  - On instances without setup times (RALBP-2), the split-based metaheuristic is evaluated and compared with two resolution approaches from literature: a metaheuristic and a branch and bound algorithm.
  - For instances with non-null setup times (SDRALBP-2), the only published method in literature assumes "only one robot per type". The later gives an upper bound for the split-based metaheuristic.

The experiments were held on a computer equipped with a 16GB RAM i7-4790 CPU. We used JAVA 8 to implement the algorithms. IBM's CPLEX (v12.7.0) is used to solve the ILP of the RTLBP and the ILP from H-BFSL.

5.2 Reconfigurable Transfer Lines

We recall that the Reconfigurable Transfer Line Balancing problem is concerned with minimizing the number of machines.

In this section, we first present the instances, then the results for exact methods (ILP and branch and bound), the heuristics, the matheuristic and the split-based metaheuristic.

5.2.1 Instances

We consider two types of instances from literature: small-size instances and large-size instances.

5.2.1.1 Small-size instances

The small-size instances are taken from Essafi et al. (2010), they are organized within three classes depending on their Scholl density. The Scholl density allows to measure the density of the precedence graph: $d_s = \frac{2*100*\sum_{i \in N} |P_i|}{n*(n-1)}$, where P_i denotes the predecessors of operation i. All the instances contain 14 operations, they are organized as follows:

- (p14-10-1)-(p14-10-10) are low-density instances. Scholl density is within [5,15].
- (p14-25-1)-(p14-25-10) are medium-density instances. Scholl density is within [15,25].
- (p14-40-1)-(p14-40-10) are high-density instances. Scholl density is within [25,40].

The set of 30 instances are all feasible except p14-10-3.

5.2.1.2 Large-size instances

The large-size instances are taken from Borisovsky, Delorme, and Dolgui (2013). They have the following characteristics:

- Number of operations: $n = 200$
- Maximum number of workstations: $s_{max} = 25$
- Maximum number of operations per workstation: $N_{max} = 10$
- Maximum number of machines per workstation: $M_{max} = 5$
- Cycle time: $C = 50$
- Processing times: $d_i \in [1, 10]$
- Setup times: $t_{i,j} \in [0, 2]$

- Number of precedence constraints: $|P| \in [\![50, 70]\!]$
- Number of inclusion and exclusion sets: $|I|, |E| \in [\![7, 15]\!]$
- Number of possible part-fixing positions: $|Pos| = 7$.

The 15 instances are noted from A1 to A15.

5.2.2 Exact methods

The results of the ILP and the branch and bound are given in Table 5.1, Table 5.2 and Table 5.3. The notations used in the tables are as follows:

- d_s: Scholl density.
- B&B: optimal objective value, i.e. the number of machines. It is obtained thanks to the split-based branch and bound.
- z_{lb}: the lower bound described in the first chapter (subsection 1.3.2.4).
- ILP: objective value given by the ILP out of 10.000 seconds at most.
- ILP time: CPU time (in seconds) needed to give the optimal solution or 10.000 seconds if the ILP cannot find the optimal solution.
- ILP GAP (%): optimality GAP given by CPLEX. It could be interpreted as the distance from optimality.
- B&B time: CPU time (in seconds) needed by the Branch and bound to terminate and obtain an optimal solution.

Table 5.1: ILP on small low-density instances.

Instance	d_s	z_{lb}	ILP	ILP time	ILP Gap	B&B	B&B time
p14-10-1	8.76	9	11	4512	0	11	1372
p14-10-2	9.89	8	10	9558	0	10	79
p14-10-4	9.89	9	10	10000	17.04	10	809
p14-10-5	9.89	8	11	10000	26.06	11	940
p14-10-6	9.89	8	11	10000	25.29	10	256
p14-10-7	9.89	8	9	1482	0	9	374
p14-10-8	12.08	9	11	9491.69	0	11	147
p14-10-9	9.89	9	10	1021	0	10	374
p14-10-10	8.79	8	9	10000	0	9	269

Table 5.2: ILP on small medium-density instances.

Instance	d_s	z_{lb}	ILP	ILP time	ILP Gap	B&B	B&B time
p14-25-1	16.48	8	10	7336	0	10	598
p14-25-2	21.97	8	10	406	0	10	93
p14-25-3	21.97	8	10	585	0	10	74
p14-25-4	18.68	9	10	1543	0	10	27
p14-25-5	19.78	8	11	511	0	11	213
p14-25-6	25.27	8	10	498	0	10	13
p14-25-7	18.68	9	10	2772	0	10	51
p14-25-8	20.87	9	9	1822	0	9	31
p14-25-9	23.07	8	10	636	0	10	52
p14-25-10	17.58	8	10	2658	0	10	36

Table 5.3: ILP on small high-density instances.

Instance	d_s	z_{lb}	ILP	ILP time	ILP Gap	B&B	B&B time
p14-40-1	36.26	8	11	229	0	11	2
p14-40-2	29.67	7	9	1778.32	0	9	15
p14-40-3	24.17	8	10	4006	0	10	163
p14-40-4	29.67	8	10	322	0	10	41
p14-40-5	26.37	8	10	838	0	10	30
p14-40-6	38.46	8	10	104	0	10	6
p14-40-7	26.37	7	9	1368	0	9	12
p14-40-8	26.37	8	9	491	0	9	13
p14-40-9	27.47	8	10	333.97	0	10	20
p14-40-10	29.67	8	9	409	0	9	10

The following conclusions can be drawn from the tables:

- ILP solves 26 instances out of 30. We remark that an optimal solution is found for all the instances except p14-10-6.
- The branch and bound algorithm solves all the 30 instances within much smaller CPU times than the ILP.
- We notice that the ILP and the B&B perform better on instances with higher density. This can be understood since constraints reduce the combinatorics of the problem.

- We may not compare the ILP results with the ILP from Essafi et al. (2010) since the solver technology has evolved since this publication. However, for information, we note that in this latter publication only 17 out of 30 instances have been solved to optimality.
- The computation times are quite high for such small instances which leads us to consider hybrid methods to tackle larger instances.

5.2.3 Heuristics

We have suggested in the thesis two heuristics to compute a feasible solution for the RTLBP, namely:

- **H-BFSL** is an approximation algorithm of type Balance-First Sequence-Last. The balancing subproblem is solved thanks to an ILP then the sequencing subproblem is solved thanks to dynamic programming. The two steps are iterated thanks to a dynamic programming algorithm.
- **H-SFBL** is a split-based heuristic of type Sequence-First Balance-Last. A compatible giant sequence is computed at first, then split is used to obtain a solution.

The two precedent heuristics are not meant to be used as standalone resolution approaches and are only used as construction methods. We evaluate in this subsection the ability of the heuristics to compute a starting feasible solution in reasonable CPU time.

We first run the two heuristics on the instances: (p14-40-1)-(p14-40-10). Table 5.4 shows the results.

We indicate in the table the objective values and the CPU times (in seconds) of H-BFSL and H-SFBL denoted respectively by # machines and time. H-BFSL is an iterative algorithm, #iter indicates the number of iterations needed by H-BFSL to converge, i.e. the number of constraints that has been added to the relaxed ILP. B&B indicates the objective values of the optimal solutions.

Table 5.4: Performance of heuristics H-BFSL and H-SFBL.

