الجمهوريــة الجزائريــة الديمقراطيــة الشعبيــة

République Algérienne Démocratique et Populaire

وزارة التعليـيـم العـالي و البـحث العـلمـي

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université MUSTAPHA Stambouli
Mascara
Faculté des Sciences Exactes
Département d'Informatique

جامعة مصطفى أسطمبولي
معسكر

THESE de DOCTORAT de 3ème cycle

Spécialité : Informatique

Option : Technologie de l'information et de la communication

Intitulée

The Resource Optimization Problem of Complex Assembly Lines

Présenté par : BELKHARROUBI Lakhdar

Le 02/05/2024 - 17:00h

A la faculté des sciences exact université de Mascara

Devant le jury :

Président	FEKIR Youcef	MCA	Université de Mascara
Examinateur 1	ABOU EL HASSAN Benyamina	Pr	Université Ahmed Benbella/Oran 1
Examinateur 2	BENHAOUA Kamel	Pr	Université de Mascara
Examinateur 3	ZAGANE Mohammed	MCA	Université de Mascara
Directeur de la thèse	YAHYAOUI Khadidja	Pr	Université de Mascara

Année universitaire : 2023/2024

Thanks

I would like to express my deepest gratitude to Allah for granting me strength, guidance, and perseverance throughout my academic journey. Additionally, I extend my heartfelt appreciation to my supervisor, YAHYAOUI Khadidja, whose unwavering support, invaluable insights, and encouragement have been instrumental in the completion of my research. I am also immensely thankful to the esteemed jury members, including President FEKIR Youcef, and the examiners BENHAOUA Kamel, ZAGANE Mohammed, and ABOU EL HASSAN Benyamina, for their time, expertise, and constructive feedback, which have significantly enriched my work. Finally, I am indebted to all my beloved family members, especially my parents, whose love, sacrifices, and encouragement have been the cornerstone of my success. Their unwavering support and belief in my abilities have been my greatest source of inspiration.

List of Publications

JOURNALS:

[1] **Lakhdar Belkharroubi**, K. Yahyaoui, "A Hybrid Grasp-genetic Algorithm for Mixed-model Assembly Line Balancing Problem Type 2", *International Journal of Computing*, Sep 2021.

[2] **Lakhdar Belkharroubi**, K. Yahyaoui, "Solving the mixed-model assembly line balancing problem type-I using a Hybrid Reactive GRASP", *Production Manufacturing Research*, Apr 2022.

[3] **Lakhdar Belkharroubi**, K. Yahyaoui, "Solving the energy-efficient Robotic Mixed-Model Assembly Line balancing problem using a Memory-Based Cuckoo Search Algorithm", *Engineering Applications of Artificial Intelligence*, July 2022.

CONFERENCES:

[1] **Lakhdar Belkharroubi**, K. Yahyaoui, "Maximization of the assembly line efficiency using an approach based on Genetic Algorithm", in *2022 2nd International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)*, March 2022, Meknes, Morocco.

[2] **Lakhdar Belkharroubi**, K. Yahyaoui, "A Hybrid approach for the Mixed-Model Assembly Line Balancing problem Type-II", in *2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, Sep 2021, Cracow, Poland.

Contents

List of Figures

List of Tables

List of Algorithms

Abstract of thesis entitled

The resource optimization problem for complex assembly lines

Submitted by **Lakhdar BELKHARROUBI**

for the degree of PhD in computer science

at University of Mustapha Stambouli Mascara

This thesis delves into the intricacies of resource optimization, with a specific focus on the Assembly Line Balancing Problem (ALBP) in the context of complex assembly lines. In a world where advanced technologies have permeated various sectors, including industry, medicine, and the military, the efficient management of assembly lines becomes paramount. The fundamental objective of this research is to enhance production processes by addressing the complexities of assembly line balancing. This entails optimizing critical factors such as cycle time, the number of workstations, and resource allocation. The ALBP, central to this study, encompasses various forms, including the Mixed-Model Assembly Line Balancing Problem (MiMALBP) and the Robotic Assembly Line Balancing Problem (RALBP), each presenting unique complexities. Moreover, as the industry diversifies and production requirements evolve, the ALBP encounters novel characteristics and constraints, underscoring the need for innovative solutions. Drawing on meta-heuristic approaches powered by Artificial Intelligence, this thesis explores the efficient resolution of ALBP, as well as its variants. The research also tackles the challenge of energy optimization in assembly lines designed for the mixed production of multiple product models. Ultimately, this thesis navigates the ever-evolving industrial landscape, offering insights into addressing the complexities of assembly line balancing while harnessing the potential of advanced technologies and meta-heuristic methods to optimize resources effectively.

Keywords: Meta-heuristics, Heuristics, Assembly lines, Artificial intelligence, Optimization, Bio-inspired Algorithms.

مُلَخّــص

تتعمق هَذِه الأُطْرُوحَة فِي تعقيدات تَحْسِين الْمَوَارِد ، مَع التركيز بِشَكْل خَاص عَلَى مُشْكِلَةٌ مُوَازَنَة خَطّ التَّجْمِيع (ALBP) فِي سِيَاقِ خُطُوط التَّجْمِيع المعقدة. فِي عَالَمٍ حَيْث تَغَلْغَلَت التقنيات الْمُتَقَدِّمَةِ فِي مُخْتَلَف القطاعات، بِمَا فِي ذَلِكَ الصِّنَاعَة وَالطِّبّ وَالْجَيْش، أَصْبَحْت الْإِدَارَة الْفَعَالَة لخطوط التَّجْمِيع ذَات أَهَمِّيَّةً قُصْوَى. الهَدَف الأَسَاسِي مِنْ هَذَا الْبَحْث هُوَ تَعْزِيز عَمَلِيَّات الإِنْتاج مِنْ خِلَالِ مُعَالَجَة تعقيدات مُوَازَنَة خَطّ التَّجْمِيع. وَيَسْتَلْزِم ذَلِكَ تَحْسِين الْعَوَامِل الحاسِمَة مِثْلِ وَقْتِ الدَّوْرَة وَعَدَد محطات الْعَمَل وَتَخْصِيص الْمَوَارِد. يَشْمَل ALBP وَهُو مِحْوَر هَذِهِ الدِّرَاسَة، أشكالًا مُخْتَلِفَة، بِمَا فِي ذَلِكَ مُشْكِلَةٌ مُوَازَنَة خَطّ التَّجْمِيع للنموذج الْمُخْتَلِط (MiMALBP) ومشكلة مُوَازَنَة خَطّ التَّجْمِيع الآلِيّ (RALBP) حَيْث يُمَثِّل كُل مِنْهُمَا تعقيدات فَرِيدَة مِنْ نَوْعِهَا. عِلَاوَةً عَلَى ذَلِكَ، وَمَع تَنَوُّع الصِّنَاعَة وتطور متطلبات الإِنْتاج ، يُوَاجِه ALBP خَصَائِص وقيودًا جَدِيدَة، مِمَّا يُؤَكِّدُ الْحَاجَةِ إِلَى حُلُولِ مبتكرة. بِالِاعْتِمَادِ عَلَى الْمَنَاهِجِ الاستدلالية الْفَوْقِيَّة المدعومة بِالذَّكاء الاِصْطِناعِي، تستكشف هَذِه الأُطْرُوحَة الْحِلّ الْفِعَال لـ ALBP بِالْإِضَافَةِ إِلَى متغيراته. يَتَنَاوَل الْبَحْث أيضًا التَّحَدِّي المتمثل فِي تَحْسِينِ الطَّاقَةِ فِي خُطُوط التَّجْمِيع الْمُصَمِّمَة لِلْإِنْتَاج المختلط لنماذج الْمُنْتَجَات الْمُتَعَدِّدَة. فِي نِهَايَةِ الْمَطَاف، تتنقل هَذِه الأُطْرُوحَة فِي الْمَشْهَدِ الصِّنَاعِيّ المتطور بِاسْتِمْرَار ،وَتَقَدَّم رَوَى حَوْل مُعَالَجَة تعقيدات مُوَازَنَة خَطّ التَّجْمِيع مَع تَسْخِير إِمْكَانَات التقنيات الْمُتَقَدِّمَة وَالْأَسَالِيب الْفَوْقِيَّة لِتَحْسِين الْمَوَارِد بِشَكْل فَعَّال.

كلمات مفتاحية : الاستدلال الفوقي، الاستدلال، خطوط التجميع، الذكاء الاصطناعي، التحسين، الخوارزميات المستوحاة من الحيوية.

Abstract

Cette thèse explore les subtilités de l'optimisation des ressources, avec un accent particulier sur le problème d'équilibrage des chaînes d'assemblage (ALBP) dans le contexte de chaînes d'assemblage complexes. Dans un monde où les technologies avancées ont imprégné divers secteurs, notamment l'industrie, la médecine et l'armée, la gestion efficace des chaînes d'assemblage devient primordiale. L'objectif fondamental de cette recherche est d'améliorer les processus de production en abordant les complexités de l'équilibrage des chaînes de montage. Cela implique d'optimiser des facteurs critiques tels que le temps de cycle, le nombre de postes de travail et l'allocation des ressources. L'ALBP, au cœur de cette étude, englobe diverses formes, notamment le problème d'équilibrage des lignes à modèles mixtes (MiMALBP) et le problème d'équilibrage des lignes robotisées (RALBP), chacun présentant des complexités uniques. De plus, à mesure que l'industrie se diversifie et que les exigences de production évoluent, l'ALBP rencontre de nouvelles caractéristiques et contraintes, soulignant la nécessité de solutions innovantes. S'appuyant sur des approches méta-heuristiques alimentées par l'intelligence artificielle, cette thèse explore la résolution efficace de l'ALBP, ainsi que ses variantes. La recherche aborde également le défi de l'optimisation énergétique dans les chaînes d'assemblage conçues pour la production mixte de plusieurs modèles de produits. En fin de compte, cette thèse explore un paysage industriel en constante évolution, offrant un aperçu de la manière de résoudre les complexités de l'équilibrage des chaînes d'assemblage tout en exploitant le potentiel des technologies avancées et des méthodes méta-heuristiques pour optimiser efficacement les ressources.

Mots Clés: Méta-heuristiques, Heuristiques, Chaînes d'assemblage, Intelligence artificielle, Optimisation, Algorithmes bio-inspirés.

Chapter 1

Introduction

In manufacturing environments, the production of a variety of products such as electronics, automobiles, and furniture is based on important systems known as assembly lines. In 1913, Henry Ford created the first assembly line in his automobile industry in America [1], and since that time, assembly lines are still integrated into different kinds of industries. An assembly line is a set of workstations, and at each workstation, a set of tasks are performed while assembling a product. A material handling system is used, such as a conveyor belt, in order to move a product from one workstation to another one. Assigned tasks can be performed by two types of operators: humans and robots. Human operators are used in order to complete some tasks that are not complex and do not require high precision, whereas robots are used in order to perform a set of repetitive tasks that require high precision. In some cases, humans work in collaboration with collaborative robots known as Cobots [2].

The assembly line balancing is based on the optimization of one or several resources, and this is the critical part of the installation of the line. Optimizing several resources at the same time, the problem becomes more complex and hard to solve. The problem related to the optimization of the assembly line is known in the literature as the assembly line balancing problem (ALBP). The ALBP is divided into several types, and in each type, there is more than one version. Each version depends on the characteristics of the assembly line, including the line layout and the resources to be optimized, such as the cycle time, the number of workstations, the number of operators, the energy consumed, etc [3]. The first publication of its mathematical formulation was in 1955 by Salveson [4], and since that time, several works have been done to solve different ALBP versions using two types of methods: exact methods and

approximation methods including heuristics and metaheuristics. The ALB is one of the NP-hard problems, which cannot be solved in polynomial time [5]. Thus exact methods can only solve small instances. As a solution to this complicated situation, researchers thought to use approximation methods in order to find good solutions for optimization problems in a reasonable time.

Researchers have classified the ALBP into two categories: simple ALB problems (SALBPs) and general ALB problems (GALBPs) [6]. This classification is based on several factors, including the assembly line layout, the resources to be optimized while designing the line, and the number of models planned for production. For instance, the SALBPs category includes all ALB problems related to simple lines that assemble only one model in a straight-paced line, whereas the GALBPs category includes all problems related to assembly lines having a set of characteristics that make them complex. Several heuristics have been proposed in order to solve many SALB problems, such as the ranked positional weight (RPW), the longest task time (LTT), the shortest task time (STT), etc [7]. The problem is that, in most cases, a simple heuristic cannot be used to solve recent complicated ALBP versions that generally belong to the GALBPs category. Therefore, several metaheuristics, such as the Genetic Algorithm (GA), the Greedy Randomized Adaptive Search Procedure (GRASP), the Ant Colony Optimization Algorithm (ACO), and the Cuckoo Search Algorithm (CSA), have been proposed to solve many GALB problems. As technology advances, new ALBP versions emerge with greater complexity than prior versions, necessitating the development of new methods to solve them.

1.1 Motivation

Nowadays, new advanced technologies are still integrated into several sectors, such as industry, medicine, and the military, which has created several complex real-world optimization problems that are harder to solve. On the other hand, advances in computing and artificial intelligence allow researchers to create new methods and techniques that can be used to find solutions to optimization problems. With the appearance of the industry 4.0 concept, a variety

of technologies have been integrated, such as the Internet of Things (IoT), Artificial Intelligence, the Cyber-Physical System (CPS), Augmented Reality (AR), 3D printing, etc [8]. All these technologies have been integrated in order to achieve better production and monitoring.

The optimization of the assembly line is an important point when talking about good production since it covers several resources, such as the cycle time, by which we can determine the production rate in a specific period. Minimizing the number of workstations in the assembly line can also help manufacturers achieve a good production process by minimizing the total idle time and minimizing the total space occupied by the assembly line. Each ALB problem that has been solved aims to optimize one or multiple resources at the same time, such as in the case of the ALBP type E, which aims to optimize both the cycle time and the number of workstations together [9]. Other new ALB problems have occurred in the industry, having new characteristics, such as the mixed-model assembly line balancing problem (MiMALBP), the multi-model assembly line balancing problem (MuMALBP), the robotic assembly line balancing problem (RALBP), the U-shaped assembly line balancing problem (UALBP) and so on.

The mixed model assembly line balancing problem is more complex than the simple one since it takes into consideration several models and constraints. There exist several MiMALBP types, such as MiMALBP type 1, which aims to minimize the number of workstations, and MiMALBP type 2, which aims to minimize the cycle time [10]. The MiMALBP has occurred in the industry due to the variation of customers' demands for more than one model of the same product. Several works have been done aiming at solving the MiMALB problems but there are some types that have not been solved due to the variation of resources and characteristics. The complexity of the MiMALBP is higher in comparison with the SALBP, and it increases when there is more than one resource to be optimized in the line. In addition, the existence of several constraints in the problem makes it difficult to solve. The number of models to be assembled in the line also influences the complexity of the problem, since each model has its own properties and tasks.

The integration of robots into the assembly lines in order to accomplish some complicated repetitive tasks has created a new problem known as the

robotic assembly line balancing problem (RALBP). The hard part of the RALBP is that we need to find not only the best assignment of tasks but also the best assignment of robots. There are several RALBP types. For instance, RALBP type 1 aims to minimize the number of workstations, RALBP type 2 aims to minimize the cycle time; RALBP type cost, and RALBP type O [11]. In a robotic assembly line, energy is an important resource, and we can find only a few studies that have been done in order to minimize the total energy consumed in the line by choosing the best set of robots that will be assigned to workstations. Furthermore, all published works that aim to minimize energy consumption focus only on those lines that are designed for the mass production of one model.

Real-world optimization problems still appear in the industry sector, and their complexity is increasing in comparison with those problems that have already been solved. AI approaches such as meta-heuristics are the most commonly employed for solving these problems due to their efficiency, especially when dealing with large-scale problems but the big challenge is to find the appropriate meta-heuristic that can find good solutions for a specific optimization problem. In the literature, we can find many several works that have used meta-heuristics to solve ALBP problems including SALPB, MiMALBP, RALBP, and other versions. Some ALB problems have not been taken into consideration for many reasons, such as the variation of the line characteristics, the nature of resources, the production type, the nature of operators, the nature of tasks, etc.

1.2 Contributions

The aim of this thesis is to study and solve some optimization problems that occur in the assembly line and have an influence on some important resources, such as the cycle time and energy consumed during production. In addition, we address some assembly line balancing problems that have not been well studied despite their importance in finding a good assembly line design in order to achieve better production.

As contributions, we have published six scientific researches including three articles published in journals indexed in some bases such as Scopus, two conference papers with proceedings published in IEEE Xplore, and a chapter included in a book published in WILLY library. Our contributions are detailed as follows:

In the first contribution [12], the energy efficient robotic mixed-model assembly line balancing problem is addressed. The main goal of solving this problem is to minimize energy consumption in robotic assembly line where each robots has its own properties. To solve this complicated problem, we have proposed a memory based cuckoo search algorithm (MBCSA). The performance of the proposed MBCSA is tested on six different problems, and the obtained results are compared with those obtained by two other meta-heuristics.

In the second contribution [13], we have addressed the mixed-model assembly line balancing problem type 1 that aims to minimize the number of workstations in the line. To solve this problem, we have proposed a hybrid reactive greedy randomized adaptive search procedure (HRGRASP). The HRGRASP is based on four principal steps, the construction phase, the local search phase, the evaluation of obtained solutions, and the update of selection probabilities. In the construction phase, we have used the shortest processing time heuristic in order to construct a solution. Obtained solutions are replaced by their best neighbors in the local search phase by applying a simple neighborhood procedure. The proposed HRGRASP is tested on seven problems, and its performance is compared with other methods.

In the third contribution [14], we have proposed a hybridization between two meta-heuristics in order to solve the mixed-model assembly line balancing

problem type 2 that aims to optimize the cycle time of the line. The two meta-heuristics are the famous genetic algorithm (GA) and the greedy randomized adaptive search procedure (GRASP). The aim of using the GRASP in this study is to create a set of feasible solutions for the addressed problem. The created set is used as the first population in the genetic algorithm. In order to create a solution in the GRASP, we used ranked positional weight (RPW) in its construction phase, and to ameliorate the solution, a neighborhood procedure is used in the local search phase. The proposed hybridization is tested on a numerical example.

In the fourth contribution [15], we have addressed the simple assembly line balancing problem type E that aims to optimize both the cycle time and the number of workstations at the same time. As a solution to this problem, we have proposed an approach that is based on the genetic algorithm (GA). We have generated three different problems in order to evaluate our approach, and each problem was solved several times according to the problem data. The obtained results are compared with those obtained by the reactive greedy randomized adaptive search procedure.

In the fifth contribution [16], we have proposed a hybridization between the ranked positional weight (RPW) heuristic and the reactive greedy randomized adaptive search procedure (RGRASP) in order to solve the mixed-model assembly line balancing problem type 2. The RPW is used in the construction phase of the RGRASP to build feasible solutions. In the local search phase of the RGRASP, we have used a local search method that aims to find better neighbor solutions. Three problems of different sizes are generated to test the proposed algorithm.

1.3 Thesis Outline

This chapter begins with an introduction, followed by our motivation and a discussion of our contributions. The remainder of this thesis includes seven chapters as follows:

The state of the art (as chapter 2) in which the background of this thesis is provided and is organized as follows; we first discuss the assembly lines,

different types. Secondly, we define the assembly line balancing problem and give its general mathematical formulation. Then, we provide the classification of the ALB problems. Next, we present the optimization methods used by researchers to solve the ALB problem. Finally, we provide the state of the art.

In chapter 3, we introduce the first contribution, which is related to the energy efficient robotic mixed model assembly line balancing problem. This chapter is organized as follows; we first start with an introduction, then we define the energy efficient RMiMALBP problem, and next we explain our memory-based cuckoo search algorithm used to solve this problem. Then we provide our results and the related discussion before concluding the chapter.

Chapter 4 presents the second contribution in which we have solved the mixed model assembly line balancing problem using the hybrid reactive GRASP. This chapter starts with an introduction. Secondly, we provide the definition of the problem and its mathematical formulation. The proposed hybrid reactive GRASP is then presented, and the results are discussed. Finally, we conclude the chapter.

In Chapter 5, we introduce our third contribution, in which we proposed a hybridization between the GRASP and the GA in order to solve type 2 of the mixed model assembly line balancing problem. First, an introduction to the chapter is provided, then the mixed model assembly line balancing problem is defined. Next, we introduce the proposed hybrid GRASP genetic algorithm. After that, we discuss the results and we conclude the chapter.

Chapter 6 presents the fourth contribution, in which we have solved the mixed model assembly line balancing problem type 2 using a hybridization between the reactive GRASP and the RPW. This chapter starts an introduction, then we define the problem with its assumptions and mathematical formulation. Next, we explain our approach used to solve the problem. Then we provide the results of the study, and we finish the chapter with a conclusion.

Chapter 7 provides the fifth contribution in which we have solved the simple assembly line balancing problem type E using a genetic algorithm based approach. It is organized as follows; first we begin with an introduction, then we present the SALBP type E and its mathematical formulation. Then, we explain our genetic algorithm-based approach. Next, the results are provided

with the corresponding discussion. Finally, we conclude the chapter.

The last chapter concludes the thesis and all the work carried. It recapitulates the challenges, contributions, and summaries our findings. It also presents future perspectives that can be followedup by this thesis.

Chapter 2

Assembly Line Balancing Problem: State of the Art

2.1 Introduction

The aim of the assembly process is to bring together two more work-pieces in order to create a specific product. The work-pieces are added successively in each workstation until a final product is completed. Historically, the assembly lines were introduced by Henry Ford in order to achieve good management in his automobile industry [17]. The first assembly lines were simple and designed for mass production for only one model such as in the case of Henry company which was created to produce one car model. The assembly line is composed of a set of workstations and a material handling system, such as the conveyor belt. The material handling system links all workstations and moves the product from one workstation to another in order to complete the assembly process [18]. The rhythm of the material handling system differs from one assembly line to another. For instance, simple assembly lines are considered paced. In some assembly lines, workstations are decoupled by buffer stocks, which hold the item when the next station is still working on the previous item [19].

During the production process, a set of assembly tasks is performed in each workstation to complete one product, and each workstation has a workstation time equal to the sum of all task times of assigned tasks [20]. The cycle

time by which we can determine the period between two successive accomplished products (the production rate), and equal to the maximum workstation time in the assembly line [21]. The difference between the cycle time and the workstation time is known as the idle time. In other words, idle time is the period in which the workstation is inactive [22]. The nature of task times varies from one assembly line to another. For instance, in the case of the manual assembly line where human operators perform the tasks, the task times are considered stochastic due to several factors, including the skills and motivation of human operators [19].

The initial assembly lines were completely manual, which meant that only human operators were used to complete assembly tasks [23]. Over time, robots were introduced into assembly lines as a new sort of operator in order to achieve high automation [24]. The introduction of robots has created a new assembly line version known as the robotic assembly line. The benefit of using robots in assembly lines is that it reduces labor costs and variability in task times from manual work [25]. Unlike human labor, robots can be programmed to perform different assembly tasks without the worry of fatigue. Robotic assembly lines are more flexible and are characterized by both their speed of production and the quality of their products [26]. Collaborative robots, or cobots, may aid in the implementation of a dynamic productive cell capable of supporting multi-model production while being more adaptable to model and volume changes [27].

There are several assembly line versions, and each version depends on its characteristics such as the assembly line rhythm, the operators used in the line, the assembly line design (or shape), and the type of production. As discussed before, the rhythm of the assembly line can be paced or unpaced in the case of using buffers with workstations, and for the operators, there are two types; humans and robots, which can be used separately or together to perform assembly tasks. For the assembly line shapes, there are several ones, including the straight shape, the U shape, and the parallel shape. In straight assembly lines, workstations are placed in a serial manner and the product moves in a straight direction, whereas in U-shaped assembly lines, the flow of the product has the form of the letter "U" as shown in fig 2.1. In the case of parallel lines, two or more neighboring lines are located in parallel and can be

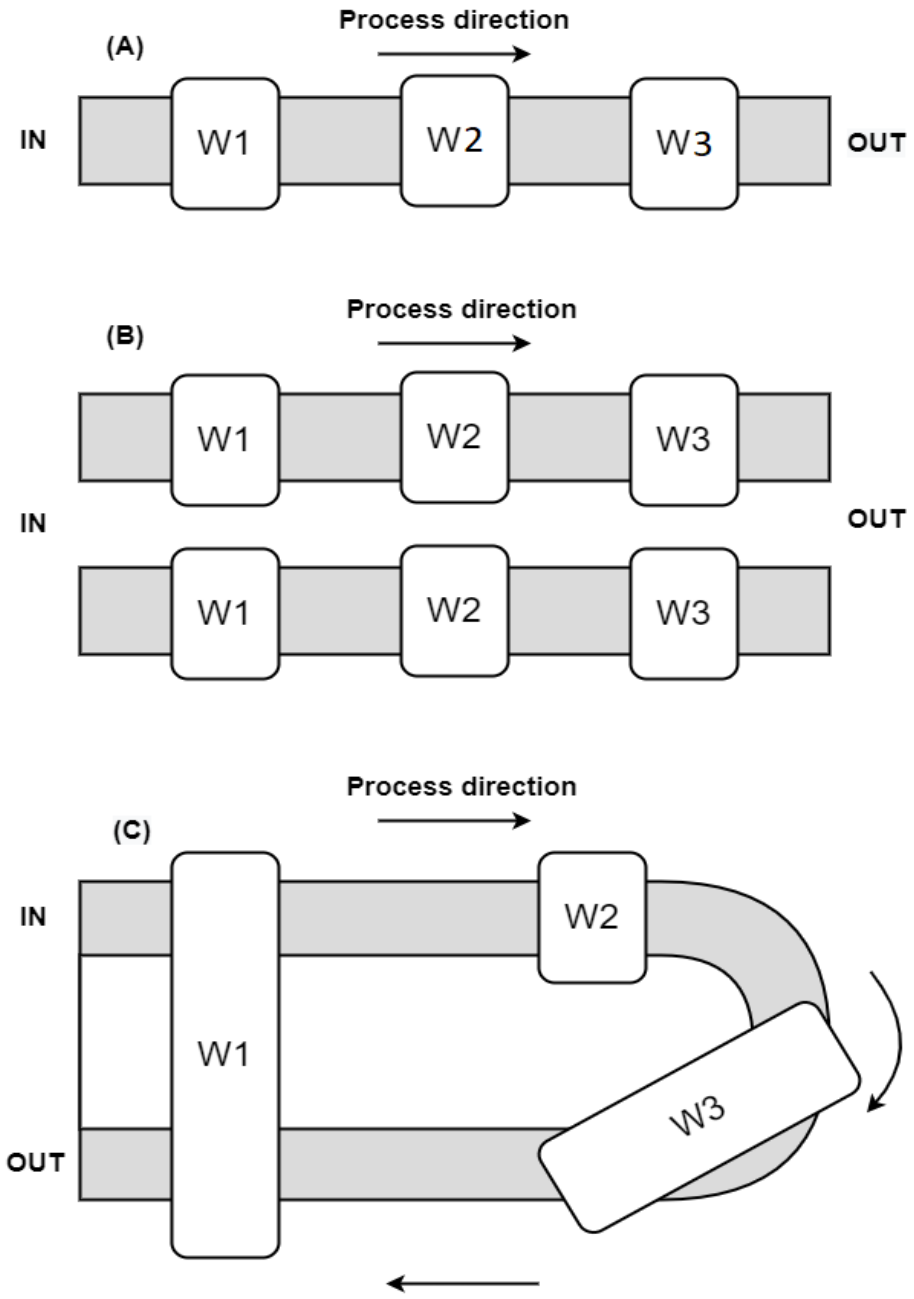

Figure 2.1: Line Layouts (A) straight line, (B) parallel line, (c) U-shaped Line.

balanced with common workstations that perform tasks for all lines or with independent workstations[28].

Regarding the type of production, there are three types of assembly lines; simple assembly lines(SAL), mixed-model assembly lines (MiMAL), and multi-model assembly lines (MuMAL). The SAL is the first version of the assembly

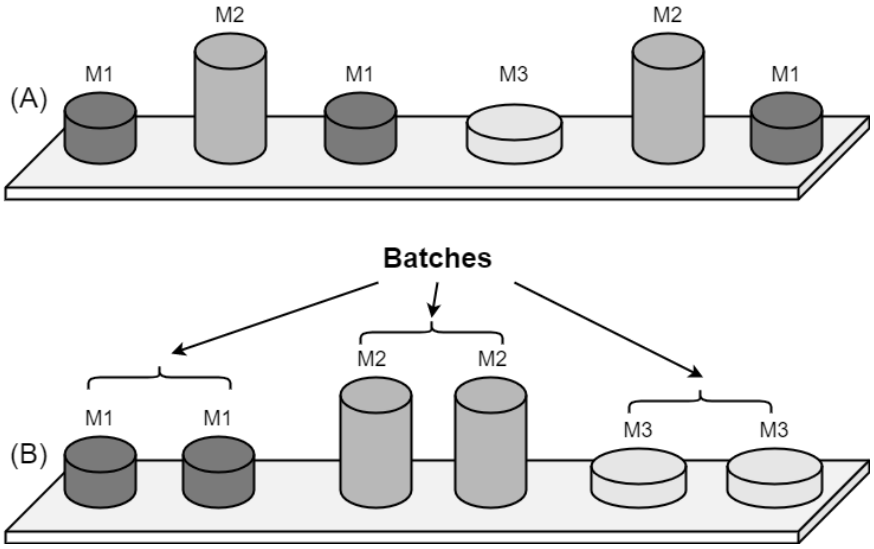

Figure 2.2: Mixed and Multi-Model assembly lines.

line. This type of line is used for the mass production of one model [29]. The MiMALs are more complex than the SALs, they can assemble more than one model of the same product in an intermixed sequence with one configuration [30]. The MuMALs are used to assemble more than one model of the same product in batches. As shown in fig 2.2, in MuMAL the number of models is more than one in each batch, unlike in MiMAL [31].

Customers nowadays look for products that have different product variants with different distinguishing qualities from other products on the market. Manufacturing companies face the challenge of upgrading their assembly lines, and the increased demand for a variety of different products with different characteristics is the main reason for the creation of flexible assembly lines such as mixed-model and multi-model assembly lines, which can meet market demands.

Designing an assembly line is difficult and complex since it takes into consideration not only the characteristics of the line but the resources to be optimized. In addition to the cycle time and the number of workstations, there are other resources that can influence the design of the assembly line, such as the number of operators (humans and robots) and the energy consumed during the assembly process. In addition to the basic constraints of the ALBP, some

other constraints can influence the complexity if they must be taken into consideration when solving the problem such as zoning constraints [32].

The assembly lines are so important and need good design and planning since they play a crucial role in the production of different kinds of products. The ALBP is the challenge of optimizing an assembly line by maximizing or minimizing numerous criteria measurements. In this chapter, the background of this thesis is provided.

1.2 Definition of the assembly line balancing problem

An assembly line consists of workstations *K=1,...,m* arranged along a mechanical handling system. The workpieces move from one workstation to another one in a unidirectional manner. At each workstation, a set of tasks *T=1,...,n* are repeatedly performed regarding the cycle time. The time needed to complete a task *j* is known as task time *tj*. Partitioning the available tasks among the workstations with respect to a set of predefined objectives is known as the Assembly Line Balancing Problem (ALBP) [5].

The assembly tasks are represented in a graph *G* known as the precedence graph which illustrates the precedence relations between all the tasks. The assignment of tasks to workstations is restricted by precedence relations and by other constraints, such as zoning constraints in some cases. If there is no zoning constraints, then any task can be performed at any workstation. The workload of workstation *k* is determined by the set of tasks *Sk* assigned to it.

The basic constraints of the assembly line balancing problem are as follows; All available assembly tasks must be assigned. An assembly task must be assigned to only one workstation. If task *i* precedes task *j*, then, task *i* must be assigned before task *j*. The sum of the task times for tasks in *Sk* is known as the workstation time. If the cycle time *C* is given, then, the workstation time *t(Sk)* must not exceed *C*. If *t(Sk)* < C, the workstation *K* has an idle time that is equal to c - *t(Sk)* time units in one cycle.

In the illustrative example fig 1.3, we have a precedence graph of one product. The number inside the circle is the task ID that is used in the problem, and the number outside the circle is the task time. To complete this product, 8 tasks must be completed, respecting the precedence relations. For example, we cannot perform task 7 before completing tasks 1, 2, 4, and 5. For this example, a feasible balance with $m = 4$ workstations and cycle time $c = 5$ is given by workstation loads *s1* = 1, 3, 4, *s2* = 2, 5, *s3* = 6, 7 and *s4* = 8 with an idle time = 2 in workstation *s4*. Another feasible balance with $m = 3$ workstations and cycle time $c = 6$ is given by *s1* = 1, 2, 3, *s2* = 4, 6, *s3* = 5, 7, 8 with no idle time.

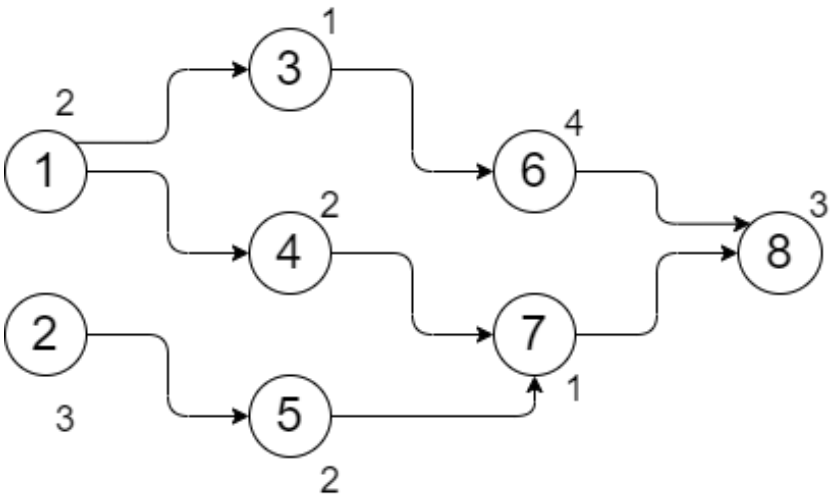

Figure 2.3: Precedence relations graph.

2.3 Classification of the assembly line balancing problem

The assembly line balancing problem is classified into two categories, the simple assembly line balancing problem (SALBP) and the general assembly line balancing problem (GALBP). The SALBP occurs in simple assembly lines and it has specific assumptions. If an ALB problem has different assumptions in comparison with the SALBP, then we can consider it like a general assembly line balancing problem [34]. The GALBP regroups several ALB problems that have appeared with the advancement in technologies and assembly lines. The classification of the ALBP is as follows.

2.3.1 Simple assembly line balancing problem (SALBP)

The simple assembly line balancing problem is the best-known and best-studied among the family of ALB problems. Although the complexity of SALPB is far from Regarding the complexity of real-world line balancing, it is considered as the core problem of the ALB. There are four SALBP problems, SALBP type 1 (SALBP-1), SALBP type 2 (SALBP-2), SALBP type E (SALBP-E), and SALBP type F (SALBP-F) [35, 36]. In the SALBP-1, the aim is to minimize the number of workstations for a fixed known cycle time, and in the SALBP-2 the aim is to minimize the cycle time for a fixed known number of workstations. The

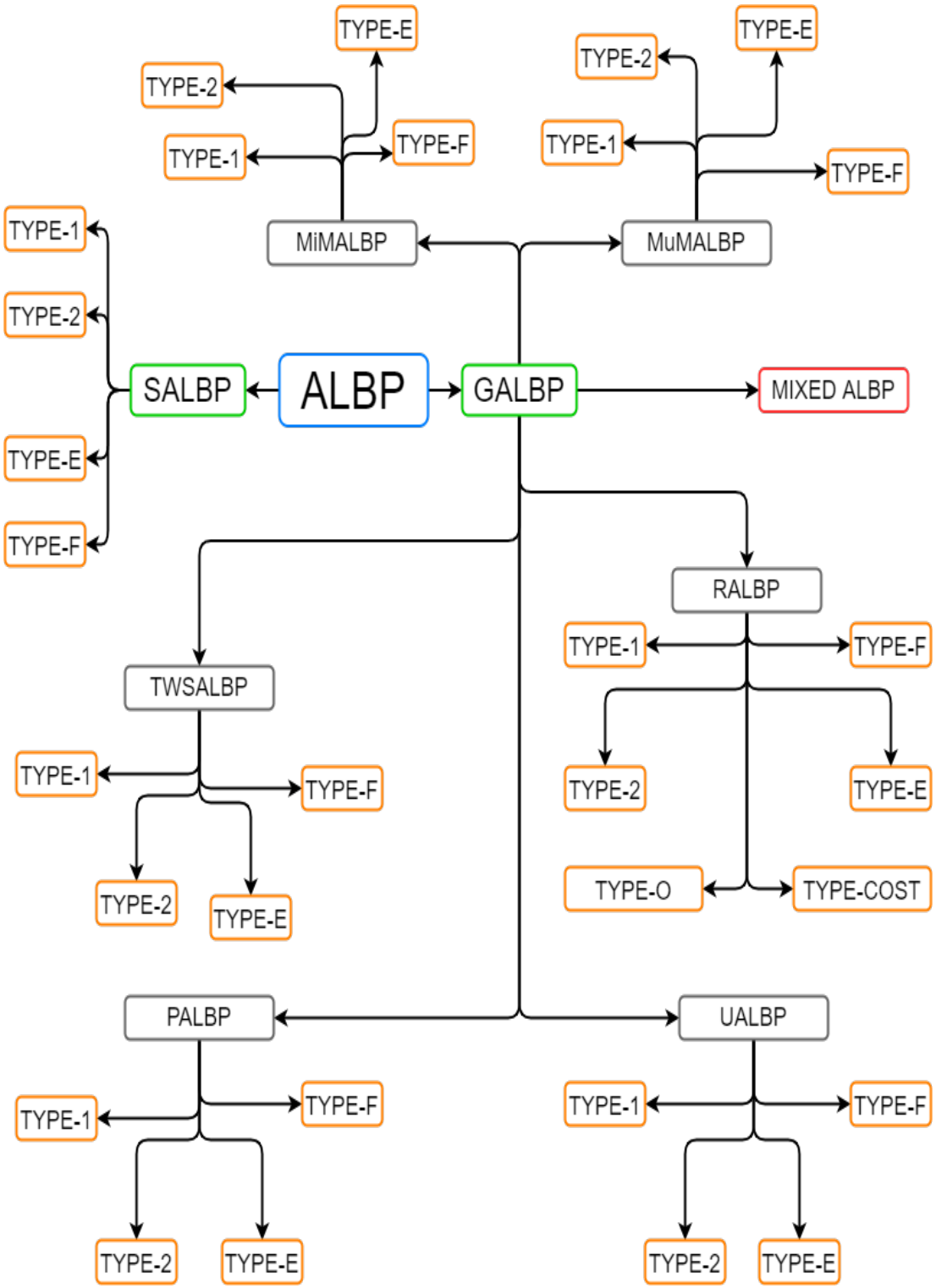

Figure 2.4: Assembly line balancing problem classification.

SALBP-E aims to optimize both the cycle time and the number of workstations at the same time [37]. In the SALBP-F both the cycle time and the number of workstations are known and the aim is to find a feasible assembly line [38, 39].

According to [18, 40, 41] the simple assembly line balancing problem has the following assumptions:

- An assembly task cannot be split among two or more workstations.
- The line is paced with a common fixed cycle time.
- The line is considered to be serial.
- No parallel elements or feeder lines.
- The sequence of tasks to be processed is restricted by precedence relations.
- No other constraints besides precedence relations.
- Task times are considered deterministic.
- Assembly tasks are processed in a predetermined mode.
- Mass production of only one product.
- All Workstations are equally equipped with workers and machines.

In the literature, we can find several works that have been done to solve a variety of simple assembly line balancing problems using exact methods and approximation methods. Baybars *et al.* [40] have developed an efficient single-pass heuristic method that can find good solutions for the deterministic SALBP. Pastor *et al.* [42] have proposed an improved mathematical program to solve both the SALBP type 1 and the SALBP type 2. The key idea in this work is based on the upper bound on the number of workstations for SALBP-1 and on the upper bound on the cycle time for SALBP-2. Sewell *et al.* [43] have proposed a branch, bound, and remember algorithm for the SALBP. The proposed algorithm combines a variety of methods to create an approach that is computationally superior to all exact algorithms that were reported in the literature.

Wei *et al.* [44] have proposed a solution procedure to solve the SALBP type E that combines the SALBP-1 and SALBP-2. Kilincci *et al.* [45] have developed a Petri net-based heuristic in order to solve SALBP type 2. They developed three versions of the heuristic by integrating forward, backward, and bidirectional procedures. They also implemented a binary search procedure in order to improve the solution. Blum *et al.* [46] have proposed a hybridization between two meta-heuristics the ant colony optimization (ACO) and the beam search to tackle the siSALBP type 1. Esmaeilbeigi *et al.* [47] have proposed a mixed integer linear programming formulation in order to solve the SALBP type E. Furthermore, two enhancement techniques in the form of valid inequalities and auxiliary variables are proposed to strengthen the presented formulation even further.

Kilincci *et al.* [48] have developed a different heuristic approach based on the P-invariants of Petri nets to solve the SALBP type 1. Zhang *et al.* [49] have proposed an improved immune algorithm for the simple assembly line balancing problem type-1. Zhang *et al.* [50] have proposed An integer-coded differential evolution algorithm for SALBP type 2. Pereira *et al.* [51] have developed empirical evaluation of lower bounding methods for the SALBP with the aim of minimizing the number of workstations in the line. Goncalves *et al.* [52] have proposed a hybrid genetic algorithm with a local search to tackle the SALBP type 1.

Gokcen *et al.* [53] have developed a goal-programming approach based on an integer programming formulation to solve the simple U-line balancing problem with the aim of minimizing the number of workstations. Dou *et al.* [54] have proposed a feasible task sequence-oriented discrete particle swarm algorithm to solve the SALBP type 1. Jirasirilerd *et al.* [55] have developed a variable neighborhood strategy adaptive search in order to solve SALBP type 2 in the garment industry. Saltzman *et al.* [56] have presented a two-process implicit enumeration algorithm for the SALBP with the objective of minimizing the largest station number to which a task is assigned. Sikora *et al.* [57] have proposed an integer-based formulation for the SALBP with multiple identical tasks with the aim of minimizing the cycle time.

Arik *et al.* [58] have proposed a mixed integer programming model to solve the SALBP type 1 with grey demand and grey task durations. Ravelo *et*

al. [59] have proposed in their work approximation algorithms to solve SALB problems including SALBP-1, SALBP-2, and SALBP-E. Emrani *et al.* [60] have used a mathematical model to solve SALBP type 1 under task deterioration. Baskar *et al.* [61] have proposed and analyzed a few heuristics based on slope indices that can be used to tackle the SALBP type 1. Toklu *et al.* [62] have proposed a fuzzy goal programming model for the simple U-line assembly line balancing problem with the aim of optimizing several conflicting goals. Lalaoui *et al.* [63] have developed an adaptive generalized simulated annealing using a fuzzy inference system for the SALBP type 1. Xu *et al.* [64] have used a tabu search algorithm for type 2 u-shaped simple assembly line balancing problem which considers workload smoothing as the secondary objective. In our study [15], we proposed a genetic algorithm-based approach to solve the SALBP type E.

2.3.2 General assembly line balancing problem (GALBP)

Real-world ALB problems have other and more complex characteristics in comparison with those of the SALBP. This difference has created new ALB problems that fall under the general assembly line balancing problems category. In GALBP, the assumptions of the SALBP are relaxed in order to correspond to the real environment [35]. Several GALB problems can be found in the literature such as the mixed-model ALBP, robotic ALBP, U-shaped ALBP, Two-sided ALBP, and parallel ALBP. As shown in fig 2.4, each GALBP has four types (type 1, type 2, type E, and type F), furthermore, there is another category named mixed ALBP that regroups those problems that are mixed in terms of line layout, number of models, number of sides and so on.

2.3.2.1 Mixed-model assembly line balancing problem (MiMALBP)

The mixed-model assembly line balancing problem (MiMALBP) is a critical problem that takes into consideration the tasks of all models when balancing the line. In order to decrease the complexity of the MiMALBP, it can be transformed into SALBP using two different methods; using the adjusted task processing times or combining all precedence graphs of all models in one graph called the combined precedence graph as shown in fig 2.5 [65]. The MiMALBP

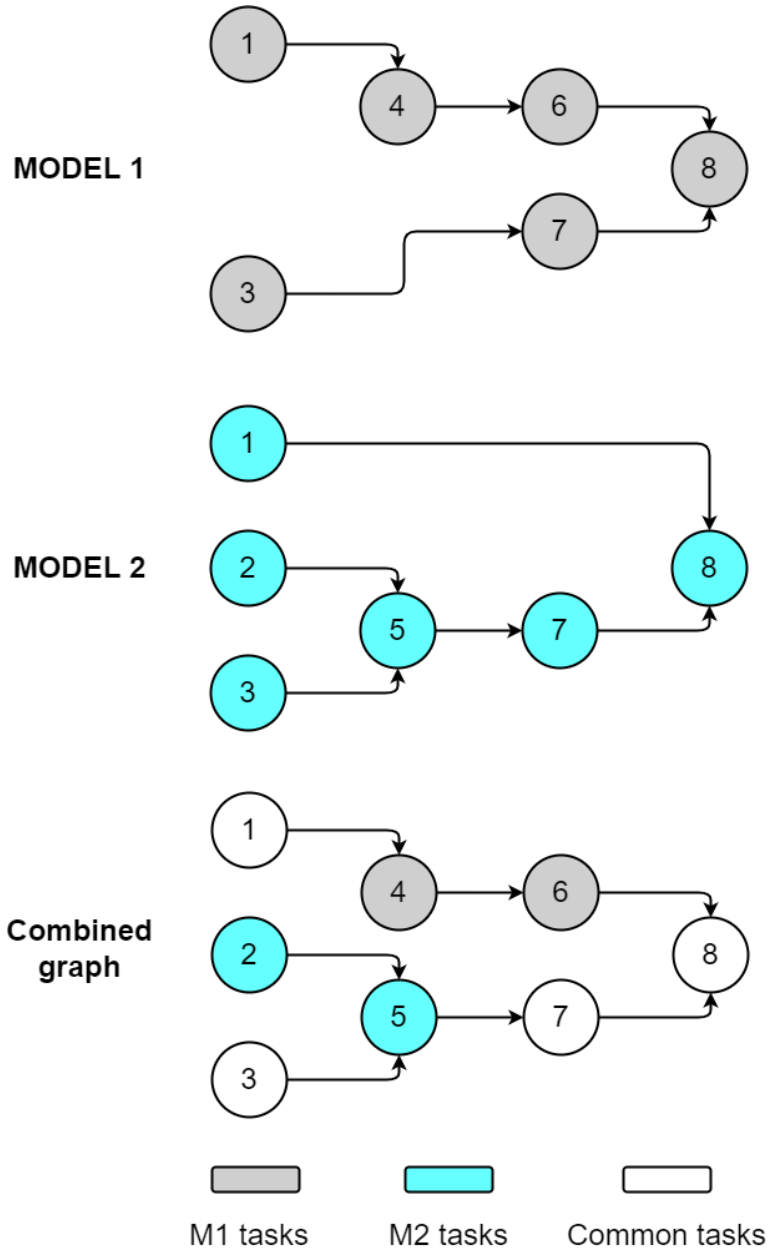

Figure 2.5: Combined precedence relations graph.

is also classified into four types; type 1, type 2, type E, and type F. All these problem types have the same objectives as those types of the SALBP.

In the literature, we can find several studies that have been done to solve a variety of MiMABLP types. Noorul *et al.* [66] have addressed the MiMALBP type 1. They proposed a hybridization between the classical genetic algorithm and the ranked positional weight (RPW) heuristic. Mamun *et al.* [67] have

proposed a genetic algorithm-based procedure to solve the MiMALBP with the aim of minimizing the total number of workstations. Akpinar *et al.* [68] have addressed the MiMALBP type 1 with sequence-dependent setup times. In order to solve the problem, they proposed a hybridization between the ant colony optimization (ACO) and the genetic algorithm (GA). Sadeghi *et al.* [69] have proposed a variable neighborhood descent method in order to solve the MiMALBP with the objectives of minimizing the number of workstations and smoothing the operator's workload.