		H-BFSL		H-SFBL		
Instance	B&B	# machines	time	#iter	# machines	time
p14-40-1	11	12	1.75	3	12	0.2
p14-40-2	9	12	0.39	1	10	0.2
p14-40-3	10	11	5.06	7	10	0.2
p14-40-4	10	12	0.77	2	11	0.2
p14-40-5	10	10	1.35	4	10	0.1
p14-40-6	10	11	0.30	2	11	0.2
p14-40-7	9	11	0.21	1	9	0.2
p14-40-8	9	10	0.86	2	9	0.2
p14-40-9	10	11	0.70	1	12	0.2
p14-40-10	9	12	0.85	1	10	0.2

We can infer the following from Table 5.4:

- Both heuristics are able to find a feasible solution for the ten instances within reasonable computation time.
- CPU time of H-SFBL is apparently lower than that of H-BFSL. Much of the CPU time of H-BFSL comes from the resolution of the relaxed ILP from the balancing step.
- Few iterations are needed by H-BFSL to converge. This tendency will be approved by the results on large-size instances.
- Even if H-BFSL is a 2-approximation algorithm, it seems that the solutions given by H-SFBL are of lower objective value than those of H-BFSL. We cannot conclude however that H-SFBL gives a better starting solution for a metaheuristic compared to H-BFSL.
- H-SFBL can retrieve an optimal solution for 4 instances out of 10. It may be seen as a good score for an heuristic but also justifies the use of some metaheuristic-based improvement methods (next subsection).

We run the heuristics H-BFSL and H-SFBL on large-size instances. Table 5.5 shows the results. The objective value (# machines) is given for both BFSL and SFBL. Regarding H-BFSL, the best solution out of 1200 seconds is taken from the integer linear programming solver (Balancing step). The sequencing step time is negligible in comparison with the balancing step. Regarding H-SFBL, 10 independent

runs are performed since the method is stochastic. The objective value corresponding to the first random seed is given in # machines. This solution will be used to initiate the experiments in the following subsection (split-based metaheuristic). The mean and the standard deviation of the objective value over 10 independent runs are given in Mean and Std deviation.

Table 5.5: Performance of heuristics H-BFSL and H-SFBL on big-size instances.

	H-BFSL		H-SFBL		
Instance	# machines	#iter	# machines	Mean	Std deviation
A1	42	1	37	39.3	1.41
A2	30	1	39	38.5	1.68
A3	37	1	38	39.1	1.29
A4	35	1	38	37.8	1.24
A5	42	1	41	39.4	1.74
A6	44	1	41	38.9	1.3
A7	46	1	41	39.9	1.13
A8	38	1	38	38.6	1.2
A9	39	1	40	38.5	1.36
A10	42	1	41	40.5	0.92
A11	37	1	40	38.6	0.79
A12	41	1	40	39	0.77
A13	45	1	38	40.4	1.56
A14	36	1	39	38.8	1.24
A15	38	1	39	39.2	0.74

The following conclusions can be drawn from the table:

- H-BFSL is able to find a feasible solution for all the instances within 1200 seconds. Besides, I-BSFL converges within a single iteration for these large-size instances: the set-up times are low (compared to the processing times) and homogeneous, which makes the estimate (given by the relaxed model) of good quality.
- H-SFBL can compute a starting compatible giant sequence for all the instances and for all the 10 random seeds used. Besides it converges in less than 1 min for most instances and random seeds.
- In regards to the comparison of objective values between H-BFSL and H-SFBL, no clear tendency could be inferred. However, we give a clear advantage to H-SFBL due to the relatively low CPU time.
- The mean and low standard deviation demonstrate the robustness of H-SFBL.

We use H-BFSL as the initial solution for M-BFSL because chronologically, we developed the BFSL type approaches first and it was only in the second part of the thesis that we developed the split-based approaches. H-SFBL is used as the initial solution for the split-based metaheuristic.

5.2.4 Metaheuristics

We experiment the matheuristic M-BFSL and the split-based metaheuristic on the large-size instances. We also compare those methods with a genetic algorithm from literature: Borisovsky, Delorme, and Dolgui (2013).

5.2.4.1 M-BFSL

Starting from the initial solution given by H-BFSL, the improvement phase (M-BFSL) is performed with different values of δ (number of iterations for each temperature). The different parameters are determined experimentally:

- Initial temperature is set to 10 and the decreasing scheme is as follows:

$$\begin{cases} T = T - 0.0036 & \text{If } T > 1 \\ T = T - 0.0004 & \text{Otherwise.} \end{cases}$$

- Initial weights w_1, w_2, w_3 of the neighborhood moves V_1 (insertion), V_2 (merger), V_3 (split move) are set to 1, then they are increased by adding one if the associated neighborhood move yields a neighbor improving the best recorded solution.

- We perform δ iterations for each temperature (inner loop). Experiments are done with different values of δ, $\delta = 1, 20, 50, 100$. For $\delta = 1$, it is an inhomogeneous simulated annealing.

Results are given is Table 5.6, Table 5.7, Table 5.8 and Table 5.9 with respectively $\delta = 1$, $\delta = 20$, $\delta = 50$ and $\delta = 100$. Since M-BFSL is stochastic, 10 independent runs are performed.

We use the following notations in the tables:

- Min: minimum number of machines obtained by the matheuristic over 10 independent runs.

- Max: maximum number of machines obtained by the matheuristic over 10 independent runs.

- Mean: average number of machines obtained by the matheuristic over 10 independent runs.
- σ: Standard deviation of the number of machines obtained by matheuristic over 10 independent runs.
- GA: best solution (with the minimum number of machines) obtained by the genetic algorithm (Borisovsky, Delorme, and Dolgui (2013)) over 10 independent runs. The algorithm uses a giant sequence to encode a solution. Either an heuristic or a MIP is used to decode it.

Table 5.6: Experiments with $\delta = 1$

Instance	min	max	mean	σ
A1	29.0	33.0	30.7	1.1
A2	28.0	30.0	29.4	0.663
A3	27.0	32.0	29.8	1.33
A4	28.0	30.0	29.2	0.6
A5	29.0	31.0	30.3	0.781
A6	28.0	31.0	29.6	0.917
A7	30.0	32.0	31.6	0.663
A8	30.0	32.0	31.0	0.775
A9	28.0	31.0	29.8	0.872
A10	31.0	35.0	33.0	1.1
A11	29.0	31.0	29.8	0.748
A12	30.0	33.0	31.4	0.917
A13	31.0	32.0	31.4	0.49
A14	29.0	32.0	30.8	0.872
A15	29.0	32.0	30.1	0.831

Table 5.7: Experiments with $\delta = 20$

Instance	min	max	mean	σ
A1	26.0	29.0	27.9	1.04
A2	26.0	27.0	26.4	0.49
A3	26.0	28.0	26.6	0.663
A4	25.0	27.0	26.1	0.539
A5	26.0	28.0	27.2	0.748
A6	26.0	28.0	26.8	0.6
A7	28.0	29.0	28.3	0.458
A8	27.0	28.0	27.8	0.4
A9	26.0	28.0	26.8	0.6
A10	29.0	31.0	29.8	0.748
A11	26.0	28.0	26.9	0.539
A12	27.0	28.0	27.8	0.4
A13	28.0	29.0	28.7	0.458
A14	27.0	28.0	27.2	0.4
A15	26.0	28.0	27.2	0.6

Table 5.8: Experiments with $\delta = 50$

Instance	min	max	mean	σ
A1	27.0	29.0	27.6	0.8
A2	25.0	27.0	25.9	0.539
A3	25.0	26.0	25.8	0.4
A4	25.0	27.0	25.9	0.539
A5	26.0	27.0	26.4	0.49
A6	25.0	27.0	26.1	0.539
A7	26.0	28.0	27.3	0.64
A8	27.0	28.0	27.3	0.458
A9	26.0	27.0	26.3	0.458
A10	28.0	30.0	29.1	0.539
A11	25.0	27.0	26.3	0.64
A12	26.0	28.0	27.3	0.64
A13	27.0	29.0	28.1	0.539
A14	26.0	28.0	26.5	0.671
A15	26.0	27.0	26.4	0.49

Table 5.9: Experiments with $\delta = 100$

Instance	min	max	mean	σ
A1	26.0	28.0	26.9	0.539
A2	25.0	26.0	25.6	0.49
A3	25.0	27.0	25.5	0.671
A4	25.0	26.0	25.7	0.458
A5	25.0	27.0	26.3	0.64
A6	25.0	26.0	25.6	0.49
A7	27.0	28.0	27.4	0.49
A8	27.0	28.0	27.2	0.4
A9	26.0	27.0	26.2	0.4
A10	28.0	30.0	28.8	0.6
A11	25.0	26.0	25.4	0.49
A12	27.0	28.0	27.2	0.4
A13	27.0	28.0	27.5	0.5
A14	26.0	27.0	26.4	0.49
A15	25.0	26.0	25.9	0.3

We notice from Tables 5.6, 5.7, 5.8 and 5.9 that increasing δ has an impact on the objective values. Indeed, we notice that when δ increases, the min, max, mean and standard deviation values get smaller. However, the improvement of results is at the expense of CPU time. Indeed, experiments with $\delta = 1$ (inhomogeneous simulated annealing) only take 25 seconds of CPU time for each instance while experiments with $\delta = 100$ (100 iterations for each temperature) takes about 3000 seconds for each instance.