Thiruvady *et al.* [70] have presented an ant colony optimization approach which is based on learning permutations of the operations in order to solve the MiMALBP type 1. Lalaoui *et al.* [71] have developed a versatile generalized simulated annealing using a fuzzy controller to tackle the MiMALBP type 2. Zhang *et al.* [72] have addressed a robust MiMALBP with interval task times. To solve the problem, they proposed a genetic algorithm in hybridization with a local search procedure and a discrete Levy flight. Gokcen *et al.* [73] have developed a binary integer formulation for the MIMALBP type 1. El-Alfy *et al.* [74] have addressed a multi-objective MiMALBP in order to minimize the idle time at the workstations, reduce the delay and avoid waste of production, improve the quality of assembled products, minimize the number of workstations. Erel *et al.* [75] have presented a shortest-route formulation of the MiMALBP. The formulation is based on an algorithm that is used to solve the single-model version. The objective of the study is to minimize the sum of idle times in the line.

Gokcen *et al.* [76] have addressed the MiMALBP type 1, and to solve it, they proposed a goal programming model based on the concepts of earlier researchers, and other models developed for the SALBP. In our contribution [14], we proposed a hybridization between the greedy randomized adaptive search procedure (GRASP) and the genetic algorithm in order to solve the MiMALBP type 2. The ranked positional weight heuristic is used in the GRASP to construct the solutions. In our second contribution [13], we solved the MiMALBP type 1 using a Hybrid reactive greedy randomized adaptive search procedure (RGRASP). Sivasankaran *et al.* [77] have presented seven priority rules in conjunction with a station-oriented approach in order to solve the MiMALBP type 2. Zhang *et al.* [78] have designed an ant colony optimization algorithm to

solve the MiMALBP with aim of minimizing the number of workstations.

Zhang *et al.* [79] have proposed a multi-objective genetic algorithm for the MiMALBP in order to minimize the cycle time of the line based on the demand ratio of each model. Razali *et al.* [80] have proposed mathematical modeling of the MiMALBP type 2 with resources constraints. Liu *et al.* [81] have proposed an improved genetic algorithm to solve the MiMALBP with objectives of minimizing station complexity, minimizing workload differences between different models, and maximizing productivity. Lia *et al.* [82] have developed a decentralized approach using a multi-agent-based framework to solve the MiMALBP type 1. In this system, a tabu search technique is applied in the line balancing process. A modified ranked positional weight is used to produce the initial solution, and a restricted neighborhood strategy is used to adjust the workloads of machines. Hop *et al.* [83] has addressed the MiMALBP with fuzzy processing time, and with the aims of minimizing the number of workstations, minimizing the total idle time, and maximizing the balancing coefficient. To solve the problem, a fuzzy binary linear programming model was formulated. Burduk *et al.* [84] have presented a heuristic and simulation-based approach for balancing both mixed and multi-model assembly line balancing problems.

2.3.2.2 Multi-model assembly line balancing problem (MuMALBP)

The multi-model assembly line balancing problem is related to those lines that produce different models in batches. The size of each batch is more than one, and the configuration of the assembly line is changed when a new batch is started, unlike the mixed-model assembly line which is designed one time with one configuration by which several models can be assembled in an intermixed sequence.

Regarding the literature, the MuMALBP is the least studied in comparison with all the author ALB problems. Chen *et al.* [85] have developed a two-phase adaptive genetic algorithm to solve MuMALBP in the thin film transistor-liquid crystal display. The objectives of this study are to minimize the number of assigned workers and the workstations opened. Hao *et al.* [86] have proposed a genetic algorithm for the MuMALBP. Jafari *et al.* [87] have

presented a multi-objective mixed-integer linear programming model for balancing a multi-model assembly line with three objectives (minimizing the cycle time, maximizing the number of common tasks assigned to the same workstations, and maximizing the level of workload distribution smoothness between workstations). Pereira *et al.* [88] has addressed the MuMALBP found in the textile industry, and to solve the problem, a hybrid method that combines classical methods for line balancing with an estimation of distribution algorithm.

2.3.2.3 Robotic assembly line balancing problem (RALBP)

The RALBP is a complex problem related to those lines that require robots in order to perform some repetitive, complex tasks (see fig 2.6). In this problem, there are two types of assignments; the assignment of tasks and the assignment of robots. The assignment of robots is based on several objectives, for instance, choosing the best robots that can complete tasks in a minimal amount of time, choosing the best robots that consume less energy, and so on. The RALBP is classified into several types; type 1, type 2, type E, type F, type Cost, and type O. The objectives in (type 1, type 2, type E, and type F) are the same as the other ALB problems. A RALBP is classified as a type Cost if the monetary and economic aspects are the main objective in the configuration of the line. Problems that are not classified as types 1, 2, E, F, and cost will be classified in type O (O for others) [11]. Like other ALB problems, the assumptions of the RALBP change depending on the situation, but there are some basic assumptions used in modeling the problem:

- The robotic assembly line is balanced for a single model.
- The assembly tasks are the smallest work element and cannot be subdivided among two or more workstations.
- The task times are deterministic.
- The precedence relations between tasks are known.
- The workstation can be used to execute any task if its robot is capable of completing it.
- Only one robot is assigned to a workstation.
- All robots can be used without any capacity limitation or breakdown.

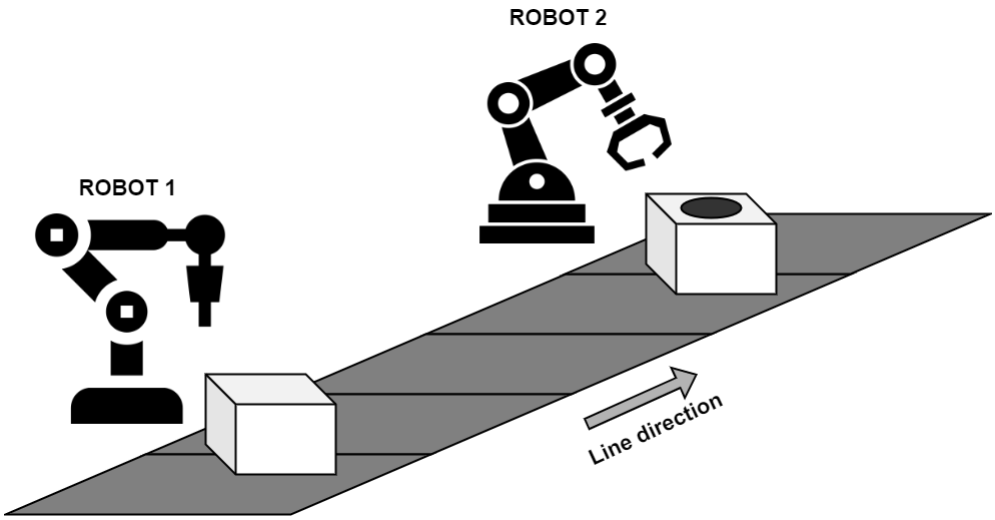

Figure 2.6: Robotic assembly line.

- The robot movement, tool changing, and setup times are negligible.

Concerning the literature, the RALBP has been addressed in several works. Li *et al.* [89] have addressed a cost-oriented RALBP with setup times to minimize the cycle time and total purchasing cost simultaneously. In order to formulate the problem, they developed a mixed-integer programming model, and to solve it, they developed two algorithms, the elitist non-dominated sorting genetic algorithm (NSGA-II) and an improved multi-objective artificial bee colony (IMBCA). Sun *et al.* [90] have proposed an effective hybrid algorithm fusing the estimation of distribution algorithm, and branch-and-bound (B&B) based knowledge to solve the RALBP with the aim of minimizing the cycle time of the line. Abidin *et al.* [91] have proposed a goal programming approach to solve the RALBP with the objectives of minimizing the cycle time, number of workstation and robot cost. Li *et al.* [92] have addressed the RALBP type 2. They have developed four mixed-integer linear programming models encoded in CPLEX for small-sized problem instances, and to solve large-sized problems, they proposed two simulated annealing algorithms: original simulated annealing algorithm and restarted simulated annealing algorithm.

Borba *et al.* [93] have proposed lower bound, exact, and heuristic algorithms for the RALBP type 2. In the lower bound, the chain decomposition is used to explore the graph dependencies, and the exact approach includes a linear mixed-integer programming model and a branch-bound-and-remember

algorithm. he heuristic is an iterative beam search. Cil *et al.* [94] have developed an efficient iterative beam search algorithm to solve the RALBP with the aim of minimizing the cycle time. Daoud *et al.* [95] have proposed two meta-heuristics which are the ant colony optimization (ACO), and the particle swarm optimization (PSO) to solve the RALBP with the aim of maximizing the line efficiency (type E). Daoud *et al.* [96] have addressed the RALBP type E. They have used a discrete event simulation model to evaluate the performance of the system. Also, they have proposed three resolution methods which are based on the ant colony optimization, particle swarm optimization, and genetic algorithm. Gao *et al.* [97] have proposed an innovative genetic algorithm hybridized with local search to solve the RALBP type 2.

Janardhanan *et al.* [98] have addressed the RALB type 2 by considering the sequence-dependent setup times. They formulated a mathematical model for the problem, and to solve small-sized problems, they used the CPLEX solver. Also, they implemented a migrating birds optimization (MBO) algorithm and some meta-heuristics to solve the problem. Kim *et al.* [99] have addressed the RALBP taking into consideration the limited space to store the parts and tool capacity of the robot hand. To solve this problem they proposed a cutting plan algorithm. Levitin *et al.* [100] have proposed a genetic algorithm to solve the RALBP with the aim of maximizing the production rate of the line (type 2). Rabbani *et al.* [101] have developed an augmented Multi-Objective particle swarm optimization (AMOPSO) in order to solve mixed model four-sided assembly line balancing problem considering the collaboration of humans and robots. Mukund *et al.* [102] have proposed a particle swarm optimization method (PSO) to solve the RALBP with the objectives of minimizing the cycle time and maximizing the production rate. The results obtained by the PSO are improved by using a local exchange procedure.

Mukund *et al.* [103] have proposed two bio-inspired search algorithms which are the particle swarm optimization (PSO), and hybrid cuckoo search and particle swarm optimization (CS-PSO) to solve the RALBP type 2. pant *et al.* [104] have addressed the RALBP with the aim of maximizing the line efficiency by minimizing the energy consumption of the assembly line. To solve the addressed problem, two meta-heuristics were implemented: particle swarm optimization (PSO) and differential evolution (DE). Pereira *et al.*

[105] have addressed the cost-oriented RALBP to maximize line efficiency. To solve the problem, they proposed an elitist memetic algorithm, a combination of a genetic algorithm with other optimization methods. Chi *et al.* [106] have addressed the RALBP with two objectives. The primary objective is to minimize the number of workstations, and the secondary objective is to minimize the energy consumption of the line. To formulate the problem, they proposed a mixed-integer linear programming (MILP). A simulated annealing algorithm was developed to solve the problem. Yousefelahi *et al.* [107] have proposed three versions of multi-objective evolution strategies (MOES) to solve the RALBP with the objectives of minimizing the total cycle time of the line, robot setup costs, and robot costs. Zhou *et al.* [108] have proposed an improved multi-objective immune clonal selection algorithm to solve bi-objective RALBP (minimizing assembly line area and the cycle time) considering time and space constraints.

2.3.2.4 U-shaped assembly line balancing problem (UALBP)

The U-shaped assembly line balancing problem (UALBP) is one of the complex general ALB problems. It is related to those assembly lines that have a U-shaped layout. In a U-shaped assembly line workstations are arranged in a U-shaped layout. The advantage of the U-shaped line is that some workstations can be revisited for some assembly tasks. In addition to this advantage, some workstations which are idle be exploited to perform other assembly tasks which helps to improve workstation utilization [3]. Another characteristic of the U-shaped assembly line is that some tasks share the same operator (human or robot) are shown in fig 2.7. This characteristic does not change the precedence relations constraints [35].

Several works that have dealt with the UALBP can be found in the literature. Pinarbasi *et al.* [109] has proposed two chance-constrained nonlinear models for the UALBP with aim of minimizing the number of workstations. the first model belongs to the mixed-integer programming (MIP) category, and the second one is constraint programming (CP). Yilmaz [110] has addressed an integrated bi-objective objective U-shaped assembly line balancing and parts feeding problem. The problem includes two different objectives, namely minimizing the operational cost and maximum workload imbalance. The second

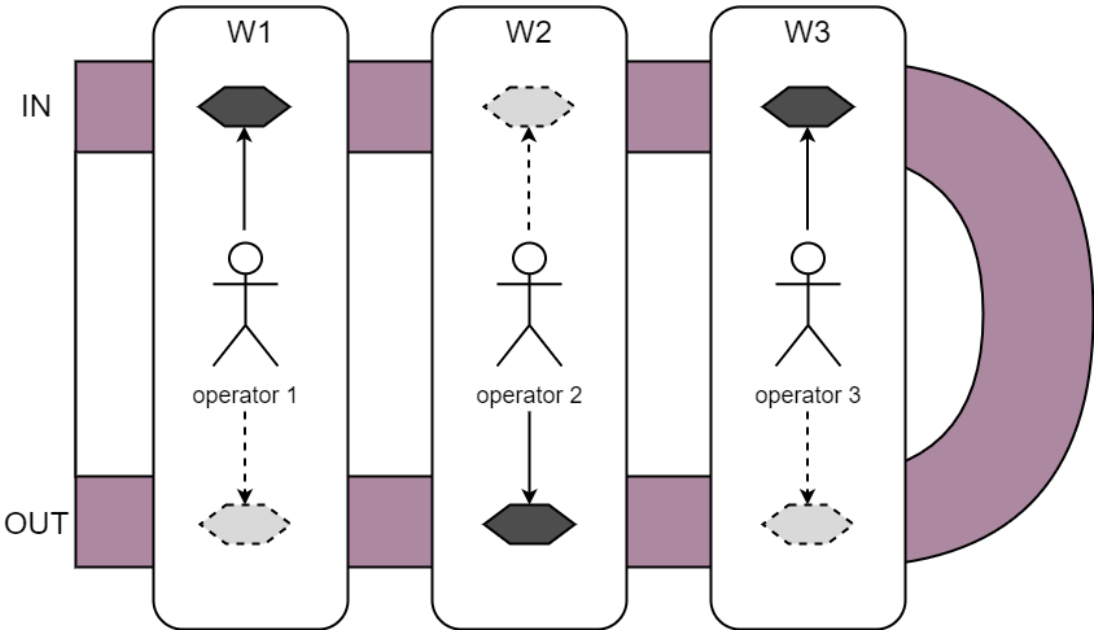

Figure 2.7: U-shaped assembly line.

version of the augmented ϵ-constrained (AUGMECON2) method was used to find the Pareto-optimal solutions. Sahin *et al.* [111] have developed a grouping genetic and simulated annealing algorithms to solve UALBP type 2. For medium and small-sized problems, they used a mathematical formulation. Li *et al.* [112] have addressed the UALBP type 1 with uncertain task times. An uncertain programming model was used in this study to minimize the number of workstations with uncertain task times and to find the optimal solution, they implemented a method based on branch and bound and remember algorithm.

Zhang *et al.* [113] have dealt with the UALBP considering the preventive maintenance scenarios. The objectives of this work are minimizing the cycle time and total assignment plan alteration cost. To solve the problem, they implemented a multi-objective JAVA algorithm. Zhang *et al.* [114] have proposed an improved flower pollination algorithm for solving the UALBP type 2 with energy consideration. Nourmohammadi *et al.* [115] have implemented a water-flow-like algorithm to solve the UALBP with two objectives: maximizing line efficiency as a primary objective and minimizing the workload variation as a secondary objective. Zhang *et al.* [116] have addressed the UALBP type 2 with workers assignment. To solve the problem, they used an enhanced migrating birds optimization (EMBO). Ajenblit *et al.* [117] have developed a

genetic algorithm in order to solve the UALBP with the aim of minimizing the number of workstations. Avikal *et al.* [118] have proposed a heuristic based on the critical path method to deal with the UALBP with aim of minimizing the number of workstations in line.

Fattahi *et al.* [119] have proposed an efficient integer programming formulation with logic cuts for the UALBP with the aim of minimizing the number of workstations on the line. Ghadiri *et al.* [120] have addressed the UALBP type 1. To solve the problem, they developed a competitive hybrid approach based on the grouping evolution strategy algorithm. The proposed approach is based on two heuristics: the ranked positional weight (RPW) and COMSOAL algorithm. Hwang *et al.* [121] have solved the UALBP with the aim of minimizing the number of workstations by using a multi-objective genetic algorithm. Jonnalagedda [122] have used a simple genetic algorithm in order to minimize the cycle time in a U-shaped assembly line (UALBP type 2). Li *et al.* [123] have proposed a heuristic approach based on multiple rules and an integer programming model to solve the UALBP with the aim of minimizing the cycle time. Li *et al.* [124] have proposed a branch-bound-and-remember algorithm to solve the UALBP with the aim of minimizing the number of workstations.

Li *et al.* [125] have developed an enhanced beam search heuristic algorithm based on five lower bounds and four dominance rules to solve both the UALBP type 1 and UALBP type 2. Sresracoo [126] have proposed a differential evolution algorithm to solve the type 1 UALBP. Khorram *et al.* [127] have presented hybrid meta-heuristic algorithms to solve the UALBP with aim of optimizing simultaneously the number of workstations, activity performing quality, and equipment cost. Ogan *et al.* [128] have proposed a branch and bound method to solve the UALBP with the aim of minimizing the total equipment cost. Zhang *et al.* [129] have used an exact method to minimize the number of workstations in a UAL (UALBP type 2). Zhang *et al.* [130] have proposed an integer programming formulation to solve the UALBP type 1 with fuzzy task times. The proposed model was implemented in Lingo solver 9.0 extended version.

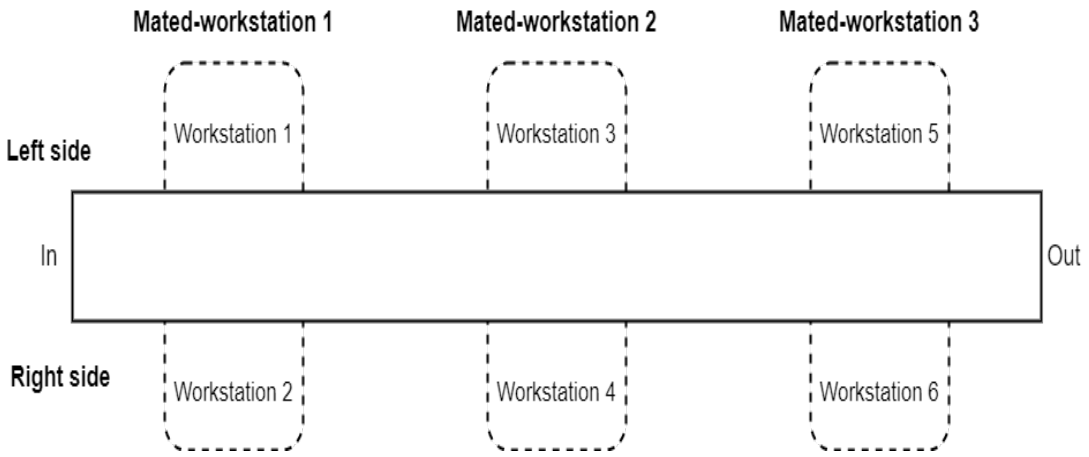

Figure 2.8: Two-sided assembly line.

2.3.2.5 Two-sided assembly line balancing problem (TALBP)

The assembly lines can be classified into two categories, one-sided assembly lines, and two-sided assembly lines. The two-sided assembly lines are generally used to assemble large-sized products such as trucks, buses, and cars. The two-sided assembly lines generally have a pair of workstations facing each other almost in all operations [5]. For instance, in fig 2.8, workstation 1 and workstation 2 form the mated workstation 1, and workstation 3 with workstation 4 form the mated workstation 2, etc. The TALBP is linked to these types of lines where tasks are divided into three types: L-type, R-type, and E-type. L-type and R-type are tasks that must be performed from the left and right sides respectively. Whereas E-type tasks can be performed either from the left or the right sides [131]. The TALB problems can be also classified into four types like the other ALB problems: TALBP type 1, TALBP type 2, TALBP type E, and TALBP type F.

Regarding the literature, we can find several studies that have been done to deal with a variety of TALBP. Ozcan *et al.* [132] have presented a tabu search algorithm to solve the TALBP with the objectives of maximizing the line efficiency and maximizing the smoothness index. Ozcan *et al.* [133] have proposed goal programming and fuzzy programming models for the TALBP with two objectives: minimize the number of mated workstations as the primary objective and the number of workstations as the secondary objective. Ozbakir

[134] have addressed the TALBP type 1 with zoning constraints. They developed a Bees colony algorithm to solve the problem. Kizilay [135] have proposed mixed-integer linear programming and constraint programming (CP) models to solve the TALBP with multi-operator workstations in order to minimize the number of mated workstations.

Baykasoglu [136] have proposed an ant colony-based heuristic to solve the TALBP type 1 (minimize the number of workstations) with zoning constraints. Duan *et al.* [137] have proposed an improved artificial bee colony algorithm with MaxTF heuristic rule to solve the TALBP with the aim of minimizing the cycle time for a given number of mated workstations. Gansterer [138] have addressed both one and two-sided ALBP with real-world constraints. They have applied three known meta-heuristics to solve the problems: the genetic algorithm, the differential evolution algorithm, and the tabu search algorithm. Kang *et al.* [139] have addressed a multi-objective TALBP (minimizing the number of workstations, minimizing cycle time, maximizing line efficiency, minimizing smoothness index, and minimizing workstation idle time). They first constructed a fuzzy multi-objective linear programming-weighted model (FMOLP-W) to solve the problem. Also, they have proposed an evolutionary genetic algorithm for large-sized problems that cannot be solved using the FMOLP-W. Khorasanian [140] have developed a simulated annealing algorithm to solve the TALBP with the objectives of minimizing the number of workstations, and the number of mated workstations, and maximizing the assembly line tasks consistency (ATC).

Kim *et al.* [141] have proposed a genetic algorithm in order to solve the TALBP with the aim of minimizing the number of workstations (type 1). Kim *et al.* [142] have proposed a mathematical model and a genetic algorithm to solve the TALBP type 2. In the GA, they adopted the strategy of localized evolution and steady-state reproduction to promote population diversity and search efficiency. Lei *et al.* [143] have developed a variable neighborhood search (VNS) algorithm to solve the TALBP type 2. Li *et al.* [144] have addressed the TALBP type 1. They have developed two decoding schemes with reduced search space to balance the workload within a mated workstation and reduce sequence depending on idle time. Then, they extended a simple iterated greedy algorithm for the TALBP. Li *et al.* [145] have presented a branch, bound, and remember

(BBR) algorithm in order to minimize the number of mated workstations in a two-sided assembly line (TALBP type 1).

Li *et al.* [146] have addressed the TALBP that considers uncertain task time attributes and incompatible task sets. To solve the problem, they used a simulated annealing algorithm accelerated with lower bounds. Roshani [147] have addressed a cost-oriented TALBP. They proposed a mixed-integer programming model to solve the problem optimally, and to solve medium and large-size problems, they developed a heuristic based on a simulated annealing algorithm. Taha *et al.* [148] have developed a genetic algorithm in order to solve the TALBP with the objectives of minimizing the number of mated workstations and workstations to increase the line efficiency. Tang *et al.* [149] have addressed TALBP type 2 in their work. To solve the problem, they proposed an improved discrete artificial bee colony (DABC) algorithm. Tapkan *et al.* [150] have proposed a Bees algorithm to solve a constrained fuzzy TALBP with three objectives: maximizing the work slackness index, minimizing the total balance delay, and maximizing line efficiency. Tapkan *et al.* [151] have addressed a constrained TALBP type 1. They proposed a mathematical programming model to describe the problem formally. To solve the problem, they used two different swarm intelligence-based search algorithms (bees algorithm and artificial bee colony algorithm).

Yang *et al.* [152] have proposed an improved genetic simulated annealing for stochastic TALBP type E (maximizing the line efficiency). Tuncel *et al.* [153] have addressed a multi-objective (minimizing the number of workstations and smoothing the workload balance of the workstations) TALBP with zoning constraints. To solve the problem they used a teaching-learning-based optimization algorithm. Wang *et al.* [154] have developed a hybrid imperialist competitive (LAHC) algorithm to solve a multi-constrained (the restriction of operator number at each workstation, positional constraints, zoning constraints, and synchronous task constraints) TALBP with the objectives of minimizing the number of mated workstations (line length) and the number of workstations. Wu *et al.* [155] have proposed a branch-and-bound algorithm to solve the TALBP with the aim of minimizing the length of the line (minimizing the number of mated workstations). Zhong *et al.* [156] have developed an effective discrete artificial fish swarm algorithm is developed to solve the

cost-oriented TALBP with the objectives of minimizing the construction cost and the number of mated workstations.