Small values of the standard deviation (usually smaller than 1) demonstrate the robustness of the proposed method.

Table 5.10 shows the average probability distributions of the neighborhood moves (V_1, V_2, V_3) after the run of the proposed matheuristic. The probability distributions are denoted by (w_1, w_2, w_3).

Table 5.10: Average probability distributions of neighborhood moves after the run of the proposed matheuristic

	w_1, w_2, w_3			
Inst.	$\delta = 1$	$\delta = 20$	$\delta = 50$	$\delta = 100$
A1	(0.75, 0.16 , 0.08)	(0.76, 0.17 , 0.07)	(0.78, 0.15, 0.07)	(0.78, 0.14 , 0.07)
A2	(0.35, 0.33, 0.33)	(0.38, 0.33, 0.29)	(0.39, 0.33, 0.28)	(0.41, 0.33, 0.26)
A3	(0.49, 0.38, 0.13)	(0.60, 0.29, 0.11)	(0.61, 0.29, 0.10)	(0.61, 0.29, 0.10)
A4	(0.58, 0.26, 0.15)	(0.64, 0.24, 0.12)	(0.65, 0.23, 0.12)	(0.67, 0.21, 0.12)
A5	(0.76, 0.16, 0.08)	(0.79, 0.14, 0.07)	(0.78, 0.15, 0.07)	(0.81, 0.12, 0.06)
A6	(0.77, 0.16, 0.07)	(0.80, 0.14, 0.06)	(0.80, 0.14, 0.06)	(0.80, 0.14, 0.06)
A7	(0.80, 0.14, 0.04)	(0.82, 0.12, 0.06)	(0.83, 0.11, 0.06)	(0.83, 0.11, 0.05)
A8	(0.64, 0.23, 0.13)	(0.66, 0.23, 0.10)	(0.69, 0.20, 0.10)	(0.67, 0.23, 0.10)
A9	(0.57, 0.33, 0.10)	(0.60, 0.32, 0.08)	(0.60, 0.31, 0.08)	(0.61, 0.31, 0.08)
A10	(0.78, 0.11, 0.10)	(0.80, 0.11, 0.09)	(0.79, 0.12, 0.08)	(0.82, 0.09, 0.08)
A11	(0.56, 0.31, 0.13)	(0.65, 0.25, 0.10)	(0.65, 0.25, 0.10)	(0.67, 0.23, 0.09)
A12	(0.54, 0.36, 0.10)	(0.60, 0.32, 0.08)	(0.61, 0.31, 0.08)	(0.63, 0.29, 0.08)
A13	(0.77, 0.16, 0.06)	(0.80, 0.14, 0.06)	(0.80, 0.14, 0.06)	(0.81, 0.13, 0.05)
A14	(0.39, 0.46, 0.16)	(0.47, 0.41, 0.13)	(0.45, 0.43, 0.12)	(0.49, 0.39, 0.11)
A15	(0.51, 0.37, 0.12)	(0.59, 0.31, 0.10)	(0.62, 0.29, 0.10)	(0.60, 0.31, 0.09)

Initially, the neighborhood moves have the same probability to be chosen in the balancing perturbation. Table 5.10 shows that the neighborhood move V_1 has the greatest probability to be chosen then V_2. The probability of V_3 is close to 0.

Since in our algorithm the probabilities are only increased if the neighborhood move yields a neighbor strictly improving the best recorded solution, the table shows that neighborhood move V_1 then V_2 are the most efficient for improving a solution. V_3 almost never improves a solution, its purpose is above all to introduce diversity into the search process.

Table 5.11 compares the performance of the suggested matheuristic with the genetic algorithm from literature (Borisovsky, Delorme, and Dolgui (2013)). In this last, only the minimum cost over 10 independent runs is reported. Therefore, we report the min over 10 independent runs of the proposed matheuristic for different values of δ.

Table 5.11: Comparison between the matheuristic and the genetic algorithm of the literature

	GA (min)	H-BFSL	M-BFSL (min)			
Instance			$\delta = 1$	$\delta = 20$	$\delta = 50$	$\delta = 100$
A1	33	42	29	**26**	27	**26**
A2	33	30	28	26	**25**	**25**
A3	31	37	27	26	**25**	**25**
A4	29	35	28	**25**	**25**	**25**
A5	32	42	29	26	26	**25**
A6	32	44	28	26	**25**	**25**
A7	34	46	30	28	**26**	27
A8	31	38	30	**27**	**27**	**27**
A9	30	39	28	**26**	**26**	**26**
A10	32	42	31	29	**28**	**28**
A11	30	37	29	26	**25**	**25**
A12	31	41	30	27	**26**	27
A13	33	45	31	28	**27**	**27**
A14	31	36	29	27	**26**	**26**
A15	33	38	29	26	26	**25**

Table 5.11 gives a clear advantage to the proposed matheuristic compared to the genetic algorithm of the literature Borisovsky, Delorme, and Dolgui (2013). Indeed, H-BFSL usually gives a solution of superior objective value compared to the genetic algorithm but this solution is rapidly improved with the M-BFSL. The results of the genetic algorithm are already outperformed with $\delta = 1$ which runs in 25 seconds. With $\delta = 100$ (3000 seconds), the improvement is even more drastic compared to literature and approaches 20% for some instances. Besides, even the max is lower than the min of GA for some instances.

However, we must note that this very advantageous results are obtained at the expense of CPU time. Indeed, the genetic algorithm runs in 900 seconds whereas the constructive phase (H-BFSL) runs in 1200 seconds and the improvement phase CPU time varies from 25 seconds (for $\delta = 1$) to 3000 seconds (for $\delta = 100$) as shown in Table 5.12. The proposed matheuristic has a greater CPU time than the genetic algorithm from literature because it embeds several components from mathematical programming (integer linear programming, constraint generation and dynamic programming). These components are an important source of the good performance of the proposed method. Table 5.12 reports the average CPU times of the algorithms.

Table 5.12: Average approximate CPU times on A1-A15 instances (in seconds)

Algorithm	Average approximate CPU time
H-BFSL	1200
Improvement phase with $\delta = 1$	25
Improvement phase with $\delta = 20$	1000
Improvement phase with $\delta = 50$	1700
Improvement phase with $\delta = 100$	3000
Genetic algorithm from literature	900

We summarize the conclusions drawn from the experiments relative M-BFSL as follows:

- H-BFSL can compute a feasible solution for all the big-size instances within 1200 seconds (at most).
- As expected, the higher number of iterations we perform for each temperature, the better results we get.
- The insertion move gives better results than the merger and the split moves. However, we believe that the merger and the split moves offer two additional degrees of freedom and prevent to be stuck in a local optimum.
- The proposed method is robust since the standard deviation is low.
- Significantly better results are obtained compared to the genetic algorithm from literature.

5.2.4.2 Split-based ILS

We present in this section the split-based iterated local search. The same set of instances than previously is considered.

Tables 5.13 and 5.14 show the results of the split-based ILS with different configurations. ILS(x,y) means x iterated local searches of y iterations each (i.e. y neighbors visited in each local search). Starting from an initial compatible giant sequence, 10 independent runs of the iterated local search are performed. In these tables, min refers to the minimum number of machines obtained by the split-based ILS over 10 independent runs. This columns allows to compare with the genetic algorithm from the literature, because in this last only the min is reported. The maximum, mean and the standard deviation (denoted σ) are also given.

Table 5.13: Split-based ILS with different configurations: ILS(100,500) means 100 iterated local searches of 500 iterations each

	ILS(100,500)				ILS(100,1'000)				ILS(100,2'000)			
Instance	min	max	mean	σ	min	max	mean	σ	min	max	mean	σ
A1	30.0	31.0	30.1	0.3	29.0	30.0	29.5	0.5	**28.0**	29.0	28.9	0.3
A2	**27.0**	29.0	28.1	0.539	**27.0**	28.0	27.7	0.458	**27.0**	28.0	27.2	0.4
A3	28.0	29.0	28.5	0.5	**27.0**	28.0	27.5	0.5	**27.0**	28.0	27.4	0.49
A4	**27.0**	29.0	28.3	0.64	**27.0**	28.0	27.6	0.49	**27.0**	28.0	27.3	0.458
A5	**28.0**	30.0	29.2	0.748	**28.0**	30.0	28.9	0.539	**28.0**	29.0	28.1	0.3
A6	28.0	30.0	29.0	0.632	**27.0**	29.0	28.1	0.539	**27.0**	28.0	27.9	0.3
A7	29.0	31.0	29.9	0.7	29.0	30.0	29.6	0.49	**28.0**	30.0	29.0	0.447
A8	29.0	31.0	29.6	0.663	29.0	30.0	29.6	0.49	**28.0**	30.0	29.0	0.447
A9	28.0	30.0	28.4	0.663	**27.0**	29.0	28.0	0.447	**27.0**	28.0	27.4	0.49
A10	31.0	32.0	31.5	0.5	**30.0**	32.0	31.2	0.6	31.0	32.0	31.1	0.3
A11	28.0	29.0	28.5	0.5	**27.0**	29.0	28.0	0.447	**27.0**	28.0	27.3	0.458
A12	29.0	31.0	29.9	0.7	29.0	30.0	29.5	0.5	**28.0**	30.0	28.8	0.6
A13	30.0	32.0	30.3	0.64	**29.0**	30.0	29.4	0.49	**29.0**	30.0	29.1	0.3
A14	28.0	30.0	29.0	0.447	28.0	29.0	28.5	0.5	**27.0**	28.0	27.9	0.3
A15	**28.0**	29.0	28.7	0.458	**28.0**	30.0	28.8	0.6	**28.0**	29.0	28.1	0.3

Table 5.14: Split-based ILS with different configurations: ILS(50,500) means 100 iterated local searches of 1'000 iterations each

	ILS(50,500)				ILS(100,10'000)				ILS(1'000,1'000)			
Instance	min	max	mean	σ	min	max	mean	σ	min	max	mean	σ
A1	30.0	31.0	30.3	0.458	**28.0**	29.0	28.2	0.4	**28.0**	30.0	29.0	0.447
A2	27.0	29.0	28.5	0.671	**26.0**	27.0	26.6	0.49	**26.0**	28.0	27.0	0.447
A3	29.0	29.0	29.0	0.0	**26.0**	27.0	26.9	0.3	27.0	28.0	27.3	0.458
A4	28.0	29.0	28.7	0.458	**26.0**	27.0	26.6	0.49	27.0	28.0	27.2	0.4
A5	29.0	31.0	29.8	0.6	**27.0**	28.0	27.4	0.49	**27.0**	29.0	28.1	0.539
A6	29.0	30.0	29.5	0.5	**26.0**	28.0	27.0	0.447	27.0	28.0	27.3	0.458
A7	30.0	32.0	30.8	0.872	**28.0**	29.0	28.3	0.458	**28.0**	30.0	29.0	0.447
A8	29.0	31.0	30.0	0.632	**28.0**	29.0	28.3	0.458	**28.0**	30.0	29.0	0.447
A9	28.0	30.0	28.6	0.663	**27.0**	27.0	27.0	0.0	**27.0**	28.0	27.2	0.4
A10	31.0	33.0	32.1	0.539	**29.0**	30.0	29.9	0.3	30.0	31.0	30.5	0.5
A11	28.0	29.0	28.7	0.458	**26.0**	27.0	26.9	0.3	27.0	28.0	27.1	0.3
A12	30.0	31.0	30.3	0.458	**28.0**	29.0	28.3	0.458	**28.0**	30.0	28.8	0.6
A13	30.0	32.0	30.8	0.748	**28.0**	29.0	28.2	0.4	**28.0**	29.0	28.9	0.3
A14	29.0	30.0	29.4	0.49	**27.0**	28.0	27.3	0.458	**27.0**	28.0	27.5	0.5
A15	28.0	30.0	29.0	0.632	**27.0**	28.0	27.3	0.458	**27.0**	29.0	28.1	0.539

Table 5.13 shows 3 configurations of the ILS with 100 iterated local searches. The number of iterations in the local searches varies from 500 to 2000. Clearly, with a fixed number of local searches, increasing the number of iterations in the local search leads to improving the results (min, max, mean and standard deviation decreases).

The idea of Table 5.14 is to investigate the behavior of the ILS when running for a long time by running ILS(100,10'000) and ILS(1'000,1'000). ILS(100,10'000) is giving the best results when compared with all the configurations, but it requires also a large CPU time: 5000". ILS(100,10'000) outperforms ILS(1'000,1'000) despite ILS(1'000,1'000) taking more time to run: 6000". The table also contains a configuration with small x and y (ILS(50,100)) to investigate the quality of the solution when the number of iterations is low. This configuration runs in 250" and already outperforms the algorithm of the literature. From Tables 5.13 and 5.14, we notice that the standard deviation is very low (almost always below 0.5) which demonstrates the robustness of the proposed method.

Table 5.15 compares the results obtained by the split-based ILS with the genetic algorithm from literature.

Table 5.15: Performance of split-based metaheuristic vs literature.

	Genetic algo. of the literature	Split-based ILS		
Instance	after 900" of execution time	after 250"	after 1000"	after 5000"
A1	33	30	29	28
A2	33	27	27	26
A3	31	29	27	26
A4	29	28	27	26
A5	32	29	28	27
A6	32	29	27	26
A7	34	30	29	28
A8	31	29	29	28
A9	30	28	27	27
A10	32	31	30	29
A11	30	28	27	26
A12	31	30	29	28
A13	33	30	29	28
A14	31	29	28	27
A15	33	28	28	27

In the column denoted "Split-based ILS", the best result out of 10 independent runs is reported. Three configurations are reported: ILS(50,500) (running in 250"),

ILS(100,1'000) (running in 1000") and ILS(100,10'000) (running in 5000"). Despite the genetic algorithms runs in 900", a clear improvement is already observed from 250" of execution time of the split-based ILS.

Table 5.16 reports the CPU time of the different algorithms. We notice that the perturbation takes about 9% of the total computation time of the ILS.

Table 5.16: Approximate CPU times of the algorithms with different configurations.

Configuration	Average time
ILS(50,500)	250"
ILS(100,500)	450"
ILS(100,1'000)	1000"
ILS(100,2'000)	1300"
ILS(100,10'000)	5000"
ILS(1'000,1'000)	6000"
Genetic algo. of the literature	900"

The split-based ILS performs better than a genetic algorithm from literature. It can be due to the reduction of the space search allowed by split. We develop the same type of methods for RALBP-2 and SDRALBP-2 in the next section.

5.3 Robotic Assembly lines

We present in this section experiments relative to the split-based ILS for the RALBP-2 and the SDRALBP-2. We are content to authorize the selection of the same type of robots to multiple workstations as it is done in Nilakantan et al. (2015) and Borba, Ritt, and Miralles (2018). We recall that the objective is to minimize the cycle time.

5.3.1 RALBP-2

We first experiment the split-based ILS for the RALBP-2 on instances taken from literature. The objective to be optimized is the cycle time and sequence-dependent setup times are null.