2.3.2.6 Parallel assembly line balancing problem (PALBP)

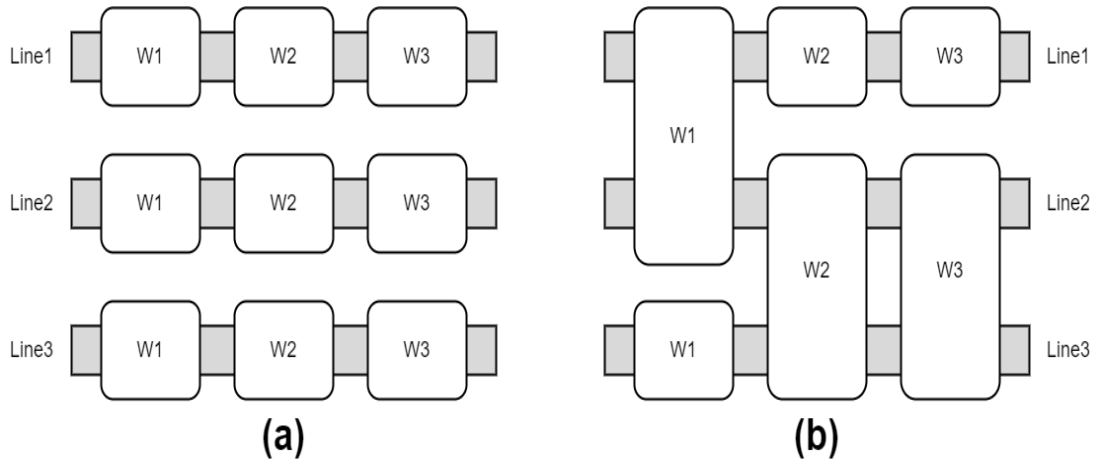

Figure 2.9: Parallel assembly lines with a) independent workstations, b) multi-line workstations.

Parallel assembly lines are two or more neighboring lines located in parallel. These lines can be balanced independently ((a) in fig 2.9) or with multi-line workstations as shown in fig ((b) in fig 2.9). Each assembly line produces one single-model or mixed/multi-model and can work with different or the same cycle time [28]. Parallel assembly lines have a higher equipment cost but their are more flexible and reliable. They also increase productivity, product quality, and workers satisfaction [157]. In the literature, we can find a few works that have dealt with the pure parallel assembly line balancing problem. [158] have proposed a hyper-approach based on simulated annealing to solve a stochastic PALBP with equipment costs.

Ozcan *et al.* [159] have addressed the PALBP with stochastic task times. To model the problem, they used chance-constrained, piecewise-linear and mixed integer programming (CPMIP) formulation. A tabu search algorithm was used to solve the problem. Baykasoglu [160] have proposed a multi-colony ant algorithm to solve the PALBP type 1 with fuzzy cycle and task times. Bard *et al.* [161] have proposed a dynamic programming (DP) algorithm for solving the PABLP with dead time. Baykasoglu *et al.* [162] have proposed an ant colony optimization-based algorithm in order to solve the PALBP with two objectives:

minimizing the number of workstations and the workstations' idle times. Kara *et al.* [163] have used two programming approaches to balance parallel assembly lines with precise and fuzzy goals. Ozbakir *et al.* [164] have developed a multiple-colony ant algorithm to solve the PALBP with two objectives: minimizing the idle time and maximizing the line efficiency.

2.3.2.7 Mixed assembly line balancing problem (Mixed ALBP)

Any ALB problem that regroups more than one ALBP can be classified as Mixed ALBP. The word mixed here means that there is a mixture of problems. For example, solving a robotic u-shaped multi-model ALBP means that we have three different problems that must be solved together: robotic ALBP, U-shaped ALBP, and multi-model ALBP which increases the global complexity. In the literature, we can find several papers that have dealt with this ALBP category. Cil *et al.* [165] have proposed a mathematical model and an iterative beam search (IBM), best search method based on IBS (BIBS) and cutting BIBS (CBIBS) algorithms to solve the robotic parallel assembly line balancing problem type 2. Cil *et al.* [166] have proposed addressed the type 2 mixed-model assembly line balancing problem with physical human-robot collaboration. They proposed mixed-integer linear programming (MILP) to solve small-size problems. Also, they used a bee algorithm and artificial bee colony to solve large-sized problems.

Akpinar *et al.* [167] have proposed a hybrid genetic algorithm to solve the type 1 mixed-model assembly line balancing problem with parallel workstations and zoning constraints. Agpak *et al.* [168] have developed a bi-objective 0-1 integer programming model to solve a two-sided U-shaped assembly line balancing problem with zoning constraints. The addressed objectives are the minimization of the number of individual stations, and the minimization of the number of positions. Yang *et al.* [169] have addressed the mixed-model two-sided assembly line balancing problem with two objectives: minimize the number of mated-workstations as a primary objective, and minimize the number of workstations as a secondary objective. To solve the problem, they proposed an effective variable neighborhood search (VNS) algorithm. Cil *et al.* [170] have proposed a heuristic algorithm based on beam search in order to solve the robotic mixed-model assembly line balancing problem type 2.

Ozcan *et al.* [171] have developed a genetic algorithm to solve the stochastic mixed-model U-shaped assembly line balancing and sequencing problem. Chutima *et al.* [172] have proposed a particle swarm optimization algorithm with negative knowledge (PSONK) to solve multi-objective two-sided mixed-model assembly line balancing problems. Esmaeilian *et al.* [173] have developed an efficient tabu search (TS) approach to solve mixed-model parallel assembly line balancing problem type 2. Zhang *et al.* [174] have proposed a Pareto artificial bee colony algorithm (PABC) to solve the energy-efficient U-shaped robotic assembly line balancing problem with the objectives of minimizing the cycle time and the total energy consumption. Huang *et al.* [175] have developed a combinatorial Benders decomposition-based exact algorithm to solve the mixed-model two-sided assembly line balancing problem type 1. Kammer *et al.* [176] have proposed a heuristic algorithm to solve the type 2 multi-model robotic assembly line balancing problem.

Kucukkoc [177] have proposed a mathematical model and ant colony optimization-based approach with optimized parameters for solving the type E parallel two-sided assembly line balancing problem. Kucukkoc *et al.* [178] have developed a flexible agent-based ant colony optimization algorithm to solve the mixed-model parallel two-sided assembly line balancing problem. Li *et al.* [26] have solved the two-sided robotic assembly line balancing problem type 2 using a co-evolutionary particle swarm optimization algorithm. Grasler *et al.* [23] have proposed a discrete cuckoo search algorithm to solve the two-sided robotic assembly line balancing problem. Li *et al.* [179] have proposed a migrating birds optimization algorithm in order to solve the type 2 robotic U-shaped assembly line balancing problem. Li *et al.* [180] have addressed the type 2 robotic two-sided assembly line balancing problem with consideration of sequence-dependent setup times and robot setup times. To solve the problem, the authors used a set of meta-heuristics which are the local search algorithm, swarm intelligence algorithm, and co-evolutionary swarm intelligence algorithm.

Li *et al.* [181] have proposed two simple local search methods, the iterated greedy algorithm and iterated local search algorithm to deal with type 1 mixed-model two-sided assembly line balancing problems. Manavizadeh *et al.* [182] have developed a simulated annealing algorithm for solving the U-shaped mixed-model assembly line balancing problem type 1 considering

human efficiency and the Just-In-Time approach. Mukund *et al.* [183] have proposed a particle swarm optimization for solving the robotic U-shaped assembly line balancing problem with the aim of minimizing the cycle time. Rabbani *et al.* [184] have proposed a non-dominated sorting genetic algorithm (NSGA-II), and a multi-objective particle swarm optimization (MOPSO) in order to deal with the type 2 robotic mixed-model assembly line balancing problem with four objectives: minimize robot purchasing costs, robot setup costs, sequence-dependent setup costs, and cycle time.

Rabbni *et al.* [185] have addressed the type 1 mixed-model assembly line balancing problem with parallel workstations. The authors used a non-dominated sorting genetic algorithm (NSGA-II) and multi-objective particle swarm optimization (MOPSO) to solve the problem. Zhang *et al.* [186] have proposed a Hybrid Pareto Grey Wolf Optimization (HPGWO) for low-carbon and low-noise U-shaped robotic assembly line balancing problems. Yagul *et al.* [187] have developed an algorithm for solving the U-shaped two-sided assembly line balancing problem. yadav *et al.* [188] have proposed a mathematical model for solving the robotic two-sided assembly line balancing problem with zoning constraints. yadav [189] have developed a mathematical model in order to solve the parallel two-sided assembly line balancing problem with tools and tasks sharing. Tapkan *et al.* [190] have proposed a bees algorithm and an artificial bee colony algorithm for solving the parallel two-sided assembly line balancing problem with walking times. Sparling *et al.* [191] have solved the mixed-model U-shaped assembly line balancing problem using an approximation algorithm. Roshani *et al.* [192] have proposed a mathematical model and a simulated annealing approach to deal with the mixed-model multi-manned assembly line balancing problem.

Chutima *et al.* [193] have developed a fuzzy adaptive biogeography-based algorithm to solve the multi-objective mixed-model parallel assembly line balancing problem type 1. Zhang *et al.* [194] have solved the mixed-model U-shaped assembly line balancing problem with Fuzzy Times using an improved heuristic procedure based on the traditional ranked positional weight method, and some improvements are made on fuzzy number operation and two-direction search for U-line layout. Zhang *et al.* [195] have addressed the mixed-model

multi-manned assembly line balancing problem with four objectives: minimizing the total number of operators, minimizing the number of workstations, minimizing the total number of operators whose finishing times exceed cycle time, and minimizing the total number of workstations whose finishing times exceed cycle time. To solve the problem, the authors used a robust mixed-integer linear programming (MILP) model and a robust solution generation mechanism embedded with dispatching rules.

2.4 Conclusion

In this chapter, we have presented the state of the art on the assembly line balancing problem. We have described the standard problem and its mathematical formulation, from which all other formulations are inspired. Also, we have discussed the different assembly line shapes and their structures. Furthermore, we have illustrated the differences between the assembly lines in terms of production types. We have explained how the new ALBP versions were created and what their main characteristics are. Also, we have given the assembly line classification in detail. We have described all existing assembly line balancing problems and the methods used to solve them. In addition, we have reviewed all the existing research papers that have been done to tackle the different ALBP problems. In the following chapters, we will present all our contributions.

Chapter 3

Energy Efficient Robotic Mixed Model Assembly line Balancing Problem

3.1 Introduction

The minimization of energy consumption is an important issue in robotic assembly lines where a set of robots are used as operators. In each workstation, only one robot is assigned to complete the corresponding assigned assembly tasks. In robotic mixed-model assembly lines where several models are assembled, the minimization of energy consumption is more complex since we have to find the best assignments of tasks and robots taking into consideration all models. This problem is known as the Energy-Efficient Robotic Mixed Model Assembly Line Balancing Problem (EERMiMALBP), and it's classified as Mixed ALBP based on our classification. As a solution to this problem, we proposed a new memory-based version of a bio-inspired meta-heuristic known as the Cuckoo Search Algorithm (CSA).

This chapter is organized as follows; first, we define the energy-efficient robotic mixed-model assembly line balancing problem. Then, we present our memory-based cuckoo search algorithm as a solution to this problem. After that, we present a numerical example that presents a small problem. Finally, we discuss our computational results.

3.2 The energy-efficient robotic mixed-model assembly line balancing problem

3.2.1 Problem description and assumptions

In a robotic mixed-model assembly line, several models are planned to be assembled in an intermixed sequence. Each model has a specific precedence relations graph, and models' graphs can be combined into one graph in order to decrease the complexity of the problem. According to precedence relations, assembly tasks are assigned to a set of workstations that are arranged in the line. Assembly tasks are performed by robots that have different capabilities and characteristics. In this problem, the assignment of robots is based on the energy consumed to perform assembly tasks; this means that the robots that consume less energy are preferred. The total energy consumption of the line is the sum of all energy consumed by all assigned robots. The task time of a specific task varies from one model to another one, and also depends on the robot. This means that the energy consumed for a specific model varies from one robot to another one. The energy consumed in one workstation is the sum of the energy consumed by the assigned robot when performing assembly tasks, and the energy consumed by the robot in standby mode. Each robot has specific characteristics including its speed in performing assembly tasks, energy consumption, and ability to perform all or only certain types of tasks.

The RMiMAALBP is more complex in comparison with existing problems related to the minimization of energy consumption in robotic assembly lines, and this is the result of two main factors: the heterogeneity of robots and the heterogeneity of models. The strategy underlying our approach is to find the best assignment of existing robots for each feasible task assignment. Since robots and models are heterogeneous, the search space becomes larger, which makes finding the best solution for the RMiMAALBP very challenging. The objective of solving this problem is to find a minimal energy consumption by finding the best assignment of tasks and robots that satisfy all models' requirements. Figure 3.1 presents an illustrative example. In this example, we have an assembly line that is configured to assemble two different models, and there are different robots assigned to the workstations in order to perform assembly tasks according to the combined graph. This configuration is feasible but

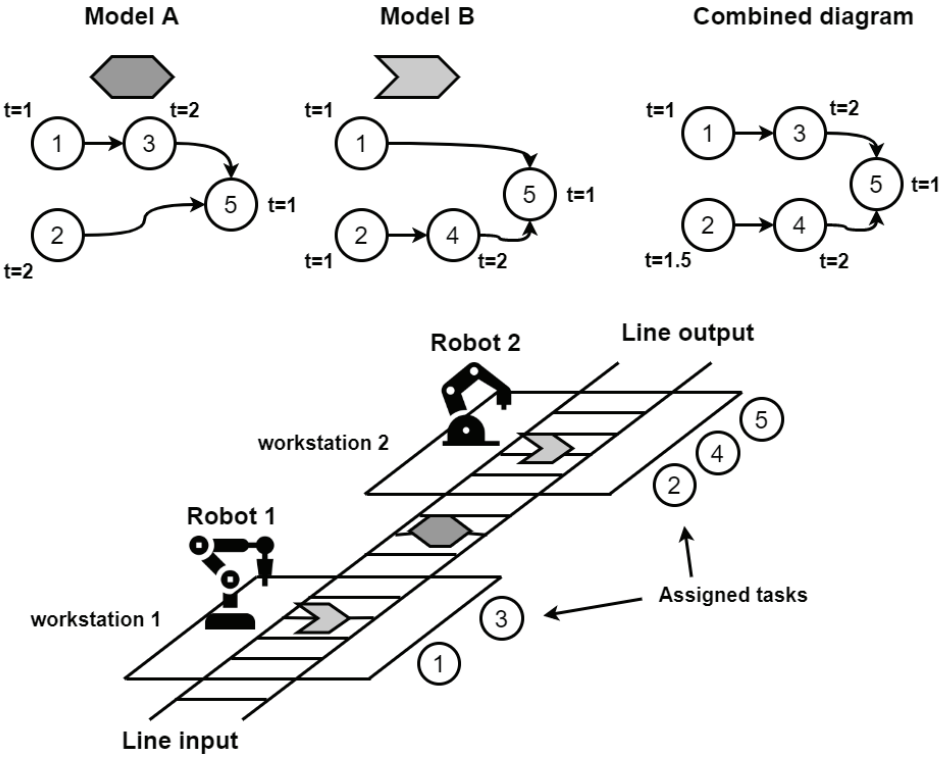

Figure 3.1: Illustrative example.

maybe there exists another configuration that is better in terms of the energy consumed by robots.

The assumptions considered in our model formulation are as follows:

1. The models are assembled in an intermixed sequence in a straight assembly line.
2. An assembly task is assigned to only one workstation.
3. The assignment of tasks is based on precedence relations.
4. The task time depends on the assigned robot.
5. There are several types of robots, and there is no limit for each type.
6. The same robot type can be assigned to different workstations.
7. Only one robot is assigned to a workstation.
8. Robots can perform any assembly task.
9. The energy consumed by a robot varies from one model to another.

10. Based on the energy consumed by robots to produce each model, the average energy consumed can be calculated and used while optimizing the line.

3.2.2 Mathematical formulation

Notations

Parameters:

N number of tasks.
i task index.
W number of workstations.
k workstation index.
r robot index.
t_{ri} processing time needed by robot r to complete task i.
e_{ri} energy consumed by robot r to perform task i.
es_r energy consumed each time unit by robot r in the standby period.
P_i immediate predecessors of task i
ECP_{rk} energy consumed by robot r to perform assigned tasks in workstation k.
ECS_{rk} energy consumed by robot r during the standby period in workstation k.

Decision variables:

X_{ik} equal to 1 if task i is assigned to workstation k, 0 otherwise
Y_{rk} equal to 1 if robot r is assigned to workstation k, 0 otherwise
TEC total energy consumed
EC_k energy consumed by workstation k
CT the cycle time of obtained assignment (or maximum workstation time).

The proposed mathematical is an amelioration of the basic models that have been used to solve the one-model robotic assembly line balancing problems. The objective considered in this mathematical formulation is minimizing the total energy consumption in the robotic mixed-model assembly line. This model is based on the energy-based model that was proposed by [196].

$$min \quad TEC = \sum_{k=1}^{W} EC_k \tag{3.1}$$

subject to:

$$EC_K = ECP_{rk} + ECS_{rk} \tag{3.2}$$

$$ECP_{rk} = \sum_{i=1}^{N} e_{ri} * X_{ik} \leq for \quad k = 1, 2, ..., W \tag{3.3}$$

$$ECS_{rk} = (CT - \sum_{i=1}^{N} t_{ri} * X_{ik}) * es_r \tag{3.4}$$

$$\sum_{k=1}^{W} k * X_{hk} \leq \sum_{k=1}^{W} k * X_{ik} \quad where\ h \in P_i \tag{3.5}$$

$$\sum_{k=1}^{W} X_{ik} = 1 \quad for \quad i = 1, 2, ..., N \tag{3.6}$$

$$\sum_{k=1}^{W} Y_{rk} = 1 \quad for \quad i = 1, 2, ..., N \tag{3.7}$$

$$X_{ik} \in \{0, 1\} \tag{3.8}$$

$$Y_{rk} \in \{0, 1\} \tag{3.9}$$

The objective function (3.1) minimizes the total energy consumption. Eq. (3.2) is used to calculate the energy consumption in one workstation. Eq. (3.3) calculates the energy consumed by robot r to perform assigned tasks at workstation k. Eq. (3.4) determines the energy consumed by robot r during the standby mode at workstation k. Eq. (3.5) ensures that all precedence relations among tasks are respected. Eq. (3.6) ensures that each task is assigned to only

one workstation. Eq. (3.7) ensures that only one robot is assigned to one workstation. Equations (3.8) and (3.9) present the type of used decision variables.

3.3 The original Cuckoo Search Algorithm

The cuckoo search algorithm (CSA) is a nature-inspired meta-heuristic. It was introduced by Xin-she Yanf and Suach Deb in 2009 [197]. This meta-heuristic is based on the brood parasitism of some cuckoo species like Ani and Guira. The concept of these cuckoos is that they lay their eggs in the nests of other host birds, and in some cases, they lay and throw the eggs of the host birds in order to increase the hatching chances of their eggs [198]. The process of laying eggs is combined with the levy flight behavior of birds and fruit flies [199]. The host bird can discover the cuckoo egg, in this case, it either discards the discovered egg or abandon the nest and build a new one elsewhere [197].

In the CSA meta-heuristic solutions present the host bird egg, and the cuckoo egg presents the new solution. The goal is to replace the old solution with the new solution. Three rules are used in the CSA: each cuckoo lays only one egg at a time in a randomly chosen nest, and the best nests are those that contain high-quality eggs which enable them to be passed on to the next generation. the number of available host nests is fixed and the probability of discovering laid eggs by cuckoos is Pa where $Pa \in$ [0,1] [29, 199].

Algorithm 1: Original Cuckoo Search Algorithm

```
Objective function: f(x), x = (x1, x2, ...xd)
Generate Initial population of N host nests
while Termination criterion is not met do
    Generate a cuckoo say Xi randomly using levy flights
    Choose a nest among N (say, Xj) randomly
    if f(Xi) < f(Xj) then
        Replace Xj by Xi in the population
    end
    A fraction Pa of worst nests are abandoned and new ones are built
    Keep the best solutions/nests
    Rank the solutions/nests based on their fitness and find the best
    current solution
end
```

3.4 Memory-based Cuckoo Search Algorithm (MBCSA)

In this contribution, we propose the MBCSA version to solve the RMiMALBP. In the literature, we can find two works that have dealt with the robotic assembly line balancing problem using the cuckoo search algorithm. Li *et al.* [24] have proposed a discrete CSA to solve the two-sided RALBP, and Mukund *et al.* [103] have proposed a hybrid CS particle swarm optimization algorithm to solve the RALBP type 1. In comparison with the existing CSA versions, the MBCSA is more intelligent in dealing with the addressed problem. Our meta-heuristic consists of the following steps.

3.4.1 Initialization of the first population

In the CSA, the population is a set of solutions and each solution presents a nest. In our MBCSA, the first population is generated randomly. Each solution is a feasible assignment of tasks and robots to the workstations. The assignment of tasks follows the precedence relations that are presented in the combined precedence graph. The assignment of robots is done after the assignment of tasks. The robots that consume less energy are preferred in the assignment process. The obtained solutions in this step are not necessarily optimal.

3.4.2 Generation of new cuckoo solution and memory usage

In order to generate a new cuckoo solution, a nest (say X) is chosen randomly from the existing solutions (population). Then, a list of neighbors of X (say N(X)) is generated. This process of choosing a random nest (X) is repeated until obtaining finding a nest that does not exist in the memory, if not, it stops when reaching a specific number of iterations fixed by the programmer. Neighbors are obtained by applying the swap mutation operation on X by swapping two tasks chosen randomly from the sequence. The number of neighbors is determined and fixed by the programmer. Each solution in N(X) has its own fitness value on which the ranking of neighbors is based. The best neighbor that has the best fitness value is selected as the new cuckoo solution.

The new cuckoo solution is compared in terms of energy consumption with another randomly selected solution (say Y) from the current population.

In our approach, if the new cuckoo has lower energy consumption in comparison with Y, we replace Y with this new cuckoo in the population. At the end of this step, selected nest X, and the list of neighbors N(X) are placed in the memory. The objective of this technique is to discover new solutions in future iterations by ignoring any previously visited solution that exists in the memory, which decreases the CPU time, especially when the search space is huge.

3.4.3 Replacement of abandoned solutions

The CSA abandons a fraction P_α of worse solutions (nets) and replaces them with new ones. In our MBCSA, the crossover operation is applied to create new solutions. The crossover operator is applied on two nests (parents) selected randomly from the remaining solutions. For instance, if two nests are abandoned, the crossover is applied to generate two new nests, and if more than two nests are abandoned, the crossover is repeated on other selected solutions that present a different couple. This process is finished until abandoned solutions are replaced. The reason behind choosing a new couple in each crossover operation is to obtain new solutions. In the newly generated solution, some tasks can be duplicated, and to tackle this problem, the MBCSA replace them with missing ones. Before inserting the new solutions in the population, the algorithm checks if these solutions do not exist in the memory. If not, the solutions are inserted in the population, otherwise, the MBCSA repeats the crossover process until reaching new nets that do not exist in the memory.

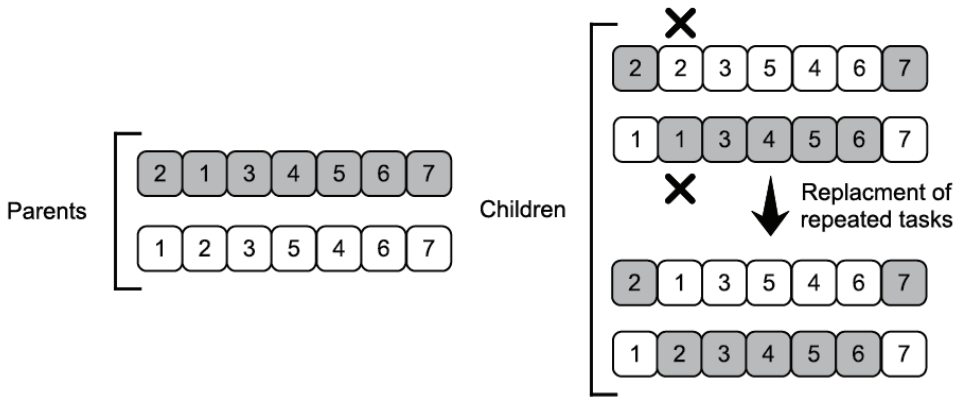

Figure 3.2: Generation of new solutions using the two-point crossover.

Algorithm 2: A memory-based Cuckoo Search Algorithm

```
Ojective funciton: f(X), X = (X1, X2, ...Xd)
Generate Initial population of N host nests
Initialize the memory M, max_search, max_generations
while max_generations do
    while max_search do
        Select randomly a nest (say, Y) from the population
        if Y is not in M then
            Reset max_search to the initial value
            Exit from while loop
        end
        Decrement max_search
        if max_search is equal to 0 then
            End the algorithm
        end
    end
    Generate a list of neighbors of Y (say, NL(Y)) using the swap
    mutation with repair mechanism
    Stock the nest Y and all its neighbors NL(Y) in M
    Select the best neighbor from NL(Y) as the new cuckoo (say, C)
    Choose a nest among N (say, H) randomly
    if f(C) < f(H) then
        Replace H by C in the population
    end
    A fraction Pa of worst nests are abandoned and new ones are built
    while max_search do
        Choose two parents from the reminder of the population
        Apply the crossover on selected parents with the repair
        mechanism
            if Generated solution are not in M then
            Reset max_search to the initial value
            Exit from while loop
        end
        Decrement max_search
        if max_search is equal to 0 then
            End the algorithm
        end
    end
    Replace abandoned solutions by generated one
    Rank the solutions based on the objective value
    Select the best solution as the best current solution
    Decrement max_generations
end
```

3.4.4 Repair mechanism

The repair mechanism is employed in the MBCSA to preserve the feasibility of solutions. The repair mechanism evaluates the feasibility of solutions by examining if the order of tasks respects the precedence relations between tasks after applying the swap mutation and the crossover. If the obtained solution is not feasible, the positions of tasks that render the solution infeasible are modified. The resultant solution, as shown in Fig 3.3, does not satisfy the priority relations depicted in the diagram; hence, the repair mechanism is used by exchanging tasks 5 and 2 the first time and tasks 5 and 3 the second time.

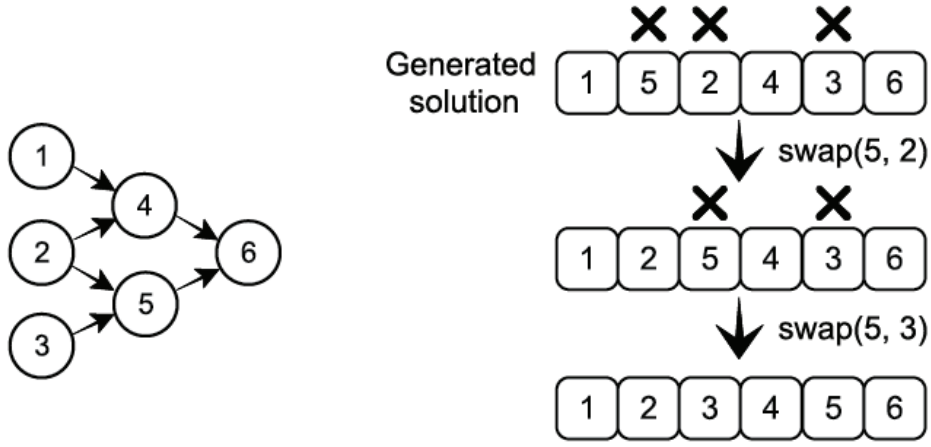

Figure 3.3: Repair mechanism.

3.4.5 Fitness evaluation

The energy-based approach provided in [200] is utilized in MBCSA to estimate the objective value (total energy consumed) of solutions in order to evaluate them. Tasks are assigned to workstations based on the energy consumption value and robots with the lowest energy consumption are selected to be assigned to workstations. The assignment starts with an initial energy consumption value E0 calculated using Eq. (10), and if all tasks cannot be assigned with this value, then, it is incremented by a small value fixed by the programmer. The assignment process is repeated until all tasks of the corresponding solution/sequence are assigned using the best robot-to-workstation assignment.

$$E_0 = [\sum_{i=1}^{N} min(e_{ri})/N_w] \tag{3.10}$$

where e_{ri} is the energy consumed by robot r ($r = 1, ..., N_r$) to perform task i and N_w is the number of workstations.

The total energy consumed at each workstation is the sum of the energy consumed by the corresponding robot while doing the given tasks and the energy consumed by this robot while in standby mode.

3.5 Numerical example

In this section, we use the MBCSA to solve a small problem that is composed of two models (A and B) and two robot types (R1 and R2). Fig 3.4 below shows the precedence graphs of models A and B and the combined graph. It is assumed in this in this numerical example that both robots consume 1 kj per one-time unit when processing tasks. While in the standby mode, R1 and R2 consume 0.5kj and 0.4 kj per one-time unit respectively. Tables 3.1 and 3.2 illustrate the parameters of each robot. The mixture of models must be taken into consideration which makes the problem more complex. In order to decrease its complexity, we combine tables 3.1 and 3.2 into one table (table 3.3). Using the combined diagram and table 3.3, we can obtain a feasible solution that satisfies all models' requirements. Table 3.3 shows the calculation of the average task time and energy consumed for each frequent task. Table 3.4 provides the parameters that were utilized to solve the numerical case. The max search parameter specifies how many times the algorithm attempts to find a random nest that does not exist in memory.

The first and the final populations are shown in table 3.5. There are ten solutions (nests) in the population, and each one (nest) presents the task and robot assignments to workstations. The table also shows the total energy spent during the processing mode (TEC-PM) and standby mode (TEC-SM) for each solution. The total energy used (TEC) is also determined, which represents the objective value and is equal to the sum of TEC-PM and TEC-SM. Based on the objective value, the reader may see that there are both bad and good solutions in the first population. For example, there are three solutions that have an

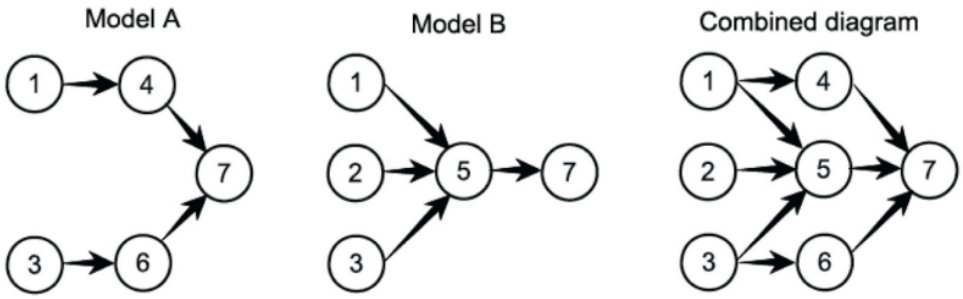

Figure 3.4: Precedence relations diagrams of the numerical example.

Table 3.1: Model A

Task	Robot 1		Robot 2	
	Task time	Energy	Task time	Energy (kj)
1	1	1	1	1
3	2	2	1	1
4	1	1	1	1
6	1	1	2	2
7	1	1	1	1

objective value of 8, one solution has an objective value of 8.1, two solutions have an objective value of 8.7, and four solutions have an objective value of 9.25. We can see after 100 generations that all nests in the final population have good solutions, with the same objective value of 8.

We may deduce from the acquired findings for the provided numerical example that the minimum energy consumed is 8, and the optimal robot assignment that gives this value is when robots 1 and 2 are assigned to workstations 1 and 2, respectively. Figure 3.5 illustrates one solution chosen from the final population. In workstation 1, robot 1 does tasks 1, 2, and 3, while robot 2 performs duties 6, 4, 5, and 7 in workstation 2.

3.6 Computational results and discussion

The proposed MBCSA was developed in Python on a personal computer with an Intel Dual-core 1.7 GHz CPU and 8 GB of memory. Six problems of varying sizes were developed during this study to test the algorithm's performance

Table 3.2: Model B

Task	Robot 1		Robot 2	
	Task time	Energy	Task time	Energy (kj)
1	1	1	1	1
2	1	1	2	2
3	2	2	2	2
5	2	2	1	1
7	1	1	1	1

Table 3.3: Combined model A and model B

Task	Robot 1		Robot 2	
	Task time	Energy	Task time	Energy (kj)
1	1	1	1	1
2	1	1	2	2
3	2	2	1.5	1.5
4	1	1	1	1
5	2	2	1	1
6	1	1	1	1
7	1	1	1	1

and are available at https://github.com/Belkharroubi-Lakhdar/RoboticMixed-Model-AL.git. Each problem has its own characteristics, as indicated in table 3.6, including the precedence relations diagram, assembly tasks, number of models, number of robots, and number of workstations. There are 12 tasks in problem 1, as well as two models (A and B), two robots (1 and 2), and three workstations. Problem 2 consists of 20 jobs, two models (A and B), two robots (1 and 2), and four workstations. There are 25 tasks in problem 3, three models (A, B, and C), three robots (1, 2, and 3), and five workstations. There are 40 tasks in problem 4, two models (A and B), three robots (1, 2, and 3), and seven workstations. There are 57 tasks in problem 5, two models (A and B), four robots (1, 2, 3, and 4), and ten workstations. Finally, there are 72 tasks, three models (A, B, and C), and 12 workstations in problem 6.

Two comparisons are done in this work to test the performance of the proposed MBCSA. The MBCSA was compared to the genetic algorithm in the first comparison, and each problem was solved nine times using both techniques.

Table 3.4: Used parameters to solve the numerical example

Parameter	Value
Maximum number of generations	100
Population size	10
Fraction P_a	0.15
max_search	100

Table 3.5: The first and the final populations for the numerical example

	First population			Final Population		
Nests	TEC-PM	TEC-SM	TEC	TEC-PM	TEC-SM	TEC
1	8	0	8	8	0	8
2	8	0	8	8	0	8
3	7.5	0.6	8.1	8	0	8
4	8.5	0.2	8.7	8	0	8
5	8.5	0.75	9.25	8	0	8
6	8	0	8	8	0	8
7	8.5	0.2	8.7	8	0	8
8	8.5	0.95	9.45	8	0	8
9	8.5	0.95	9.45	8	0	8
10	8.5	0.95	9.45	8	0	8

Table 3.6: Problems' specifications

Problem	Tasks	Models	Robots	Workstations
P1	12	A, B	1, 2	3
P2	20	A, B	1, 2	4
P3	25	A, B, C	1, 2, 3	5
P4	40	A, B	1, 2, 3	7
P5	57	A, B	1, 2, 3, 4	10
P6	72	A, B, C	1, 2, 3	12

The MBCSA was compared to another memory-less CSA (MLCSA) identical to the discrete CSA proposed in [24] as the second comparison. For each problem, the MLCSA was run nine times. Table 3.7 shows the parameters utilized to solve each problem using the MBCSA and MLCSA versions. Table 3.8 shows the parameters used in the GA. Table 3.9 displays the findings of the first comparison, including the objective value (total energy consumed), cycle time, and CPU time in seconds.

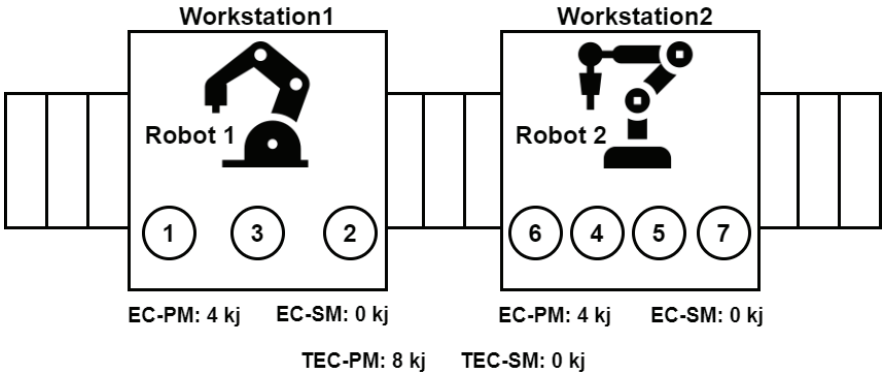

Figure 3.5: Numerical example.

Table 3.7: Used parameters in the MBCSA

Problems	Generations	population's size	Fraction P_a
P1	200	20	0.15
P2	300	30	0.15
P3	300	30	0.15
P4	300	100	0.15
P5	300	100	0.15
P6	300	100	0.15

Beginning with the data for problem 1 obtained using both the MBCSA and the GA 9 times, we can see that both algorithms achieved the same objective value (16.4 kJ) and cycle time (5.5) in all executions. The only difference is that the MBCSA outperforms the GA in terms of CPU time across all executions. For problem 2, both algorithms produced nearly identical objective values. The MBCSA and the GA both obtained the minimum obtained value of 24.3 kJ. Furthermore, in all runs, the MBCSA beats the GA in terms of CPU time. The average of the MBCSA's obtained objective values is 24.69 kJ, whereas the GA's is 24.58 kJ.

For problem 4, the average values acquired by the MBCSA and the GA are 75.71 kJ and 78.27 kJ, respectively, indicating that the MBCSA obtained better results, as shown in the table. The MBCSA obtained the smallest value of 72.61 kJ. In all executions, the MBCSA outperforms the GA in terms of CPU time. The averages of the values obtained by the MBCSA and the GA for problem 5 are 123.70 kJ and 125.17 kJ, respectively. The MBCSA obtained the best value of 121.39 kJ, while the GA obtained 123.19 kJ. In most circumstances, the GA

Table 3.8: Used parameters in the GA

Problems	Generations	population's size	Crossover	Mutation	Elitism
P1	200	20	0.25	0.1	15 %
P2	300	30	0.25	0.1	10 %
P3	300	30	0.25	0.1	10 %
P4	300	100	0.25	0.1	8 %
P5	300	100	0.25	0.1	7 %
P6	300	100	0.25	0.1	7 %

outperforms the MBCSA in terms of CPU time. For problem 6, the GA outperforms the MBCSA in terms of CPU time, but the MBCSA outperforms the GA in terms of objective values, with an average of 164.76 kJ versus 168.18 kJ for the GA. The best values achieved by the MBCSA and the GA are 159.95 kJ and 165.42 kJ, respectively. In Fig 3.6, the obtained objective values for all issues are presented to show the difference between the MBCSA and the GA. We can clearly conclude that the suggested MBCSA outperformed the GA in terms of attaining the lowest energy consumption for major problems 4, 5, and 6. For problems 1, 2, and 3, both algorithms produced nearly identical results.

The details of the best-obtained solutions by both algorithms are shown in table 3.10. The assignment of robots R in workstations W, the processing time PT and idle time ID in each workstation, the energy consumed in both processing mode E(PM) and standby mode E(SM), and the total energy consumed TE. Table **??** shows the objective values obtained by the MBCSA and MLCSA. Nine times, each problem was solved. According to the best-obtained numbers in bold font, the MBCSA outperforms the MLCSA when it comes to solving large-scale problems. These results show the importance of memory integration in supporting the algorithm in obtaining good solutions that would be difficult to locate using the memory-less strategy.

3.7 Conclusion

Finding a suitable configuration for the RMiMAL, in which numerous models are assembled by a group of robots in an intermixed sequence, can save

Table 3.9: All executions of both algorithms for solving all problems.

	MBCSA			GA		
Problem	C	TEC	CPU time (S)	C	TEC	CPU time (S)
P1	5.5	16.4	1.54	5.5	16.4	10.19
	5.5	16.4	2.15	5.5	16.4	9.57
	5.5	16.4	1.81	5.5	16.4	9.32
	5.5	16.4	3.30	5.5	16.4	9.56
	5.5	16.4	3.00	5.5	16.4	9.75
	5.5	16.4	1.36	5.5	16.4	11.60
	5.5	16.4	2.28	5.5	16.4	12.48
	5.5	16.4	1.85	5.5	16.4	9.65
	5.5	16.4	2.03	5.5	16.4	9.18
P2	7	24.65	21.83	7.5	24.55	70.15
	7	24.65	17.91	7	24.65	71.54
	7.5	24.3	23.18	7	24.65	68.21
	7.5	24.3	20.23	7.5	24.55	65.29
	8	24.79	13.19	7	24.35	67.96
	7.5	25.55	27.86	8.5	25.35	67.12
	7.5	24.8	19.99	7.5	24.3	65.23
	7	24.6	21.13	7.5	24.3	76.38
	7.5	24.55	19.50	7.5	24.55	67.42
P3	8.5	41.5	29.55	8.5	41.5	71.07
	8.5	41.3	47.21	9	42.35	77.70
	8.5	41.5	62.97	8.5	41.5	73.24
	8	41.45	33.03	8.5	41.5	77.27
	8	40.2	27.94	8.5	41.35	77.68
	7.5	39.8	28.64	8	40.2	83.83
	8.5	41.5	28.43	8	40.0	79.36
	9.5	43.5	56.18	8	41.5	83.64
	8	40.0	35.9	8.5	41.5	77.30
P4	7.7	75.73	254.35	8	78.06	313.27
	7	72.61	235.50	8	78.21	332.09
	7.5	75.75	171.87	8	78.15	334.41
	8	77.6	204.86	8	79.1	361.2
	7.5	75.4	421.30	8	78.61	355.45
	7.2	73.11	91.02	7.5	77.2	344.04
	7.5	74.45	239.74	8	78.75	325.42
	8	78.75	294.45	8	77.66	348.96
	8	78.01	182.86	8	78.71	351.52
P5	9.2	125.20	790.58	9.1	126.48	996.63
	8.85	121.39	1618.92	8.85	124.45	996.66
	8.95	124.85	1706.37	8	126.97	951.62
	9.2	125.21	1448.19	9.25	126.55	1006.02
	8.95	122.10	1556.07	9	123.98	1165.54
	9.2	125.20	1870.87	9.15	125.48	981.38
	9.1	125.08	1856.95	9.15	125.23	1215.63
	8.95	122.62	1658.89	9.1	124.2	1016.84
	8.9	121.61	2029.61	8.85	123.19	944.15
P6	8.5	166.37	2393.53	8.5	168.936	1331.91
	8.5	167.27	2362.11	8.5	169.18	1305.63
	8.5	168.37	2343.04	8.5	170.02	1264.41
	8.5	166.58	1631.94	8.5	168.90	1246.40
	8.5	165.94	2204.33	8.5	168.58	1421.76
	8	162.47	2364.93	5.5	167.17	1318.00
	8	161.59	2059.96	8.5	165.42	1260.41
	8	159.95	1712.13	8.5	168.44	1356.92
	8	164.3	2255.78	8.5	166.99	1370.88

Table 3.10: Details of the best-obtained solutions by both MBCSA and GA

		MBCSA						GA					
P	W	R	PT	ID	E(PM)	E(SM)	TE	R	PT	ID	E(PM)	E(SM)	TE
P1	1	1	5.5	0	5.5	0	5.5	1	5.5	0	5.5	0	5.5
	2	1	5	0.5	5	0.2	5.2	1	5	0.5	5	0.2	5.2
	3	2	4.5	1	5.4	0.3	5.7	2	4.5	1	5.4	0.3	5.7
P2	1	1	7.5	0	6	0	6	1	7.5	0	6	0	6
	2	1	7	0.5	5.6	0.25	5.85	1	7	0.5	5.6	0.25	5.85
	3	2	6	1.5	6	0.45	6.45	2	6	1.5	6	0.45	6.45
	4	1	7.5	0	6	0	6	1	7.5	0	6	0	6
P3	1	1	6.5	1	7.8	0.5	8.3	1	7.5	0.5	9	0.25	9.25
	2	3	7.5	0	7.5	0.0	7.5	3	8	0	8	0	8
	3	3	7	0.5	7	0.15	7.15	3	7	1	7	0.3	7.3
	4	2	4.5	2.5	6.75	1.80	8.55	2	4.5	3.5	6.75	2.1	8.85
	5	1	6.5	1	7.80	0.5	8.3	3	6	2	6	0.6	6.6
P4	1	2	7	0	10.5	0	10.5	2	7	0.5	10.5	0.3	10.8
	2	1	7	0	10.5	0	10.5	1	5.5	2	8.25	1.4	9.65
	3	2	6.5	0.5	9.75	0.3	10.05	3	7	0.5	11.2	0.3	11.5
	4	3	7	0	11.2	0	11.2	2	7	0.5	10.5	0.3	10.8
	5	2	6.5	0.5	9.75	0.3	10.05	2	7.5	0	11.25	0.0	11.25
	6	1	6.7	0.3	10.05	0.21	10.26	1	7.2	0.3	10.8	0.21	11.01
	7	2	6.5	0.5	9.75	0.3	10.05	2	6.5	1	9.75	0.6	10.35
P5	1	3	8.35	0.5	11.69	0.5	12.19	3	8.2	0.65	11.48	0.65	12.13
	2	3	8.1	0.75	11.34	0.75	12.09	1	7.2	1.65	10.8	1.65	12.45
	3	3	8.7	0.15	12.18	0.15	12.33	3	8.7	0.15	12.18	0.15	12.33
	4	3	8.85	0	12.39	0	12.39	3	8.85	0	12.39	0	12.39
	5	3	8.8	0.05	12.32	0.05	12.37	3	8.5	0.35	11.9	0.35	12.25
	6	3	8.35	0.5	11.69	0.5	12.19	3	8.8	0.05	12.32	0.05	12.37
	7	3	8.8	0.05	12.31	0.05	12.36	3	8.85	0	12.39	0.0	12.39
	8	3	8.65	0.20	12.11	0.2	12.31	3	8.5	0.35	11.90	0.35	12.25
	9	1	8.4	0.45	11.85	0.45	12.3	2	8.75	0.15	12.75	0.11	12.86
	10	3	8.85	0	10.85	0.0	10.85	2	8.85	0	11.76	0.0	11.76
P6	1	3	8	0	14.4	0	14.4	3	8	0.5	14.4	0.5	15
	2	2	7.5	0.5	12.75	0.55	13.3	2	8.5	0	14.45	0	14.45
	3	2	7.5	0.5	12.75	0.55	13.3	2	8.5	0	14.45	0	14.45
	4	2	7.5	0.5	12.75	0.55	13.3	2	8	0.5	13.60	0.5	14.1
	5	2	8	0	13.60	0	13.60	3	7.5	1	13.60	1	14.6
	6	2	8	0	13.60	0	13.60	2	8.5	0	14.45	0	14.45
	7	3	8	0	13.50	0	13.50	2	7.2	1.3	11.90	1.43	13.33
	8	2	7	1	11.90	1.1	13	2	7.5	1	12.75	1.0	13.75
	9	2	7.5	0.5	12.75	0.55	13.3	1	6.75	1.75	10.5	1.75	12.25
	10	1	6.75	1.25	10.5	1.25	11.75	2	8.25	0.25	14.02	0.275	14.295
	11	2	7.75	0.25	13.17	0.275	13.44	2	8.25	0.25	14.02	0.275	14.295
	12	2	7.75	0.25	13.17	0.275	13.44	3	2.67	5.38	4.806	5.83	10.636

Table 3.11: Comparison of best objective values obtained by the MBCSA and the MLCSA

Problems	MBCSA	MLCSA
P1	16.4 kj	16.4 kj
P2	24.3 kj	24.3 kj
P3	39.8 kj	41.35 kj
P4	72.61 kj	72.76 kj
P5	121.39 kj	123.01 kj
P6	159.95 kj	162.83 kj

wasted time and save energy by avoiding the need for the line to be reconfigured at each entrance of a new model. This contribution addresses the energy-efficient RMiMALBP and proposes a memory-based Cuckoo Search Algorithm (MBCSA) to solve it. Searching for good solutions to this complicated problem in a reasonable time is a big challenge, especially for large-sized problems, which is why the memory technique is integrated into the MBCSA to get a compromise between diversification and intensification in the search space while solving (EEMMALBP). Six challenges of varying sizes were constructed to assess the performance of the proposed MBCSA. The MBCSA findings were compared to those of the well-known genetic algorithm and the memory-less CSA. The MBCSA and the GA achieved nearly identical results for issues 1, 2, and 3, but the MBCSA beat the GA in terms of objective values acquired for large-scale problems 4, 5, and 6. The CPU times required by the MBCSA to solve tasks 1, 2, and 3 were longer than those required by the GA. In addition, the MBCSA surpasses the MLCSA when it comes to tackling large-scale issues. When tackling large-scale issues, the suggested MBCSA is a bit sluggish and requires additional CPU time. This flaw is caused by the algorithm's use of memory, which allows it to escape from the local optimum and continue looking for new solutions that were not discovered previously.