5.3.1.1 Instances

We take the instances for RALBP-2 from Gao et al. (2009). The set of instances contain small-size, medium-size and big-size instances. The number of operations varies from $n = 11$ to $n = 148$. For each number of operations, different number of types of robots are considered.

The considered instances were used by two recent studies in literature about RALBP-2:

- Nilakantan et al. (2015): the authors suggest sophisticated bio-inspired metaheuristics: Particle swarm optimization and an hybrid cuckoo search-PSO. A solution is represented thanks to giant sequence, the consecutive assignment procedure from Levitin, Rubinovitz, and Shnits (2006) is used to decode a solution. Compared to split, the procedure has the disadvantage of not being polynomial. A special attention will be paid to the comparison with this method in order to prove the relevance of using split as a decoding procedure in a metaheuristic.
- Borba, Ritt, and Miralles (2018): the authors suggest a branch and bound and remember algorithm. The suggested exact method tends to be very efficient since many instances are optimally solved.

5.3.1.2 Split-based ILS

The experiments relative to RALBP-2 are shown on Table 5.17. The following notations are used in the table:

- Nilakantan et al. (2015): obtained objective value C (cycle time) in Nilakantan et al. (2015).

- Borba, Ritt, and Miralles (2018): obtained cycle time in Borba and Ritt (2014).
- LS: obtained cycle time with a split-based stochastic local search where 10 000 neighbors are visited.
- ILS: obtained cycle time with a split-based ILS of 10 iterated local searches where 10 000 neighbors are visited in each.

Table 5.17: Instances with null setup times

n	s_{max}, n_r	Nilakantan et al. (2015)	Borba, Ritt, and Miralles (2018)	LS	ILS
11	4	-	-	126	126
25	3	503	503*	**503**	**503**
	4	327	291*	294	**291**
	6	200	194*	195	**194**
	9	110	109*	**109**	**109**
35	4	341	341*	342	**341**
	5	332	329*	**329**	**329**
	7	211	201*	**201**	**201**
	12	103	93*	98	**93**
53	5	449	449*	**449**	**449**
	7	294	283*	284	**283**
	10	221	203*	213	**203**
	14	142	134*	137	**134**
70	7	430	391*	401	392
	10	264	233*	238	234
	14	194	170*	183	176
	19	140	121*	129	126
89	8	460	436*	446	445
	12	320	296*	309	301
	16	219	205*	211	207
	21	170	156*	164	161
111	9	523	468	491	472
	13	321	275	295	287
	17	240	212	230	224
	22	182	154	173	166
148	10	593	550	583	-
	14	419	351	376	-
	21	273	225	244	-
	29	189	154	171	-

**: optimal solution*

The two following conclusions can be drawn from the table:

- Compared to Nilakantan et al. (2015), we obtain far better solutions even with the simple split-based local search (LS). This result is even more pronounced for the split-based ILS (ILS).
- Even if the split-based ILS is only an approximate method, it can retrieve many optimal objective values obtained by Borba, Ritt, and Miralles (2018).
- The split-based ILS (ILS) gives better results than the split-based local search (LS) at the expense of 10 times higher CPU time.

The CPU times needed to obtain LS are comparable to those of Nilakantan et al. (2015). The CPU time is proportional to the number of neighbors visited. It barely equals to 10'000 times the CPU time needed to perform a single split run for LS and 10 times more for ILS. For instances with $n = 11$ to $n = 70$, the CPU times are quite reasonable, less than 5 microseconds are necessary to run the split algorithm. We notice that the CPU times grow then drastically when increasing the value of n but remain reasonable up to $n = 148$. For $n = 148$ and $n_r = 29$, 0.1 seconds is needed to perform a single split run. For instances with $n = 297$, this value can rise up to 1.7 second for the biggest values of n_r.

In conclusion, the split-based local search and the split-based ILS outperform metaheuristics from literature (Nilakantan et al. (2015)) and retrieve many optimal objective values obtained with a latest exact method from literature (Borba and Ritt (2014)). This demonstrates the relevance of using split as a decoding procedure in a metaheuristic instead of existing decoding procedures. split is very fast for instances with $n = 1..70$ (small instances) and reasonably fast for instances with $n = 89..148$ (medium and big instances). However, for very big instances ($n = 297$) and high number of possible robots, the CPU times of the split algorithm exceeds one second which limits the number of neighbors that can be visited in some metaheuristic.

5.3.2 Sequence-dependent Robotic Assembly Line Balancing Problem-2

We now experiment the split-based ILS for the Sequence-dependent Robotic Assembly Line Balancing Problem-2 on instances taken from literature. The objective to be optimized is always the cycle time and sequence-dependent setup times are now considered.

5.3.2.1 Instances

To the best of our knowledge, Janardhanan et al. (2019) is the only study published in literature dealing with SDRALBP-2. The authors suggest a new set of instances that are derived from the instances of Gao et al. (2009) by adding sequence-dependent setup times.

Two classes of instances are considered (Janardhanan et al. (2019)):

- Instances with low setup times: they correspond to the instances of Gao et al. (2009) to which setup times are added and generated randomly and uniformly within $[0, 0.25 * min_{i,r} d_{i,r}]$.
- Instances with high setup times: they correspond to the instances of Gao et al. (2009) to which setup times are added and generated randomly and uniformly within $[0, 0.75 * min_{i,r} d_{i,r}]$.

Unfortunately, Janardhanan et al. (2019) suppose the assumption "only one robot per type". Their method gives only an upper bound for our problem since they don't allow themselves to use a type of robot several times in different workstations.

5.3.2.2 Split-based ILS

The experiments relative to SDRALBP-2 are shown on Table 5.18. We present there the results for low setup and high setup instances. We do not have the results of Janardhanan et al. (2019) for instances with $n \geq 89$.

Table 5.18: Instances with low and high setup times

Instance		Low Setup			High Setup		
n	$s_{max}(= n_r)$	Janardhanan et al. (2019)	LS	ILS	Janardhanan et al. (2019)	LS	ILS
11	4	137	137	137	152	152	151
25	3	516	536	535	579	584	579
	4	346	303	303	380	343	343
	6	227	203	198	242	216	214
	9	131	116	116	142	125	121
35	4	462	352	352	494	376	374
	5	355	335	335	392	368	365
	7	237	208	208	261	225	224
	12	118	100	100	131	113	113
53	5	574	471	461	619	508	486
	7	334	286	286	359	319	308
	10	256	223	213	276	244	237
	14	170	146	143	185	155	155
70	7	469	426	408	507	466	448
	10	282	252	246	309	270	266
	14	211	189	182	233	206	202
	19	158	135	131	175	145	144
89	8	-	465	458	-	491	491
	12	-	318	308	-	344	344
	16	-	224	220	-	240	238
	21	-	166	163		184	181
111	9	-	517	495	-	541	521
	13	-	306	301	-	329	228
	17	-	233	233	-	257	255
	22	-	178	172	-	193	191
148	10	-	610	-	-	685	-
	14	-	391	-	-	431	-
	21	-	256	-	-	289	-
	29	-	182	-	-	202	-

In addition to the conclusions made in the previous subsection, the following can be inferred:

- The cycle time LS (and ILS) is much smaller than the cycle time from Janardhanan et al. (2019) for most instances. We can deduce from it that the assumption allowing the use of the same robot type in multiple workstations yields better efficiency.
- By comparing the cycle times between the instances without setup, with low setup and with high setup, we can deduce that the consideration of sequence-dependent setup times has a real impact on the cycle time. This justifies that the sequence-dependent setup times cannot be negligible and should be taken into consideration in the modeling/optimization step of the Robotic Assembly Line Balancing Problem.

Experiments relative to the assumption "one robot per type" are presently being held. The results could then be compared with Janardhanan et al. (2019).

5.4 Conclusion

We have tested in this chapter the suggested resolution approaches for the RTLBP, the RALBP-2 and the SDRALBP-2.