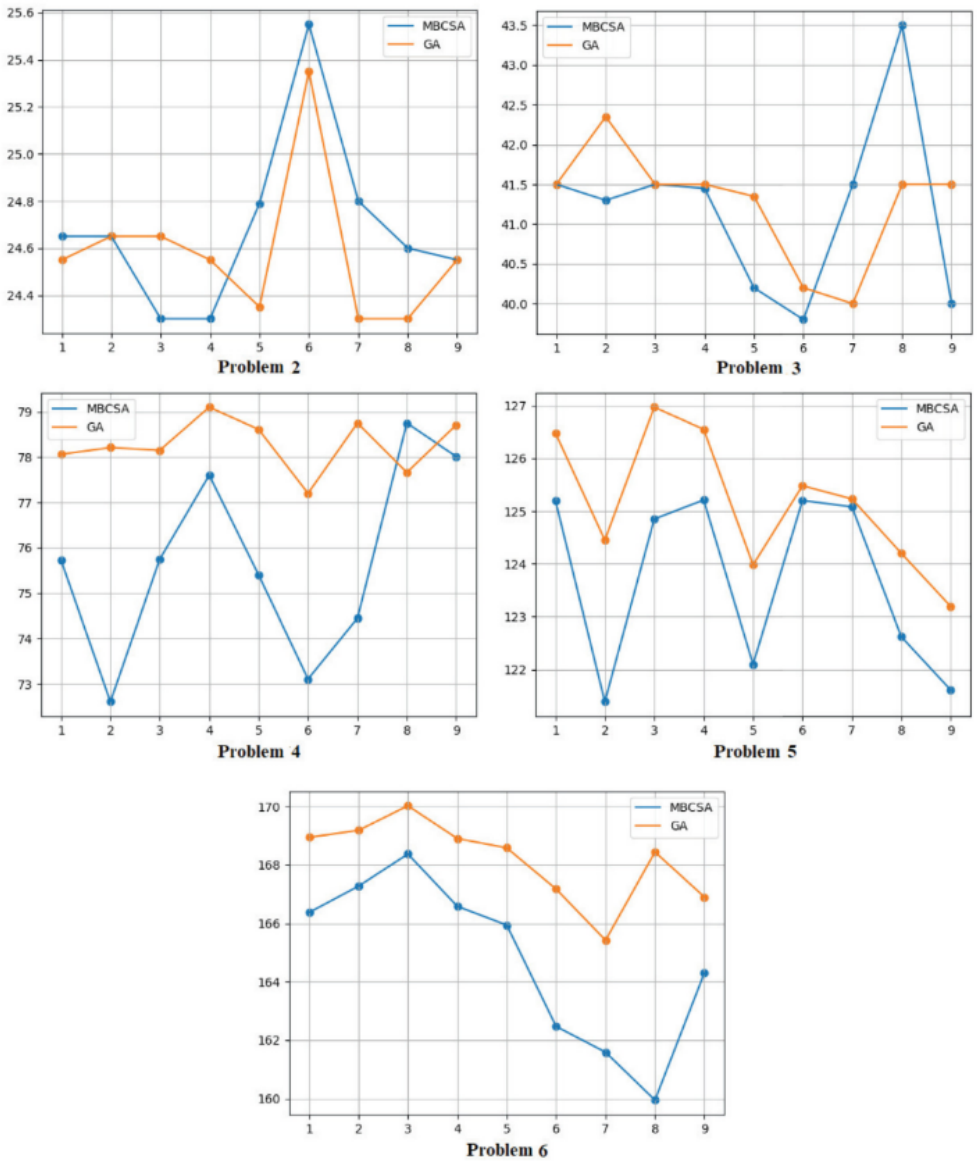

Figure 3.6: Numerical example.

Chapter 4

The Mixed Model Assembly Line Balancing Problem Type 1

4.1 Introduction

The mixed-model assembly line balancing problem consists of finding the best configuration of the assembly line that can assemble more than one model of the same product. This problem has become more interesting in the last few decades due to the increased demands of customers for different models. Type 1 of this problem aims at finding the optimal number of workstations for a fixed cycle time in order to maximize the workload and minimize the line length. To solve this problem, we proposed a Hybrid Reactive Greedy Randomize Adaptive Search Procedure (HRGRASP).

This chapter is organized as follows; First, we describe the mixed-model assembly line balancing problem type 1. Then, we discuss the basic and the reactive GRASP. After that, we present our proposed Hybrid Reactive Greedy Randomized Adaptive Search Procedure. Finally, we discuss our computational results.

4.2 Problem description and mathematical formulation

4.2.1 Problem description

A mixed-model assembly line is created to make M models with similar features, and the demand d_m $(m \in M)$ for each model is known. The sum of all requests that must be fulfilled in a certain period of time, or PT, is the total demand, or D_m. To convert the MiMALBP to SALBP, each model m has a precedence graph G, and all graphs can be joined into a single graph. The sum of all processing durations for tasks allocated to the same workstation cannot exceed the cycle time, which is determined using Equation 1. Each task I in the combined graph must be assigned to a single workstation.

According to the characteristics of the fundamental form of the assembly line balancing problem, authors adopted the following assumptions for the MiMALBP-I they are addressing:

1. Task processing times are predictable.
2. The cycle time is predetermined and known.
3. The assembly line is straight.
4. All tasks must be assigned.
5. Common tasks between models are assigned to the same workstation
6. Processing time of common tasks may differ from model to another one.
7. Task assignments are constrained by precedence rules.

4.2.2 Mathematical formulation

i task
N number of tasks
t_i processing time of task i
C Cycle time
w workstation
W_{max} upper bound on the number of workstation

P_i the set of predecessors of task i
X_{iw} equal to 1 if task i is assigned to workstation w, 0 otherwise
y_{iw} equal to 1 if any task is assigned to workstation w, 0 otherwise

The upper bound is derived from the literature to specify the maximum number of workstations and to restrict the search space. The number of tasks is the upper constraint in this research, hence $Wmax = UB = N$ denotes that each workstation will, in the worst scenario, only do one task. Equation 2 provides the solution to the problem's objective function.

$$Minimize \sum_{w=1}^{W_{max}} y_w \tag{4.1}$$

Under the following constraints:

$$\sum_{w=1}^{W_{max}} X_{iw} = 1 \quad for \quad i = 1, 2, ..., N \tag{4.2}$$

$$\sum_{i=1}^{N} t_i * X_{iw} \leq C \quad for \quad w = 1, 2, ..., W_{max} \tag{4.3}$$

$$\sum_{w=1}^{W_{max}} w * X_{hw} \leq \sum_{w=1}^{W_{max}} w * X_{iw} \quad where\ h \in P_i \tag{4.4}$$

$$y_{iw} \in \{0, 1\} \tag{4.5}$$

$$X_{iw} \in \{0, 1\} \tag{4.6}$$

Constraint (4.2) ensures that each task is only assigned to one workstation; constraint (4.3) ensures that the total processing time of tasks assigned to the same workstation does not exceed the cycle time; constraint (4.4) imposes the precedence relations between tasks; constraint (4.5) defines the possible values of y_w; and constraint (4.6) defines the possible values of X_{iw}.

4.3 Basic and Reactive GRASP

Many combinatorial problems have been resolved using the greedy randomized adaptive search process, a multi-start meta-heuristic. Each iteration of the GRASP employs the construction phase and the local search phase. The building phase is used to build a workable solution, but necessarily the best one. The local search phase uses a local search process to identify a better solution based on the constructed one. In the building phase, two lists are employed. The candidate list (CL) comprises all candidates that can be included in the partial solution, and from this list, the best candidates are chosen to produce the limited candidate list (RCL). In the fundamental GRASP, the best candidates are chosen from the CL depending on the fixed parameter. The threshold value is represented by the following equation, which makes use of the alpha (α) parameter:

$$TC_{th} = T_{min} + \alpha(T_{max} - T_{min}) \tag{4.7}$$

where the incremental costs T_{min} and T_{max} are, respectively, the minimum and maximum. This is the greedy aspect of the GRASP: the RCL is formed by choosing all candidates whose costs are less than or equal to the threshold value. Selecting a candidate at random from the RCL to be added to the partial solution is where the random element comes into play [201, 202].

The basic GRASP's drawback is that it cannot learn from previous iterations because all information about obtained solutions is discarded. Additionally, in some circumstances, using a fixed value for the parameter may not be able to assist the GRASP in convergent toward a global optimum [201]. The first improvement to the basic GRASP is the reactive GRASP. It was first proposed in [203] for the time slot assignment problem and has since been applied to a number of optimization problems, including the Strip Packing problem [204], Capacitated Clustering problem [205], and the Vehicle Routing Problem [206]. IInstead of using a fixed value in each iteration of the reactive GRASP algorithm, the α parameter is chosen at random from a defined range of potential values $Alpha = \{\alpha_1, ..., \alpha m\}$ employing probabilities *Pi, i=1,...,m*. These probability are based on previously discovered solutions. The probability of

selection is *Pi=1/m* for all feasible values of α during the initial GRASP iteration where m is the number of possible α values. At any subsequent iteration Let $\hat{Z}$ be the best solution found, And let A_i be the average value of all solutions found using $\alpha=\alpha_i$ where *i=1,...,m*. The selection probabilities are updated periodically using the following equation :

$$P_i = \frac{q_i}{\sum_{j=1}^{m} q_j} \tag{4.8}$$

where $q_i = \hat{Z}/A_i$, *i=1,...,m*. The value of q_i is increased if the values of $\alpha=\alpha_i$ lead to the best solutions on average. The probabilities of appropriate values will then increase when they are updated [202].

4.4 The proposed Hybrid Reactive Greedy Randomized Adaptive Search Procedure

The suggested hybrid Reactive GRASP in this study is based on the structure of the original Reactive GRASP, and we integrated some modifications in the construction phase and the local search phase to adapt it to handle the mixed-model assembly line balance problem type-I. The Hybrid Reactive GRASP is implemented using four main steps: building, local search, evaluation of the found solution, and updating selection probabilities at the end of each session. For the following reasons, the authors of this study decided to solve the MiMALBP-I using the Reactive GRASP. In compared to other meta-heuristics that require more steps and intricate calculations, the Reactive GRASP is straightforward. Second, by employing a different heuristic or meta-heuristic, we can easily make a hybridization in any GRASP step. The GRASP has reportedly been utilized to solve a number of optimization issues, and the outcomes have demonstrated that it is an effective meta-heuristic.

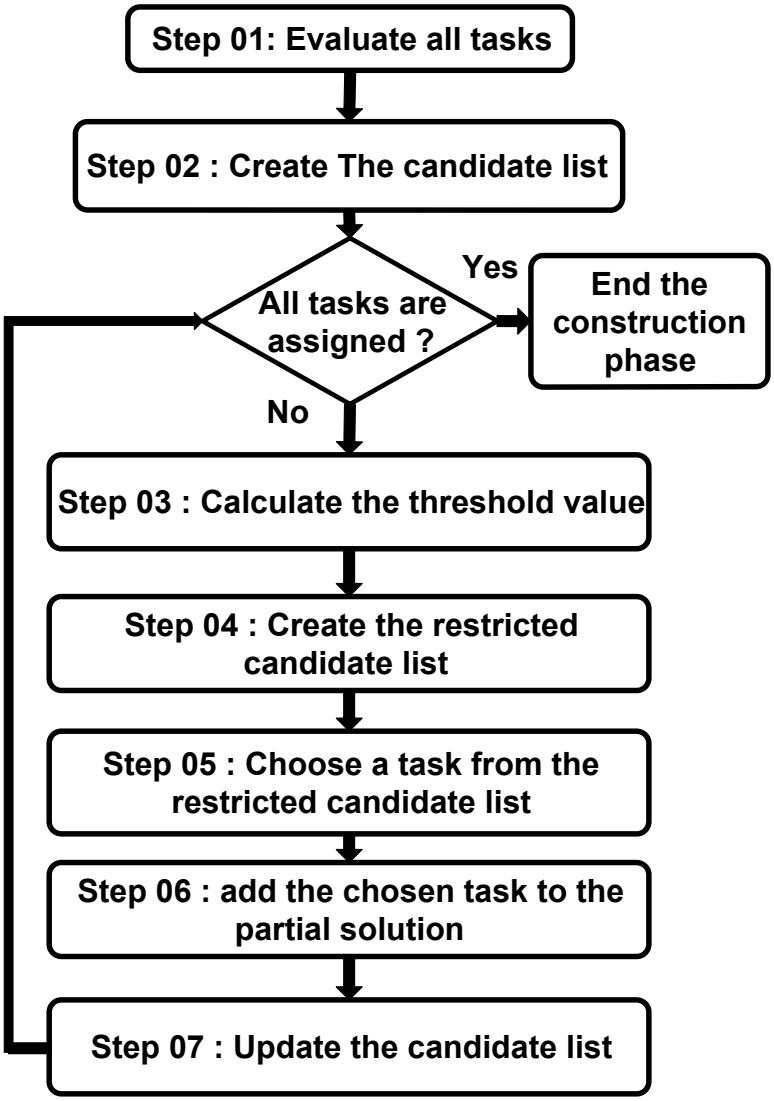

Figure 4.1: The proposed construction phase.

4.4.1 The construction phase

The construction phase is where the initial solution is built, and it's at this phase that a greedy heuristic is employed to pick the best element first, depending on cost, which varies from problem to problem. In the proposed hybrid Reactive GRASP, task processing times are taken into account as costs, and in the construction phase, the Shortest Processing Time heuristic is used as a greedy heuristic. As a result, tasks with the shortest processing times are viewed as the best elements. This approach is known as Hybrid Reactive GRASP-SPT because it combines the Reactive GRASP and the Shortest

Processing Time heuristic. The adopted construction phase's flowchart is displayed in figure 4.1.

Figure 4.1 illustrates the seven processes that make up the proposed construction phase.

1. All tasks are evaluated by their processing times.
2. To create the candidate list, assignable tasks are selected from the list of tasks that are available. A work that has no predecessors or whose predecessors have all been assigned is considered assignable.
3. Equation 8 from Section 4 is used to calculate the threshold value based on the processing times of the jobs that are present in the candidate list.
4. Each task is selected to be on the restricted candidate list if its processing time is less than or equal to the threshold value (RCL).
5. A task is randomly selected from the restricted candidate list.
6. Add the selected task to the partial solution.
7. Remove the assigned task from the candidate list and add new candidates to update it (assignable tasks).

4.4.2 The local search phase

The solution that was reached utilizing the construction phase might not be the best one, so a local search is used to see whether a neighboring solution with a higher objective value exists. In the proposed reactive GRASP-SPT, we employ a straightforward local search that seeks out neighbor solutions for a fixed *max_search* number, and to search locally, two tasks that are not connected by a precedence relation are swapped at random. If tasks i and j are randomly selected to be swapped, and task i is placed before task j in the sequence, the algorithm first checks to see if the new solution will adhere to the precedence relations.

The local search phase is based on four main processes, as indicated in Figure 4.2:

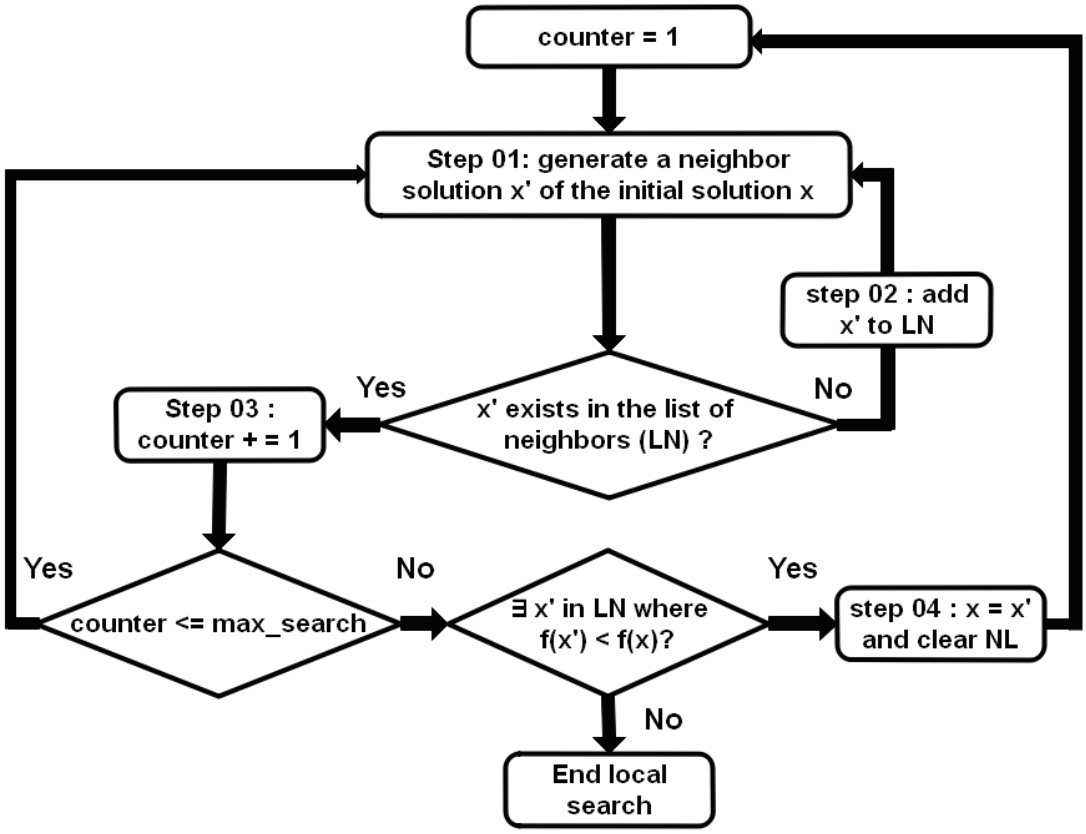

Figure 4.2: The proposed local search method.

1. After the counter is initialized, the random swap function is used to create a neighbor solution. Go to step 2 if the neighbor has not been located in earlier iterations; otherwise, move to step 3.
2. A neighbor list is updated to include the generated neighbor (LN).
3. If the generated neighbor is already present in the list of neighbors, the counter is increased by one. Then, if the counter is less than or equal to the max search number, go to step 1 to start looking for a new neighbor. If, however, a neighbor with a better objective value is found in the list of neighbors, go to step 4, otherwise the local search should be stopped.
4. Switch the first solution out for the new, better neighbor solution and reset the counter to 1.

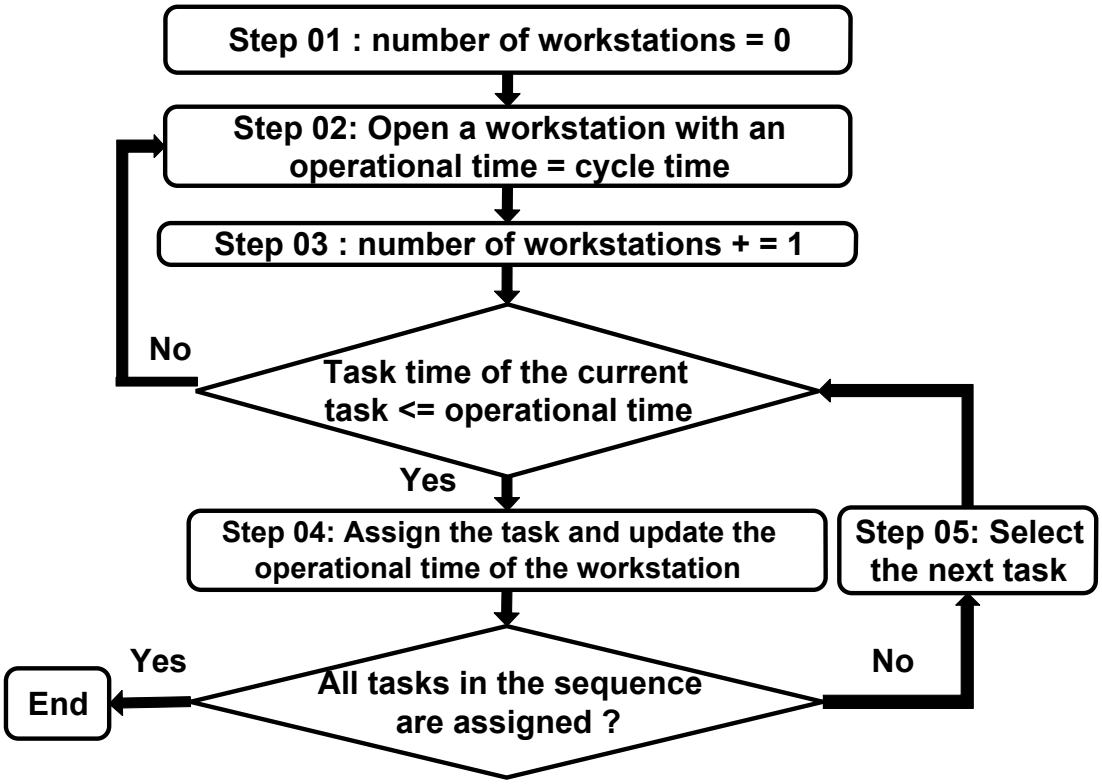

Figure 4.3: The proposed evaluation method.

4.4.3 Evaluation of solutions

The algorithm compares the new solution found during the local search phase to the solution previously selected as the best solution found; if the new solution has a better objective value (provides a minimum number of workstations), it is then considered the best solution found and is passed to the subsequent iteration.

The evaluation of solutions is based on the five processes depicted in Figure 4.3:

1. Initialize the number of workstations to 0 in Step 1.
2. Create a new workstation and set its operational time with the fixed known cycle time.
3. Add 1 to the number of workstations after each opening procedure.
4. If the selected task's processing time (task time) is less than or equal to

the operational time of the current workstation, assign it to that workstation, and update the operational time by deducting the assigned task's processing time from it. If not, proceed to step 02 to open a new workstation.

5. If there are still unassigned tasks, choose the next one and continue the process; otherwise, the evaluation process will end.

4.4.4 Updating of selection probabilities

The number of solutions found and the sum of the achieved objective values are calculated for each alpha value in each iteration by the reactive GRASP. With this knowledge, the algorithm uses Equation 9 from Section 4 to update the selection probabilities for each alpha value for each period. This learning mechanism aids the algorithm in determining which alpha value has the most likelihood of being chosen from the selection probabilities and, as a result, produces effective solutions.

4.5 Computational results

On a personal computer with an Intel Core i3-4005 U CPU, 1.70 GHz, and 6GB memory, the proposed hybrid Reactive GRASP is constructed using Python. To test its performance, 6 types of problems have been produced at random. A small size problem with two models A and B, ten tasks, and ten precedence relations is shown in table 4.1. Model A requires 14 time units of operational time in total, whereas Model B requires 12. There are 24 demands for model A and 22 for model B, for a total of 46. We get a cycle time of = 4.1 by dividing the specified time period by the total number of demands. Two models A and B must be put together in a straight line in the second small-size problem as shown in table 4.2, which also has a total of 12 tasks and a total of 14 precedence relations. Model A takes 17 time unit, whereas Model B takes 14 seconds. A total of 60 demands 30 each for models A and B are presented. With the overall demands and the allotted time (PT = 300), the obtained cycle time is 5, which is the given number. Table 4.3 shows a medium-sized MiMALB problem with two models, A and B, 15 tasks, and a total of 15 precedence relations. Model A requires 16 time units to be completed, whereas Model B requires 20. 25 units

of model A and 15 units of model B must be put together in a period of time equal to 150, resulting in a cycle time of 4.3.

Table 4.1: Problem 1.

Task	Model A	Model B	Task time	Predecessors
1	1.0	1.0	1.0	-
2	2.0	0.0	2.0	-
3	0.0	2.0	2.0	-
4	3.0	1.0	2.0	1, 2
5	2.0	2.0	2.0	-
6	1.0	1.0	1.0	3, 5
7	2.0	0.0	2.0	4
8	0.0	2.0	2.0	4
9	1.0	1.0	1.0	6
10	2.0	2.0	2.0	7, 8, 9
Demands of models : DA = 24, DB =22				
Period time = 190				
Cycle time = 190/46 = 4.1				

Table 4.2: Problem 2.

Task	Model A	Model B	Task time	Predecessors
1	1.0	1.0	1.0	-
2	1.0	1.0	1.0	-
3	1.0	1.0	1.0	-
4	2.0	2.0	2.0	1
5	3.0	1.0	2.0	2
6	1.0	3.0	2.0	3
7	0.0	1.0	1.0	4
8	2.0	0.0	2.0	4
9	0.0	1.0	1.0	5, 6
10	2.0	2.0	2.0	7, 8, 5
11	3.0	0.0	3.0	6
12	1.0	1.0	1.0	9, 10, 11
Demands of models : DA = 30, DB =30				
Period time = 300				
Cycle time = 300/60 = 5				

The second medium-sized problem (Table 4.4) contains three models (A, B, and C), each of which must be completed in 21, 19, and 20 units of time,

Table 4.3: Problem 3.