Benchmark instances have been used for the experiments. Some results from the literature have been improved, namely Borisovsky, Delorme, and Dolgui (2013) in regards to the RTLBP and Nilakantan et al. (2015) in regards to RALBP-2.

We can conclude from the experiments the following:

- In regards to Reconfigurable Transfer Lines:
  - The ILP can solve 26 out of the 30 small instances but the computation time is quite high. The branch and bound algorithm solved all the instances within fewer CPU time.
  - H-SFBL is more efficient that H-BFSL since it can find a feasible solution for all the large-size instances within much fewer CPU time.
  - Both the matheuristic M-BFSL and the split-based metaheuristic outperform a genetic algorithm from literature.
- In regards to robotic assembly lines:

- In regards to RALBP-2, the split-based metaheuristic clearly outperforms bio-inspired metaheuristics from literature and retrieve most optimal cycle times values obtained by a branch and bound algorithm.
- Unfortunately, no existing method for SDRALBP-2 allows to evaluate the split-based metaheuristic. Janardhanan et al. (2019) gives only an upper bound. Experiments show that the obtained cycle times are far below this bound.

All in all, using split as a decoding procedure in a metaheuristic tends to be efficient for the RTLBP, the RALBP-2 and the SDRALBP-2.

Conclusion and perspectives

General conclusion

Assembly line balancing problems were studied in the frame of this thesis. They consist in assigning a set of operations to a set of workstations placed along a line while respecting some constraints and minimizing some objectives. The problem is old-rooted; however, few studies consider this problem jointly with the (operations) sequencing problem and the (robot) selection problem.

We first studied the balancing and the sequencing problems jointly. Besides, further industrial constraints and parallel workstations are also considered, which led us to the Reconfigurable Transfer Line Balancing Problem (RTLBP) which is an existing problem from literature.

Since two decision problems are involved, the following three types of approaches have been proposed for this problem:

- Integrated approach: the balancing and the sequencing decisions are addressed at the same time. For this type of approach, we have proposed an integer linear programming model.
- Approaches of type BFSL ("Balance-First, Sequence-Last"): Lahrichi et al. (2020b). The balancing decision is addressed at first. The operations are afterwards sequenced in each workstation. We suggest an approximation algorithm and a matheuristic of this type. The approximation algorithm performs the balancing step by means of ILP and the sequencing step by means of dynamic programming. The two steps are piloted thanks to a constraint generation algorithm. The matheuristic uses the previous heuristic to compute an initial solution and improves it thanks to an adaptive simulated annealing. The three different local search moves used disturb the balancing solution then resequence the operations in the modified workstations.

- SFBL-type approach ("Sequence First, Balance Last"): Lahrichi et al. (2020a). It is the reverse approach of BFSL. The sequencing problem is solved at first by giving a "giant sequence" of operations. This balancing problem is then solved while respecting the giant sequence. Our main contribution is at the level: a polynomial algorithm called "split" for the exact resolution of this second subproblem. An iterated local search exploring the space of giant sequences is proposed where the split algorithm is used to construct and evaluate the sequences. A split-based Branch and Bound algorithm has also been proposed. The experiments carried out on benchmark instances show a clear improvement compared to certain methods in the literature.

The last approach has been retained for the remainder of the thesis to tackle an additional decision: the robot selection problem. In the context of robotic assembly lines, the processing times and the sequence-dependent setup times depend on the type of robot assigned to the workstation. We study the Robotic Assembly Line Balancing Problem-2 (RALBP-2) which includes the balancing decision and the robot selection decision then the Sequence-Dependent Robotic Assembly Line Balancing Problem-2 (SDRALBP-2) which includes in addition to the two previous decisions, the sequencing decision. We consider the objective of minimizing the cycle time with a given maximum number of workstations.

RALBP-2 and SDRALBP-2 have been studied under two different assumptions identified in the literature:

- Many robots per type: in this assumption, we are given a set of types of robots. Each type of robot can be assigned to multiple workstations without any limitation.

- Only one robot per type: in this assumption, we are given a set of robots each of which can be assigned to at most workstation.

We tackle RALBP-2 and SDRALBP-2 thanks to methods of type Sequence-First Balance-And-Select-Last (SFBSL): Lahrichi et al. (2020c). This type of method represents an extension of the Sequence-First Balance-Last type approach proposed for the RTLBP. The split algorithm, intended to solve the particular case where the sequence of operations is given, is adapted for robotic lines. A split-based metaheuristic is suggested. The results on literature instances are positive.

For the three problems studied in the thesis, the major contribution is the split algorithm. This algorithm solves the particular case of a fixed giant sequence relying on the search for an optimal path in some auxiliary graph (Table 5.19). We note

that the particular case has also been solved for other problems in the thesis, in particular for SALBP-2.

Problem	Underlying graph problem	Time complexity
SALBP-2	Constrained min-max path	$O(n^4)$
RTLBP	Constrained shortest path	$O(n^4)$
RALBP-1/SDRALBP-1	Shortest path	$O(n^2)$
RALBP-2/SDRALBP-2	Constrained min-max path	$O(n^4)$

Table 5.19: Underlying graph problems and complexity

The particular case of a fixed giant sequence is often encountered in literature because it allows the reduction of the search space in a metaheuristic where a solution is encoded by a giant sequence. The use of split makes it possible to preserve an optimal solution.

To the best of our knowledge, this particular case has only been solved in the past by means of heuristics (without performance guarantee) or mathematical model (exponential complexity) in Borisovsky, Delorme, and Dolgui (2013) for the RTLBP. The same observation is observed for RALBP-2, the corresponding particular case was solved by an exponential method in Levitin, Rubinovitz, and Shnits (2006) and Nilakantan et al. (2015).

Short-term perspectives

Given the results of the thesis, the following short-term perspectives can be considered:

- **Further improvements of the methods:**

  Split has been proven to integrate well in quite simple metaheuristics. We can investigate its embedding in more sophisticated metaheuristics. The tuning of the metaheuristics can also be considered. We can also consider many improvement tracks for the split-based Branch and bound algorithm. In particular, the search strategy of the graph can be improved. We are also thinking about a tighter lower bound. Once improved, we consider to test it for bigger instances and for the other problems.

- **Assumption "one robot per type":**

  We have seen in Chapter 4, that when dealing with the assumption "one robot per type", the polynomiality is lost. To overcome this issue, speeding-up heuristics were suggested. An alternative solution could be to consider

another vector representing a permutation of robots. Thus, a polynomial algorithm can be obtained to compute the optimal solution respecting a given giant sequence and a given permutation of robots. The two encoding schemes can be compared.

- **Multi-objective Robotic Assembly Line Balancing Problem:**

  Optimization of assembly line is often driven by two contradictory objectives, namely the efficiency and the cost. The efficiency can be represented by the cycle time while the cost can be expressed as a weighted sum of the number of workstations and the cost of the robots used. In this case, we consider that each robot $r \in R$ has a cost c_r. A bi-objective robotic assembly line balancing problem is formulated this way. We prove that *split* can be applied to compute the set of all Pareto optimal solutions corresponding to a given giant sequence. The split is then embedded in a metaheuristic of type NSGA-II. This work is part of an international collaboration with Assistant Professor Jordi Pereira from Aldofo Ibañez University in Santiago, Chile.

Long-term perspectives

The following three long-term perspectives can be considered, they are justified from an industrial point of view:

- In many situations, robots and human operators should cooperate to perform some high added value operations. For this reason, the robotic assembly line balancing problem should integrate human operators to be more realistic. This consideration can bring new elements to the problem such as ergonomics.

- Robots can perform cutting-edge operations and bring considerable benefits in the context of assembly lines. However, their energy consumption is quite critical. It can be beneficial to investigate the considered balancing problems with energetic considerations.

- Reconfigurability is a major issue in Industry 4.0. In that direction, we can consider studying multiple products by investigating the re-balancing problem taking place when the line should be re-balanced for a new product.

Bibliography

Akyol, Sebnem Demirkol, and Adil Baykasoğlu. 2019. "A multiple-rule based constructive randomized search algorithm for solving assembly line worker assignment and balancing problem." *Journal of Intelligent Manufacturing* 30 (2): 557–573.

Allahverdi, Ali, CT Ng, TC Edwin Cheng, and Mikhail Y Kovalyov. 2008. "A survey of scheduling problems with setup times or costs." *European journal of operational research* 187 (3): 985–1032.

Andres, Carlos, Cristobal Miralles, and Rafael Pastor. 2008. "Balancing and scheduling tasks in assembly lines with sequence-dependent setup times." *European Journal of Operational Research* 187 (3): 1212–1223.

Battaïa, Olga, and Alexandre Dolgui. 2013. "A taxonomy of line balancing problems and their solution approaches." *International Journal of Production Economics* 142 (2): 259–277.

Battaïa, Olga, Alexandre Dolgui, Nikolay Guschinsky, and Genrikh Levin. 2012. "A decision support system for design of mass production machining lines composed of stations with rotary or mobile table." *Robotics and Computer-Integrated Manufacturing* 28 (6): 672–680.

Baybars, Ilker. 1986. "A survey of exact algorithms for the simple assembly line balancing problem." *Management science* 32 (8): 909–932.

Beasley, John E. 1983. "Route first—cluster second methods for vehicle routing." *Omega* 11 (4): 403–408.

Becker, Christian, and Armin Scholl. 2006. "A survey on problems and methods in generalized assembly line balancing." *European journal of operational research* 168 (3): 694–715.

Bianco, Lucio, Aristide Mingozzi, Salvatore Ricciardelli, and Massimo Spadoni. 1994. "Exact and heuristic procedures for the traveling salesman problem with

precedence constraints, based on dynamic programming." *INFOR: Information Systems and Operational Research* 32 (1): 19–32.

Blum, Christian, and Cristobal Miralles. 2011. "On solving the assembly line worker assignment and balancing problem via beam search." *Computers & Operations Research* 38 (1): 328–339.

Borba, Leonardo, and Marcus Ritt. 2014. "A heuristic and a branch-and-bound algorithm for the assembly line worker assignment and balancing problem." *Computers & Operations Research* 45 (1): 87–96.

Borba, Leonardo, Marcus Ritt, and Cristóbal Miralles. 2018. "Exact and heuristic methods for solving the robotic assembly line balancing problem." *European Journal of Operational Research* 270 (1): 146–156.

Borisovsky, Pavel A., Xavier Delorme, and Alexandre Dolgui. 2013. "Genetic algorithm for balancing reconfigurable machining lines." *Computers Industrial Engineering* 66 (3): 541 – 547. Special Issue: The International Conferences on Computers and Industrial Engineering (ICCIEs) - series 41.

Borisovsky, Pavel A, Xavier Delorme, and Alexandre Dolgui. 2014. "Balancing reconfigurable machining lines via a set partitioning model." *International Journal of Production Research* 52 (13): 4026–4036.

Boysen, Nils, Malte Fliedner, and Armin Scholl. 2008. "Assembly line balancing: Which model to use when?" *International Journal of Production Economics* 111 (2): 509–528.

Bukchin, Joseph, and Jacob Rubinovitz. 2003. "A weighted approach for assembly line design with station paralleling and equipment selection." *IIE transactions* 35 (1): 73–85.

Buxey, G. M. 1974. "Assembly Line Balancing with Multiple Stations." *Manage. Sci.* 20 (6): 1010–1021.

Chechik, Shiri, Haim Kaplan, Mikkel Thorup, Or Zamir, and Uri Zwick. 2016. "Bottleneck paths and trees and deterministic graphical games." In *33rd Symposium on Theoretical Aspects of Computer Science (STACS 2016)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

Çil, Zeynel Abidin, Süleyman Mete, and Kürşad Ağpak. 2016. "A goal programming approach for robotic assembly line balancing problem." *IFAC-PapersOnLine* 49 (12): 938–942.

Cohen, Yuval, and M Ezey Dar-El. 1998. "Optimizing the number of stations in assembly lines under learning for limited production." *Production Planning & Control* 9 (3): 230–240.

Delorme, Xavier, Sergey Malyutin, and Alexandre Dolgui. 2016. "A multi-objective approach for design of reconfigurable transfer lines." *IFAC-PapersOnLine* 49 (12): 509 – 514. 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM 2016.

Dolgui, A., N. Guschinsky, and G. Levin. 2000. "Approaches to balancing of transfer lines with blocks of parallel operations." *Prepr./Inst. of eng. cybernetics of the Nat. acad. of sciences of Belarus* (8): 42.

Dolgui, Alexandre, and Evgeny Gafarov. 2019. "Can a Branch and Bound algorithm solve all instances of SALBP-1 efficiently?" *IFAC-PapersOnLine* 52 (13): 2788–2791.

Dolgui, Alexandre, and Sergey Kovalev. 2012. "Scenario based robust line balancing: Computational complexity." *Discrete Applied Mathematics* 160 (13-14): 1955–1963.

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

Essafi, Mohamed. 2010. "Conception et optimisation d'allocation de ressources dans les lignes d'usinage reconfigurables." PhD diss.

Essafi, Mohamed, Xavier Delorme, and Alexandre Dolgui. 2010a. "Balancing lines with CNC machines: A multi-start ant based heuristic." *CIRP Journal of Manufacturing Science and Technology* 2 (3): 176–182.

Essafi, Mohamed, Xavier Delorme, and Alexandre Dolgui. 2010b. "Balancing machining lines: a two-phase heuristic." *Studies in Informatics and Control* 19 (3): 243–252.

Essafi, Mohamed, Xavier Delorme, and Alexandre Dolgui. 2012. "A reactive GRASP and Path Relinking for balancing reconfigurable transfer lines." *International Journal of Production Research* 50 (18): 5213–5238.

Essafi, Mohamed, Xavier Delorme, Alexandre Dolgui, and Olga Guschinskaya. 2010. "A MIP approach for balancing transfer line with complex industrial constraints." *Computers Industrial Engineering* 58 (3): 393 – 400.

Gao, Jie, Linyan Sun, Lihua Wang, and Mitsuo Gen. 2009. "An efficient approach for type II robotic assembly line balancing problems." *Computers & Industrial Engineering* 56 (3): 1065–1080.

Gökçen, Hadi, Kürşad Ağpak, and Recep Benzer. 2006. "Balancing of parallel assembly lines." *International Journal of Production Economics* 103 (2): 600–609.

Held, Michael, and Richard M Karp. 1962. "A dynamic programming approach to sequencing problems." *Journal of the Society for Industrial and Applied Mathematics* 10 (1): 196–210.

Janardhanan, Mukund Nilakantan, Zixiang Li, Grzegorz Bocewicz, Zbigniew Banaszak, and Peter Nielsen. 2019. "Metaheuristic algorithms for balancing robotic assembly lines with sequence-dependent robot setup times." *Applied Mathematical Modelling* 65: 256–270.

Kara, Yakup, Cemal Özgüven, Neşe Yalçın Seçme, and Ching-Ter Chang. 2011. "Multi-objective approaches to balance mixed-model assembly lines for model mixes having precedence conflicts and duplicable common tasks." *The International Journal of Advanced Manufacturing Technology* 52 (5-8): 725–737.

Koren, Yoram, Uwe Heisel, Francesco Jovane, Toshimichi Moriwaki, Gumter Pritschow, Galip Ulsoy, and Hendrik Van Brussel. 1999. "Reconfigurable manufacturing systems." *CIRP annals* 48 (2): 527–540.

Koren, Yoram, and Moshe Shpitalni. 2010. "Design of reconfigurable manufacturing systems." *Journal of Manufacturing Systems* 29 (4): 130 – 141.

Koren, Yoram, Wencai Wang, and Xi Gu. 2017. "Value creation through design for scalability of reconfigurable manufacturing systems." *International Journal of Production Research* 55 (5): 1227–1242.