Task	Model A	Model B	Task time	Predecessors
1	1.0	2.0	1.0	-
2	2.0	0.0	1.0	-
3	2.0	0.0	1.0	-
4	0.0	1.0	2.0	-
5	0.0	3.0	2.0	-
6	0.0	1.0	2.0	-
7	2.0	1.0	1.0	1, 2, 4
8	1.0	2.0	2.0	3, 5, 6
9	3.0	2.0	1.0	7
10	1.0	1.0	2.0	8
11	0.0	1.0	3.0	10
12	1.0	1.0	1.0	9, 11
13	0.0	2.0	1.0	12
14	1.0	1.0	1.0	12
15	2.0	2.0	1.0	13, 14
Demands of models : DA = 20, DB =15				
Period time = 150				
Cycle time = 150/35 = 4.3				

respectively. There are 15 tasks total, and there are overall 17 precedence relations. There are 38 demands for model C, 35 demands for model A, and 35 demands for model B. Based on the problem specifications (PT = 480, Total number of demands = 108), a cycle time of 4.4 was computed. A large-scale problem with three models A, B, and C and 22 tasks connected by 25 precedence connections is shown in Table 4.5. Operational times for models A and B are 27, 31, and 30, respectively. There are 25, 35, and 25 requests for Model A, Model B, and Model C, respectively. These demands have a time requirement of 480, which calls for a cycle time of 5.6. There are four models (A, B, C, and D) in the second large-size problem Table 4.6 that can be assembled in the same line. There are 25 tasks in total, with a total of 41 precedence relations. Models A, B, D, and C require 22, 25, 26, and 26 units of time, respectively, to complete. There are 40 requests for model A, 15 for model B, 15 for model C, and 20 for model D. With a period duration of 700 and a cycle time of 7.8, all of these requirements must be met.

The extremely large problem in Table 4.7 has two models, A and B. 44

tasks are included, and there are 45 precedence relationships. 66 time units are required to complete model A, while 63 time units are needed to complete model B. 50 demands for model A, compared to 45 for model B, for a total of 95. The given period time is 500 time units, and the cycle time is 5.26 when this number is divided by the total number of demands. This problem is utilized to compare the performance of the LINGO solver and the proposed hybrid Reactive GRASP. Table 4.8 lists all the parameters that were used to address each issue utilizing the hybrid reactive GRASP.

Table 4.4: Problem 4.

Task	Model A	Model B	Model C	Task time	Predecessors
1	1.0	1.0	1.0	1.0	-
2	0.0	0.0	3.0	3.0	-
3	0.0	2.0	0.0	2.0	-
4	2.0	0.0	1.0	1.5	-
5	3.0	3.0	3.0	3.0	1
6	1.0	1.0	1.0	1.0	2, 3, 4
7	2.0	2.0	2.0	2.0	5, 6
8	1.0	1.0	0.0	1.0	-
9	2.0	2.0	1.0	1.7	4, 8
10	1.0	0.0	2.0	1.5	6, 9
11	2.0	2.0	2.0	2.0	7
12	2.0	2.0	2.0	2.0	10
13	3.0	1.0	1.0	1.3	11, 12
14	0.0	1.0	0.0	1.0	12
15	1.0	1.0	1.0	1.0	13, 14
Demands of models : DA = 35, DB =35, DC =38					
Period time = 480					
Cycle time = 480/108 = 4.4					

Comparisons are made between the hybrid reactive GRASP's findings and those from the standard GRASP, which employs a fixed alpha value throughout all iterations. We also suggest another method based on three heuristics. In [207] and [208], the assembly line balance problem was solved using the fundamental GRASP. We refer to the suggested method as RWP-NS-LT since it is based on three heuristics from [209]: the ranked positional weight, the greatest number of successors, and the longest processing time. According to the

Table 4.5: Problem 5.

Task	Model A	Model B	Model C	Task time	Predecessors
1	1.0	1.0	2.0	1.3	-
2	2.0	1.0	2.0	1.7	-
3	2.0	1.0	1.0	1.3	1, 2
4	1.0	0.0	0.0	1.0	-
5	0.0	2.0	1.0	1.5	-
6	2.0	3.0	3.0	2.7	-
7	1.0	1.0	0.0	1.0	-
8	3.0	4.0	1.0	2.7	4, 5, 6, 7
9	1.0	0.0	0.0	1.0	3
10	0.0	1.0	4.0	2.5	3
11	1.0	2.0	3.0	2.0	8
12	2.0	4.0	4.0	3.3	9, 10, 11
13	3.0	0.0	2.0	2.5	-
14	1.0	2.0	1.0	1.3	13
15	3.0	2.0	0.0	2.5	14
16	0.0	3.0	2.0	2.5	15
17	0.0	0.0	1.0	1.0	12
18	1.0	0.0	0.0	1.0	12
19	0.0	1.0	0.0	1.0	16
20	2.0	0.0	1.0	1.5	16
21	0.0	1.0	0.0	1.0	12
22	1.0	2.0	2.0	1.7	17,18,19, 20,21
Demands of models : DA = 25, DB =35, DC =25					
Period time = 480					
Cycle time = 480/85 = 5.6					

suggested method, the task allocated to the workstation with the highest importance is the one with the most positional weight. When two tasks have the same positional weight, their successor counts are compared, and the task with the most successors is given top priority. If both tasks have the same number of successors, their processing times are also compared, and the task with the longest processing time receives the highest priority. In the event that two jobs have the same processing time, the RWP-NS-LT finally selects one at random. In the fundamental GRASP, the fixed alpha value utilized for each presented problem is 0.

The numbers of solutions found by each alpha value used to solve each

Table 4.6: Problem 6.

Task	Model A	Model B	Model C	Model D	Task time	Predecessors
1	1.0	1.0	1.0	1.0	1.0	-
2	0.0	1.0	0.0	0.0	1.0	-
3	0.0	2.0	2.0	0.0	2.0	-
4	2.0	0.0	1.0	0.0	1.5	-
5	0.0	0.0	0.0	1.0	1.0	-
6	0.0	0.0	0.0	1.0	1.0	-
7	3.0	2.0	0.0	1.0	2.0	-
8	2.0	3.0	2.0	2.0	2.25	1
9	0.0	0.0	2.0	0.0	2.0	4
10	1.0	0.0	0.0	3.0	2.0	4, 5
11	1.0	0.0	0.0	2.0	1.5	4, 6, 7
12	0.0	0.0	1.0	0.0	1.0	3
13	1.0	1.0	0.0	0.0	1.0	2, 7
14	0.0	1.0	0.0	0.0	1.0	3, 7
15	0.0	0.0	3.0	0.0	3.0	3
16	2.0	0.0	3.0	0.0	2.5	8, 9, 13
17	2.0	0.0	5.0	5.0	4.0	8,10,11,12, 13
18	4.0	0.0	0.0	4.0	4.0	11
19	0.0	3.0	1.0	3.0	2.3	8,11,12,13, 15
20	1.0	0.0	1.0	1.0	1.0	16,17,18
21	0.0	4.0	2.0	0.0	3.0	14,17,19
22	0.0	5.0	0.0	0.0	5.0	21
23	1.0	0.0	0.0	0.0	1.0	17
24	0.0	0.0	0.0	1.0	1.0	18,19
25	1.0	2.0	2.0	2.0	1.75	20,22,23,24
Demands of models : DA = 40, DB =15, DC =15, DD = 20						
Period time = 700						
Cycle time = 700/90 = 7.8						

problem are shown in 4.9. In the majority of cases, when α =1, the search space is expanded and the algorithm can identify more solutions. Some of the solutions found by different alpha values are similar, and some of the solutions found by α=1 cannot be found by α=0 or α=0.5, but as we can see, the number of solutions found by α=0 in solving the large size problem 6 is higher than numbers found by α=0.45 and α=1, so we can conclude that when the alpha value increases the algorithm can find more solutions. The probability of selection for each alpha throughout the solution of the small size problem 1 is

Table 4.7: Problem 7.

Task	Model A	Model B	Task time	Predecessors
1	1.0	1.0	1.0	-
2	2.0	2.0	2.0	-
3	1.0	0.0	1.0	-
4	1.0	1.0	1.0	-
5	1.0	1.0	1.0	-
6	0.0	1.0	1.0	-
7	2.0	2.0	2.0	1
8	2.0	2.0	2.0	2
9	3.0	0.0	3.0	3
10	2.0	2.0	2.0	4
11	1.0	1.0	1.0	5
12	0.0	1.0	1.0	6
13	1.0	1.0	1.0	7, 8
14	1.0	0.0	1.0	9
15	4.0	2.0	3.0	10
16	3.0	3.0	3.0	11, 12
17	2.0	4.0	3.0	13
18	1.0	0.0	1.0	14
19	3.0	5.0	4.0	15
20	1.0	1.0	1.0	16
21	1.0	0.0	1.0	17
22	0.0	2.0	2.0	17
23	2.0	0.0	2.0	18
24	1.0	1.0	1.0	19, 20
25	1.0	0.0	1.0	21
26	0.0	2.0	2.0	22
27	1.0	0.0	1.0	23
28	1.0	1.0	1.0	24
29	1.0	1.0	1.0	25, 26
30	3.0	0.0	3.0	27
31	1.0	1.0	1.0	28
32	1.0	5.0	3.0	29
33	2.0	2.0	2.0	30, 31
34	1.0	1.0	1.0	32
35	3.0	1.0	2.0	33
36	0.0	4.0	4.0	34
37	3.0	0.0	3.0	34
38	1.0	1.0	1.0	35
39	0.0	1.0	1.0	36
40	1.0	0.0	1.0	37
41	2.0	2.0	2.0	38
42	2.0	2.0	2.0	39, 40
43	1.0	1.0	1.0	41
44	5.0	5.0	5.0	42, 43

Demands of models : DA = 50, DB =45
Period time = 500
Cycle time = 500/95 = 5.26

shown in Figure 4.4, and since there was no change from the first to the last period, all alpha values have an equal likelihood of obtaining solutions with the same objective values. Figures 4.5 and 4.6 demonstrate that, when solving medium-sized problem 3 and small-sized problem 2, the selection probabilities of α=0 and α=0.5 have the same variation over all periods as compared to α=1, and they increase. This indicates that when alpha takes 0 and 0.5 as values, the algorithm can find good solutions that have the same objective values (numbers of workstations).

We can see that the variations of the selection probabilities of the α=0 and

Table 4.8: Used parameters in the hybrid reactive GRASP for all problems

	Max iterations	Max search	period	alpha values
small size problem 1	1000	30	100	0, 0.5, 1
small size problem 2	1000	30	100	0, 0.5, 1
medium size problem 3	1500	40	100	0, 0.5, 1
medium size problem 4	1500	40	100	0, 0.75, 1
large size problem 5	2000	40	100	0, 0.75, 1
large size problem 6	2000	40	100	0, 0.45, 1

α=0.75 values are similar and improved over the probability of the α=1 from the second to the last period in Figures 4.7 and 4.8 respectively, which show the variation of the used alpha values while solving medium-size problem 4 and large-size problem 5. Figure 4.9 demonstrates that, when compared to α=0.45 and α=1, the probability of selection of α=0 is the highest, indicating that 0 is the optimum alpha value for the algorithm to use in order to identify the best solutions for the large-scale problem 6.

Figures 4.10 and 4.11 display six comparisons of the number of workstations that each algorithm was able to generate after solving the given challenges. When problems 1 and 6 were solved, all algorithms yielded the same number of workstations (wr = 5 for problem 1 and wr = 7 for problem 6), as shown in comparisons (a) and (f). Comparisons (b) and (d) demonstrate that, when compared to the proposed RWP-NS-LT, the hybrid Reactive GRASP and the basic GRASP perform better in solving tasks 2 and 4, respectively. After resolving difficulties 3 and 5, comparisons (c) and (e) reveal that only the hybrid Reactive GRASP, when compared to the basic GRASP and the suggested RWP-NS-LT, finds the optimal values. We can therefore deduce from all comparisons that the suggested hybrid Reactive GRASP had an advantage over the basic GRASP and the proposed RWP-NS-LT in that it was able to find the best solutions for all size problems. Tables (4.10, 4.11, 4.12, 4.13, 4.14, 4.15) illustrate found solutions of the hybrid reactive GRASP for all problems.

Table 4.16 compares the CPU time required to solve problem 7 by the proposed Hybrid Reactive GRASP and the LINGO solver. We can easily deduce from the data analysis for all tries that the Hybrid Reactive GRASP can locate the ideal number of workstations (w = 16) faster than the LINGO solver.

Table 4.9: Numbers of solutions found by the hybrid Reactive GRASP for each problem

small size problem 1	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$	total number of solutions
	56	58	247	361
small size problem 2	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$	total number of solutions
	84	194	310	588
medium size problem 3	$\alpha = 0$	$\alpha = 0.5$	$\alpha = 1$	total number of solutions
	111	106	494	711
medium size problem 4	$\alpha = 0$	$\alpha = 0.75$	$\alpha = 1$	total number of solutions
	8	128	469	605
large size problem 5	$\alpha = 0$	$\alpha = 0.75$	$\alpha = 1$	total number of solutions
	48	646	671	1365
large size problem 6	$\alpha = 0$	$\alpha = 0.45$	$\alpha = 1$	total number of solutions
	689	652	624	1965

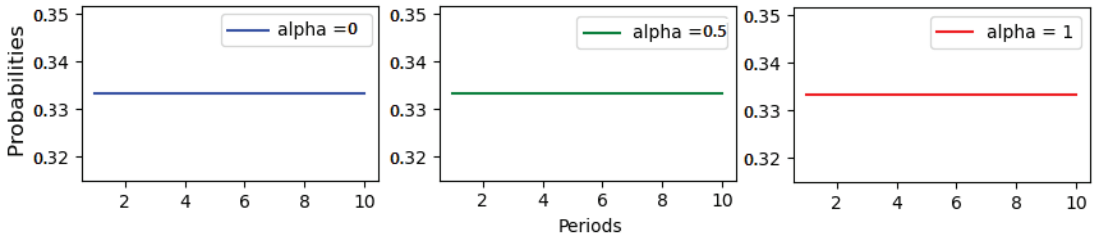

Figure 4.4: Variation of selection probabilities while solving problem 1.

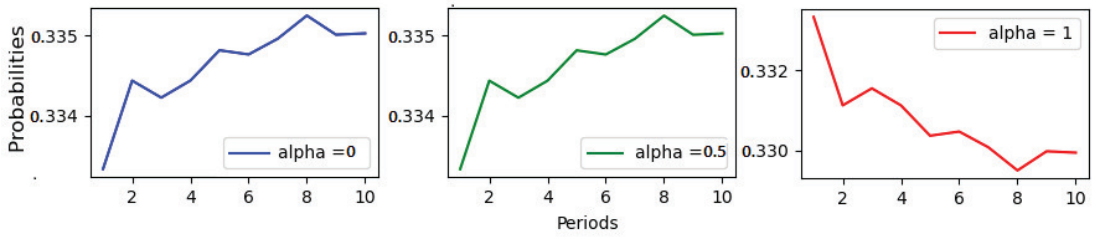

Figure 4.5: Variation of selection probabilities while solving problem 2.

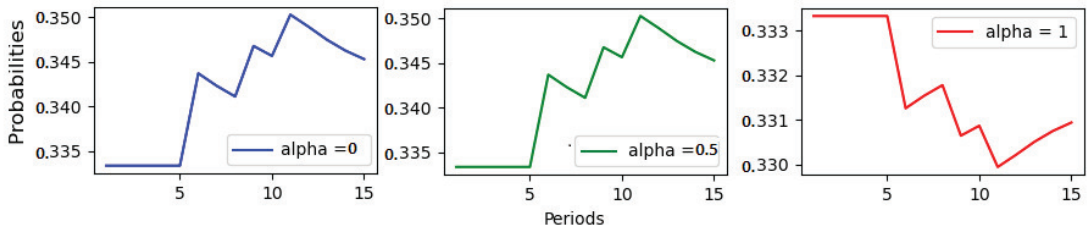

Figure 4.6: Variation of selection probabilities while solving problem 3.

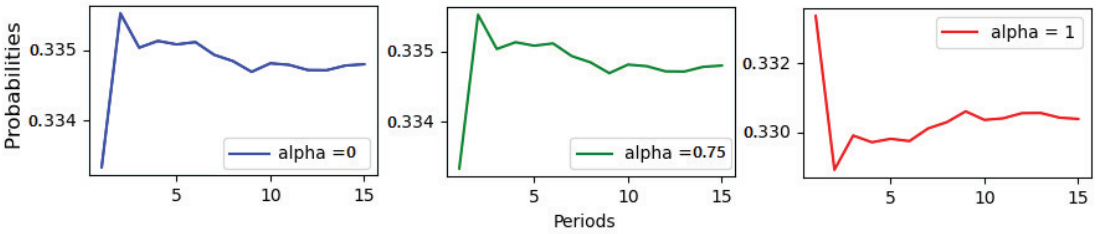

Figure 4.7: Variation of selection probabilities while solving problem 4.

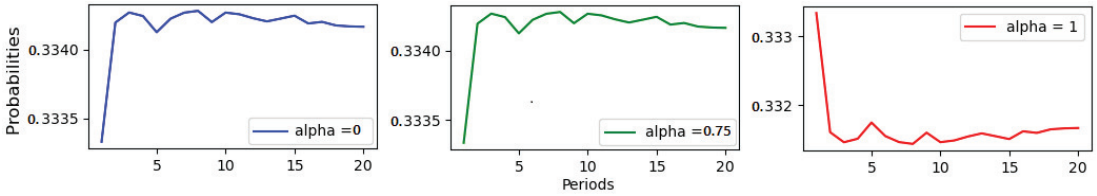

Figure 4.8: Variation of selection probabilities while solving problem 5.

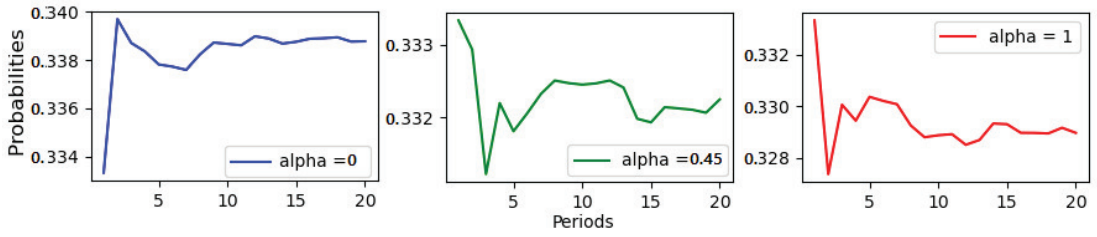

Figure 4.9: Variation of selection probabilities while solving problem 6.

4.6 Conclusion

In this chapter, we presented our contribution to solve MiMALBP type 1. The mixed-model assembly line is intended to create many models of a single product in order to satisfy customer needs on time, and determining the appropriate number of workstations to optimize workload at each workstation for each model is a challenging task. A hybrid reactive GRASP meta-heuristic

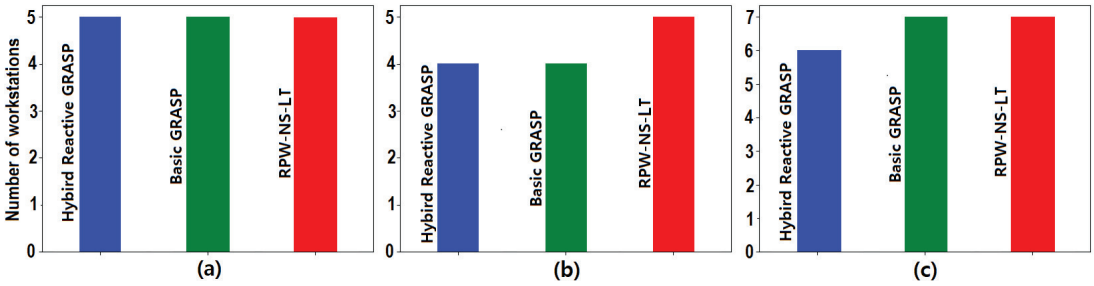

Figure 4.10: Obtained number of workstations for problem 1 (a), problem 2 (b), and problem 3 (c).

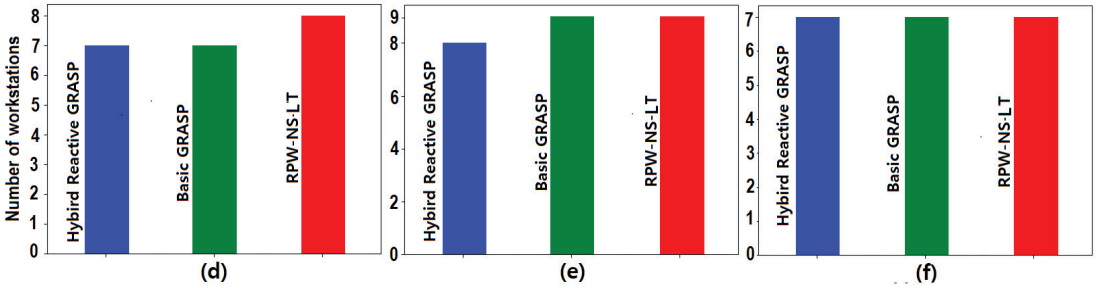

Figure 4.11: Obtained number of workstations for problem 4 (d), problem 5 (e), and problem 6 (f)

was presented in this contribution to solve the straight mixed-model assembly line balance problem while decreasing the number of workstations. In the construction phase, the shortest processing time heuristic is used to identify the priority rule and assess jobs based on their processing times, and in the local search phase of the algorithm, a basic local search approach that seeks for a better neighbor solution is utilized. The hybrid reactive GRASP discovered optimum solutions for all provided issues after addressing all proposed problems utilizing the hybrid reactive GRASP, the basic reactive GRASP, and the RPW-SN-LT. The hybrid reactive GRASP findings further demonstrate that utilizing a set of alpha values makes the algorithm more efficient by discovering more solutions than using a fixed alpha value and allows it to avoid the local optimum trap. Furthermore, when compared to the LINGO solver, the suggested reactive GRASP took less time to solve the extremely big issue.

Table 4.10: Assignment of tasks of problem 1.

Wr1	Wr2	Wr3	Wr4	Wr5
1, 3	5, 6, 9	2, 4	7, 8	10

Table 4.11: Assignment of tasks of problem 2.

Wr1	Wr2	Wr3	Wr4
3, 2, 1, 6	5, 9,4	7, 8, 10	11, 12

Table 4.12: Assignment of tasks of problem 3.

Wr1	Wr2	Wr3	Wr4	Wr5	Wr6
3, 2, 1	4, 7, 9	5, 6	8, 10	11, 12	13, 14, 15

Table 4.13: Assignment of tasks of problem 4.

Wr1	Wr2	Wr3	Wr4	Wr5	Wr6	Wr7
8, 4, 1	2	3, 9	5, 6	7, 11	10, 12	14, 13, 15

Table 4.14: Assignment of tasks of problem 5.

Wr1	Wr2	Wr3	Wr4	Wr5	Wr6	Wr7	Wr8
2,7, 5,1	3,4, 9	13,14	15,10	16,19, 20	6,8	11,12	17,18, 21,22

Table 4.15: Assignment of tasks of problem 6.

Wr1	Wr2	Wr3	Wr4	Wr5	Wr6	Wr7
6,1,5, 2,4,10	3,12,9, 7	14,13,11, 8	16,15,19	17,21	23,18,24, 20	22,25

Table 4.16: Comparison of taken CPU time for solving problem 7 by the Hybrid Reactive GRASP and LINGO solver.

Attempts	Hybrid Reactive GRASP (CPU / Workstations)	LINGO solver (CPU / Workstations)
Attempt 1	10 min : 18 sec / 16	29 min : 46 sec / 16
Attempt 2	14 min : 21 sec / 16	21 min : 41 sec / 16
Attempt 3	16 min : 47 sec / 16	45 min : 09 sec / 16
Attempt 4	10 min : 22 sec / 16	28 min : 41 sec / 16
Attempt 5	10 min : 27 sec / 16	37 min : 31 sec / 16
Attempt 6	10 min : 14 sec / 16	23 min : 50 sec / 16

Chapter 5

A Hybrid Grasp-genetic Algorithm for Mixed-model Assembly Line Balancing Problem Type 2

5.1 Introduction

The mixed model assembly line balancing problem type 2 aims at finding the minimal cycle time for a fixed number of workstations. By minimizing the total cycle time of the line, we can reach an important production rate. This problem became more interesting in the last decades due to the high demands of customers for a variety of models of the same product. To solve this problem, we proposed a hybrid approach that combines two meta-heuristics, the greedy randomized adaptive search procedure (GRASP), and the famous genetic algorithm (GA). In order to build initial solutions, the ranked positional weight (RPW) was used in the construction phase of the GRASP. Initial solutions are then enhanced in the local search phase using a neighborhood search procedure.