Lahrichi, Y, N Grangeon, L Deroussi, and S Norre. 2020a. "A new split-based hybrid metaheuristic for the reconfigurable transfer line balancing problem." *International Journal of Production Research* 1–18, DOI = 10.1080/00207543.2020.1720929.

Lahrichi, Youssef, Laurent Deroussi, Nathalie Grangeon, and Sylvie Norre. 2020b. "A Balance-First Sequence-Last Algorithm to design RMS. A Matheuristic with performance guaranty to balance Reconfigurable Manufacturing Systems." *Journal of Heuristics. Special issue on Matheuristics (Accepted on May 2020. To appear)* .

Lahrichi, Youssef, Laurent Deroussi, Nathalie Grangeon, and Sylvie Norre. 2020c. "A min-max path approach for balancing robotic assembly lines with sequence-dependent setup times." In *International Conference on Modeling, Optimization and Simulation*, 10.

Levitin, Gregory, Jacob Rubinovitz, and Boris Shnits. 2006. "A genetic algorithm for robotic assembly line balancing." *European Journal of Operational Research* 168 (3): 811–825.

Lourenço, Helena R, Olivier C Martin, and Thomas Stützle. 2010. "Iterated local search: Framework and applications." In *Handbook of metaheuristics*, 363–397. Springer.

Martino, Luigi, and Rafael Pastor. 2010. "Heuristic procedures for solving the general assembly line balancing problem with setups." *International Journal of Production Research* 48 (6): 1787–1804.

Merengo, C, F Nava, and Alessandro Pozzetti. 1999. "Balancing and sequencing manual mixed-model assembly lines." *International Journal of Production Research* 37 (12): 2835–2860.

Miltenburg, GJ, and Jacob Wijngaard. 1994. "The U-line line balancing problem." *Management science* 40 (10): 1378–1388.

Miralles, Cristóbal, José P García-Sabater, Carlos Andrés, and Manuel Cardós. 2008. "Branch and bound procedures for solving the assembly line worker assignment and balancing problem: Application to sheltered work centres for disabled." *Discrete Applied Mathematics* 156 (3): 352–367.

Miralles, Cristobal, Jose Pedro Garcia-Sabater, Carlos Andres, and Manuel Cardos. 2007. "Advantages of assembly lines in sheltered work centres for disabled. A case study." *International Journal of Production Economics* 110 (1-2): 187–197.

Moreira, Mayron César O, Marcus Ritt, Alysson M Costa, and Antonio A Chaves. 2012. "Simple heuristics for the assembly line worker assignment and balancing problem." *Journal of heuristics* 18 (3): 505–524.

Morrison, David R, Sheldon H Jacobson, Jason J Sauppe, and Edward C Sewell. 2016. "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning." *Discrete Optimization* 19: 79–102.

Morrison, David R, Edward C Sewell, and Sheldon H Jacobson. 2014. "An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset." *European Journal of Operational Research* 236 (2): 403–409.

Nilakantan, J Mukund, Sivalinga Govinda Ponnambalam, N Jawahar, and Ganesan Kanagaraj. 2015. "Bio-inspired search algorithms to solve robotic assembly line balancing problems." *Neural Computing and Applications* 26 (6): 1379–1393.

Özcan, Uğur. 2019. "Balancing and scheduling tasks in parallel assembly lines with sequence-dependent setup times." *International Journal of Production Economics* 213: 81–96.

Prins, Christian. 2004. "A simple and effective evolutionary algorithm for the vehicle routing problem." *Computers & operations research* 31 (12): 1985–2002.

Prins, Christian, Philippe Lacomme, and Caroline Prodhon. 2014. "Order-first split-second methods for vehicle routing problems: A review." *Transportation Research Part C: Emerging Technologies* 40: 179–200.

Rubinovitz, Jacob, Joseph Bukchin, and Ehud Lenz. 1993. "RALB–A heuristic algorithm for design and balancing of robotic assembly lines." *CIRP annals* 42 (1): 497–500.

Şahin, Murat, and Talip Kellegöz. 2017. "An efficient grouping genetic algorithm for U-shaped assembly line balancing problems with maximizing production rate." *Memetic Computing* 9 (3): 213–229.

Salii, Yaroslav. 2019. "Revisiting dynamic programming for precedence-constrained traveling salesman problem and its time-dependent generalization." *European Journal of Operational Research* 272 (1): 32–42.

Salveson, Melvin E. 1955. "The assembly line balancing problem." *The Journal of Industrial Engineering* 18–25.

Scholl, Armin, Nils Boysen, and Malte Fliedner. 2013. "The assembly line balancing and scheduling problem with sequence-dependent setup times: problem extension, model formulation and efficient heuristics." *OR spectrum* 35 (1): 291–320.

Scholl, Armin, and Robert Klein. 1997. "SALOME: A bidirectional branch-and-bound procedure for assembly line balancing." *INFORMS journal on Computing* 9 (4): 319–334.

Scholl, Armin, and Robert Klein. 1999. "Balancing assembly lines effectively–A computational comparison." *European Journal of Operational Research* 114 (1): 50–58.

Sewell, Edward C, and Sheldon H Jacobson. 2012. "A branch, bound, and remember algorithm for the simple assembly line balancing problem." *INFORMS Journal on Computing* 24 (3): 433–442.

Sotskov, Yuri N, Alexandre Dolgui, and Marie-Claude Portmann. 2006. "Stability analysis of an optimal balance for an assembly line with fixed cycle time." *European Journal of Operational Research* 168 (3): 783–797.

Stützle, Thomas. 2006. "Iterated local search for the quadratic assignment problem." *European Journal of Operational Research* 174 (3): 1519–1539.

Tiacci, Lorenzo. 2017. "Mixed-model U-shaped assembly lines: Balancing and comparing with straight lines with buffers and parallel workstations." *Journal of Manufacturing Systems* 45: 286–305.

Toksarı, M Duran, Selçuk K İşleyen, Ertan Güner, and Ömer Faruk Baykoç. 2010. "Assembly line balancing problem with deterioration tasks and learning effect." *Expert Systems with Applications* 37 (2): 1223–1228.

Van Laarhoven, Peter JM, and Emile HL Aarts. 1987. "Simulated annealing." In *Simulated annealing: Theory and applications*, 7–15. Springer.

van Zante-de Fokkert, Jannet I, and Ton G de Kok. 1997. "The mixed and multi model line balancing problem: a comparison." *European Journal of Operational Research* 100 (3): 399–412.

Vecchi, Thelma PB, Douglas F Surco, Ademir A Constantino, Maria TA Steiner, Luiz MM Jorge, Mauro ASS Ravagnani, and Paulo R Paraíso. 2016. "A sequential approach for the optimization of truck routes for solid waste collection." *Process Safety and Environmental Protection* 102: 238–250.

Vila, Mariona, and Jordi Pereira. 2014. "A branch-and-bound algorithm for assembly line worker assignment and balancing problems." *Computers & Operations Research* 44: 105–114.

Vilarinho, Pedro M, and Ana Sofia Simaria. 2002. "A two-stage heuristic method for balancing mixed-model assembly lines with parallel workstations." *International Journal of Production Research* 40 (6): 1405–1420.

Vilarinho, Pedro M, and Ana Sofia Simaria. 2006. "ANTBAL: an ant colony optimization algorithm for balancing mixed-model assembly lines with parallel workstations." *International Journal of Production Research* 44 (2): 291–303.

Wang, Wencai, and Yoram Koren. 2012. "Scalability planning for reconfigurable manufacturing systems." *Journal of manufacturing systems* 31 (2): 83–91.

Wee, TS, and Michael J Magazine. 1982. "Assembly line balancing as generalized bin packing." *Operations Research Letters* 1 (2): 56–58.

Yoosefelahi, A, M Aminnayeri, H Mosadegh, and H Davari Ardakani. 2012. "Type II robotic assembly line balancing problem: An evolution strategies algorithm for a multi-objective model." *Journal of Manufacturing Systems* 31 (2): 139–151.