This chapter is organized as follows; first, we describe the mixed-model assembly line balancing problem type 2. Then, we discuss the proposed algorithms including the genetic algorithm and the greedy randomized adaptive search procedure. After that, we present a numerical example. Finally, we discuss the obtained results.

5.2 Problem description and mathematical formulation

5.2.1 Problem description

Similar models are put together in the mixed model assembly line in a predetermined random order. Each model has a set of tasks, and each task has a specified processing time (or task time). The precedence relations illustrate all tasks and the relations between them. A task cannot be allocated before its predecessors since all tasks must be assigned to workstations while taking into account their relationships of precedence. The cycle time cannot be exceeded by the total of task times for a single workstation (workstation time). By integrating the precedence relations graphs of the models into a single precedence graph and calculating the average task processing time for each task, the mixed model assembly line balancing problem can be reduced to a simple assembly line balancing problem. So, the goal of solving the MiMALBP-2 is to determine the optimum job assignment with the shortest cycle time for a set number of workstations while maintaining the order of precedence. In this study, finding the ideal sequence of models is not taken into account.

5.2.2 Mathematical formulation

C Cycle time
n number of tasks
i task i where $i=1,\ldots, n$
m The number of workstations
k Workstation k where $k=1,\ldots,m$
t_i processing time of task i
P_i the set of predecessors of task i
X_{ik} equal to 1 if task i is assigned to workstation k, 0 otherwise

The goal is to maximize the production rate by optimizing the cycle time (minimizing the cycle time equation 5.1). Baybars *et al.* [18] developed the following mathematical formulation of the problem to solve SALBP-2 and it can be used to solve MiMALBP-2 :

$$\min C \tag{5.1}$$

Under the following constraints:

$$\sum_{k=1}^{m} X_{i,k} = 1 \tag{5.2}$$

$$\sum_{i=1}^{n} t_i \cdot X_{i,k} \leq C \tag{5.3}$$

$$\sum_{k=1}^{m} k \cdot X_{h,k} \leq \sum_{k=1}^{m} k \cdot X_{i,k} \quad where \quad h \in P_i \tag{5.4}$$

$$X_{i,k} \in \{0, 1\} \tag{5.5}$$

Equation (5.2) assures that each task is only affected once. Equation (5.3) requires that the sum of the process times of tasks allocated to the same workstation be less than or equal to the cycle time. The precedence relations between tasks are imposed by Equation (5.4). If task h must be completed before task i the index of the station where task h is affected must be less than or equal to the index of the station where task i is affected. Lastly, Equation (5.5) reflects the constraint of the decision variables' integrity.

5.3 The proposed algorithms to solve the MiMALBP type 2

The mixed model assembly line problem type 2 is addressed in this study using the hybridization of two meta-heuristics, the genetic algorithm (GA) and the GRASP (Greedy Randomized Adaptive Search Process). Every individual is a unique solution discovered using GRASP because it is utilized to seed the initial population of the GA. Figure 5.1 displays the suggested hybridization.

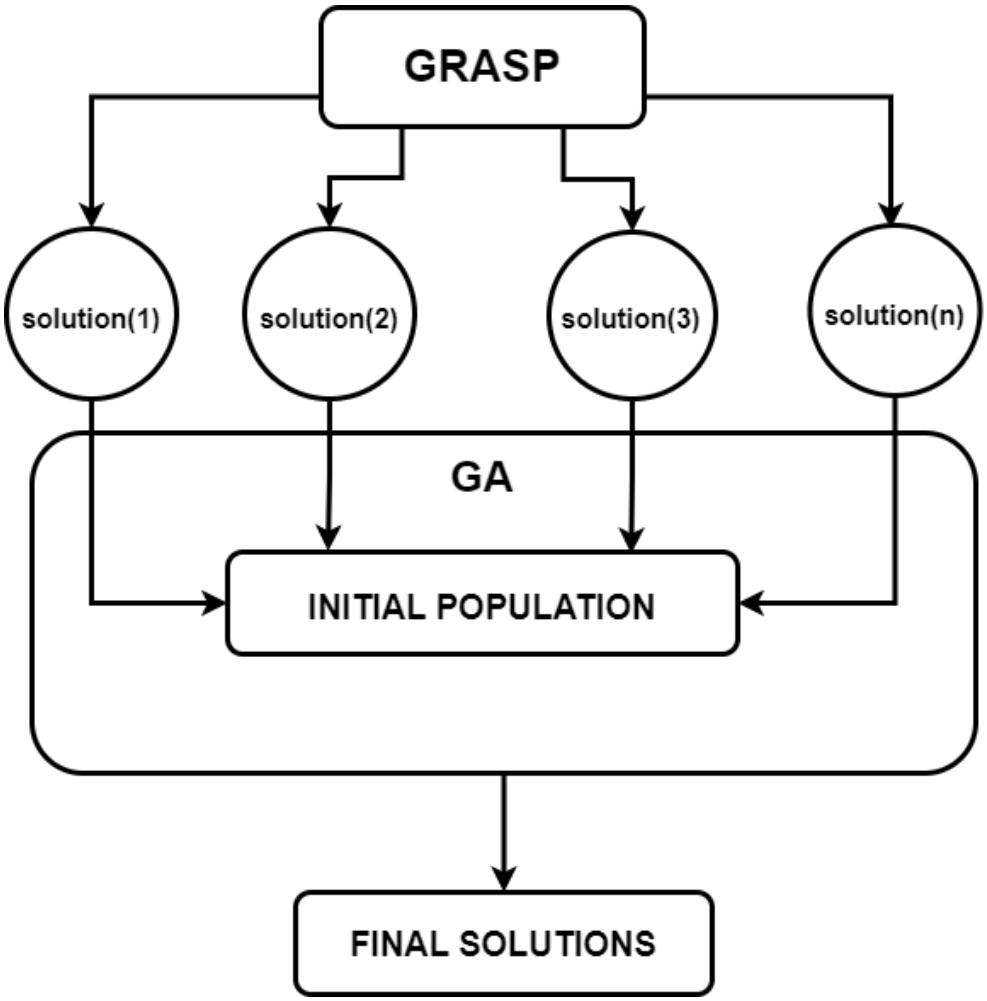

Figure 5.1: GA-GRASP hybridization.

5.3.1 Genetic algorithm

A number of assembly line balancing issues have been solved using the genetic algorithm, which is a well-known and efficient metaheuristic [72]. Each solution (individual) is evaluated in this metaheuristic after its fitness has been determined using a fitness function F. It starts with initial solutions (population), which are the starting point. For operators (crossover and mutation), the best solutions are chosen. In the end, evaluations are done on the individuals produced by the genetic operators. The number of generations is the total number of times all these GA processes are repeated [**thiruvady_ant_2020** , 210]. We employed the Elitism technique [211] in the proposed GA to ensure that the best solutions persisted until the final population by keeping 25 % of

the population's top performers in each generation to undergo GA operators. The GA stages are described as follows:

1. **Encoding of solutions**: Priority-based encoding is employed in this paper to build the task sequence based on the precedence graph. The task node is represented by the value of the gene, and its assignment priority is indicated by where it is located in the sequence.

2. **The initial population**: Each member of the initial population, which consists of a collection of individuals, symbolizes the final result of the greedy randomized adaptive search method.

3. **Fitness function**: The fitness function is computed for each individual in order to evaluate them. Because the goal of this GA is to maximize the production rate, the fitness function (F) is dependent on cycle time (C):

$$F = 1/C \tag{5.6}$$

4. **Selection**: Tournament selection is used to choose the finest candidates. Two players are randomly selected from the remaining 75 % of participants in each tournament, and the two with the shortest cycle times go through GA operators (crossover and mutation). Up until the necessary number of individuals are present in the mating pool, the selection procedure is repeated.

5. **Crossover**: The one-point crossover is employed to produce new offspring. First, two parents are picked from the list of those who were part of the selection process. Then, the one-point crossover is used between two parents to produce new offspring, as depicted in figure 5.2. Repeated tasks (genes) in the chromosome must be replaced by missing tasks.

6. **Mutation**: A swap mutation is one in which the values of two randomly selected genes (tasks) are switched. To create novel sequences, the process of mutation is done to a few individuals selected at random. In the end, tasks that disregard precedence relations must be reordered in order to fix unfeasible solutions.

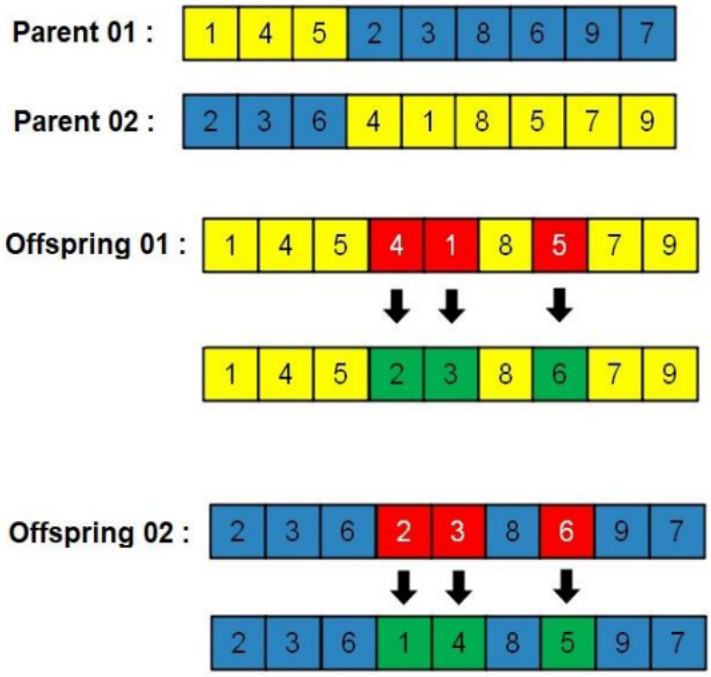

Figure 5.2: One-point crossover.

5.3.2 Greedy randomized adaptive search procedure (GRASP)

Several combinatorial problems have been solved using the meta-heuristic GRASP. Each iteration of this meta-heuristic comprises the construction phase and the local search phase. Building a feasible solution is the purpose of the construction phase, while the local search phase seeks to improve the solution established during the construction phase using a local search method [212]. The popular heuristic of ranked positional weight (RPW), which has been utilized to tackle the assembly line balancing problem, is employed to identify a feasible solution in the construction phase. With this heuristic, tasks are listed in declining order according to their positional weights. All other durations attributable to the successors are added to the duration of the selected task to determine the positioning weight [213].

During the construction process, each iteration is based on two lists: the Candidate List (CL) and the Restricted Candidate List (RCL) [201]. First, all tasks are included in the candidate list, and the RCL is generated from this list by selecting tasks with the highest positional weight while adhering to all criteria (precedence relations and available workstation time). The p elements

with the best positional weight limit the number of elements in the restricted candidate list. A task is picked at random from the RCL for the assignment. The selected task is removed from the RCL and CL, and the workstation time is updated. When all tasks in the CL have been allocated and the specified number of workstations has been met, the construction phase comes to an end. Otherwise, the construction phase resumes until a feasible solution with the precise number of workstations specified is found.

Because the solution identified during the construction phase may not be ideal, the local search is utilized to improve the produced solution. In the local search phase, neighborhood search is used to identify an optimal neighborhood solution by randomly changing the positions of two tasks in the sequence while obeying precedence relations. The cycle time of the new sequence is determined in order to compare it to the sequence discovered during the construction phase. When no alternative best-neighborhood solution is found, the process in the local search phase ends.

5.4 Numerical example

The suggested hybrid technique is tested on an example that illustrates a mixed model problem with two models A and B using Python 3.7.3 on a Computer with an Intel(R) Core (TM) i3-4005U Processor 1.70 GHz. Each model has its own precedence relations for each model, and each task in each model may have a distinct processing time. Tables 5.1 and 5.2 show data from models A and B, respectively. The first column is the task number, the second is the task time, and the last column is the immediate predecessors. Certain tasks are not included in assembling each model. To solve this mixed model assembly line problem, we transformed it into a simple problem by combining the graphs of models A and B into one graph, as shown in Figure 5.4. For each common task between models, the average processing time is calculated, and unnecessary relations are deleted. The problem contains 12 tasks and 12 precedence relations that must be respected during the assignment of tasks to workstations. As shown in Figure 5.4, the values inside the circles are the task numbers, and T is the average processing time. So, the aim is to find the minimum cycle

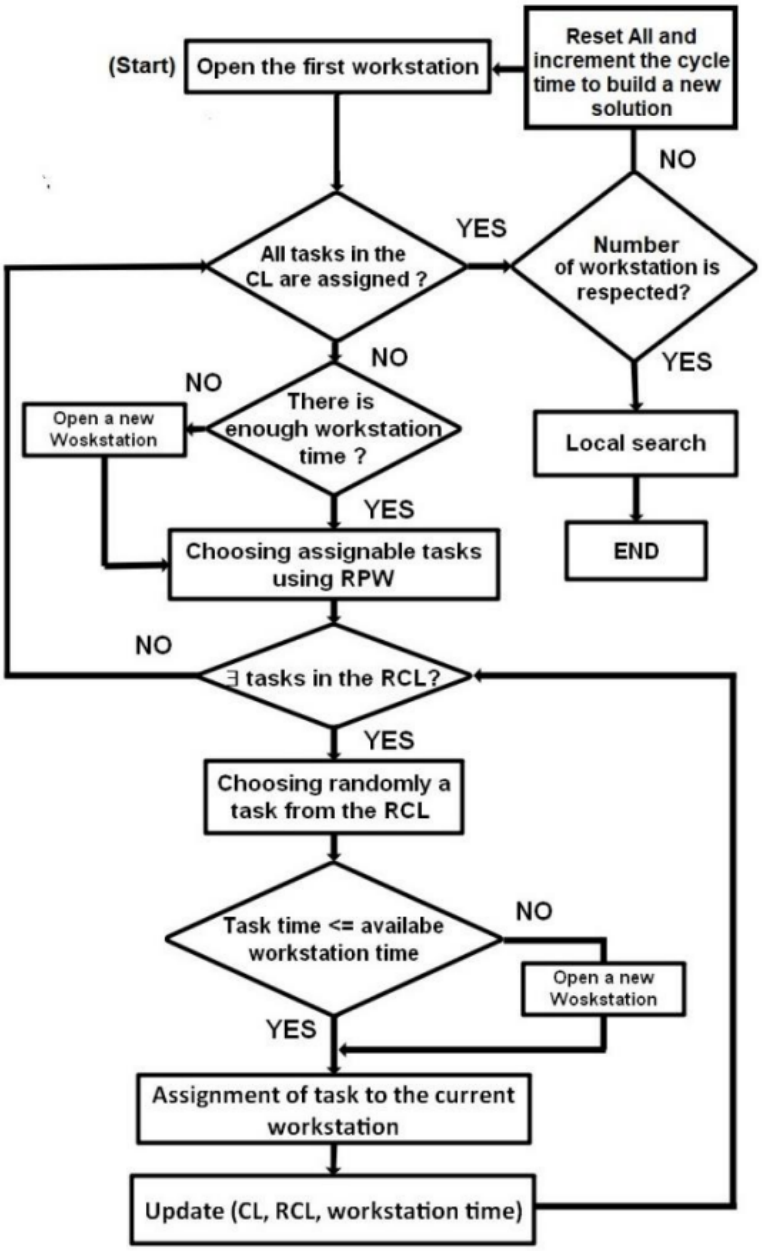

Figure 5.3: Adopted GRASP approach for MiMALBP resolution.

time based on the number of workstations. In this example, the number of workstations is 4.

Table 5.1: Model A

Task	Task time	Immediate predecessors
1	9	-
2	21	-
3	25	-
4	14	1
5	23	2
6	12	3
7	11	4, 5
8	7	5
9	20	5, 6
10	4	7, 8

Table 5.2: Model B

Task	Task time	Immediate predecessors
1	3	-
2	25	-
4	19	1
5	17	2
7	7	5, 5
8	15	5
9	8	5
10	13	7, 8
11	17	9
12	13	11

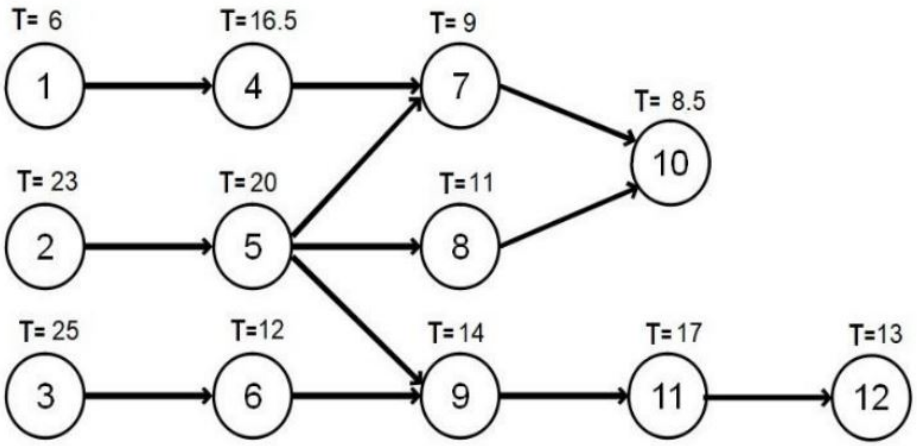

Figure 5.4: Combined precedence graph.

Table 5.3: GRASP-GA parameters.

	Parameters	Values
GRASP	RCL (number of elements)	3
	Number of iterations	10
GA	Number of generations	100
	Population size	20
	Crossover probability	0.5
	Mutation Probability	0.15
	Elitism	25 %

5.5 Discussion of obtained results

Figure 5.5 depicts the GRASP application in the proposed case. The GRASP was run 20 times to generate the initial population, with each iteration yielding a different feasible solution. As seen, each solution includes the solution discovered during the construction phase as well as its best neighborhood as determined by the local search technique. With cycle time (c = 47.5), 8 optimal solutions (S1, S5, S7, S9, S10, S12, S14, S17) were discovered. In certain situations (S7, S11, S13, and S18), no superior nearby solutions were found during the local search phase.

Figure 5.6 shows the outcome of the hybridization of the two proposed meta-heuristics (GRASP and Genetic algorithm). All neighborhood solutions identified in the 20 GRASP executions were employed as an initial population

in the genetic algorithm, and after 100 generations, 8 improved solutions (S1, S2, S3, S4, S5, S11, S15, S17) were discovered with cycle time (c= 45). Table 5.4 shows how best-found solutions varied in the order of tasks.

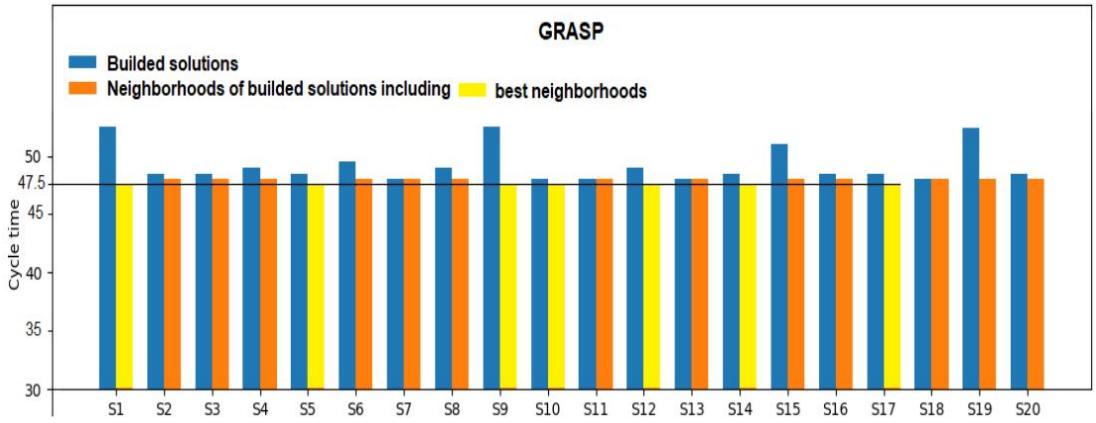

Figure 5.5: Obtained solutions using GRASP.

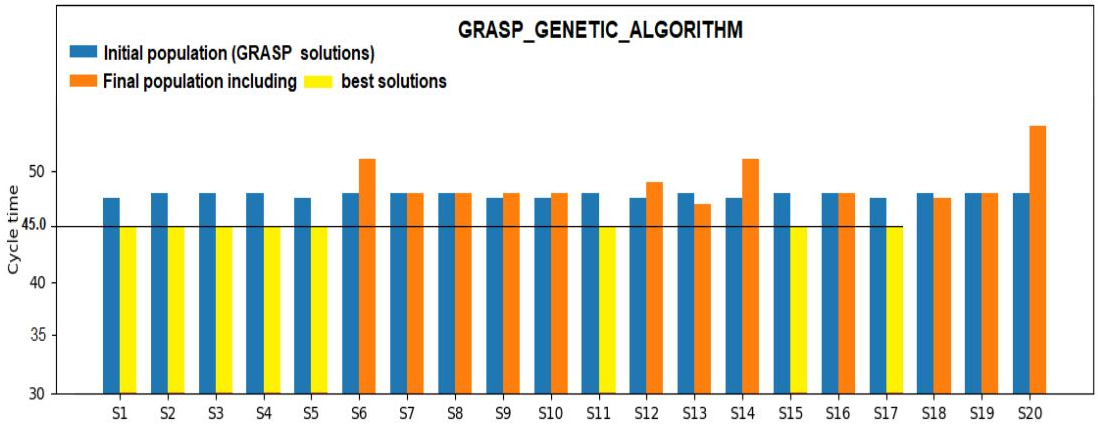

Figure 5.6: Obtained Solutions using GRASP-GA.

Any of the eight best solutions found in the final population can be chosen as the final solution because the only variation between them is the sequence of tasks. The first answer is picked at random as the final solution, and table 5.5 below illustrates the assignment of jobs to workstations based on the chosen solution.

The workload for each model can be calculated based on the chosen assignment; thus, for each workstation, the sum of processing times of assigned tasks is calculated to find the workload; however, because there is a difference

Table 5.4: Best sequences found in the final population

Solution	Tasks order
S1	2, 5, 3, 6, 1, 9, 11, 12, 8, 4, 7, 10
S2	2, 5, 1, 3, 6, 9, 11, 12, 4, 7, 8, 10
S3	3, 6, 1, 2, 5, 9, 11, 12, 4, 8, 7, 10
S4	2, 5, 1, 3, 6, 9, 11, 12, 8, 4, 7, 10
S5	2, 5, 1, 3, 6, 9, 11, 12, 4, 8, 7, 10
S11	2, 5, 3, 1, 6, 9, 11, 12, 4, 7, 8, 10
S15	3, 6, 1, 2, 5, 9, 11, 12, 8, 4, 7, 10
S17	3, 6, 1, 2, 5, 9, 11, 12, 8, 4, 7, 10

Table 5.5: Assignment of tasks to workstations

Workstation	Tasks
W1	2, 5
W2	3, 6, 1
W3	9, 11, 12
W4	8, 4, 7, 10

in processing times of common tasks between models, the workloads will be different, and thus the utilization of workstations during production is not stable due to the variety of products. Figure 5.7 illustrates the workload for models A, and B, and the Average workload. Overload and lead time were also calculated.

The reader can observe that the cycle time is surpassed in workstation 2 for model A and in workstation 4 for model B. This difficulty arises from the variety of models and can affect line efficiency; therefore, to address this issue in mixed-model assembly lines, the appropriate sequence of goods that can minimize work overload must be found. This is recognized in the literature as the mixed model sequencing problem .

According to the results, the suggested greedy randomized adaptive search technique was trapped in the local optima in different executions with (c = 47.5) in this example, and the local search procedure did not discover a neighborhood solution that reduces the cycle time in some situations. After hybridizing GRASP with the suggested GA and starting with all neighborhood solutions discovered using GRASP as the initial population, superior solutions

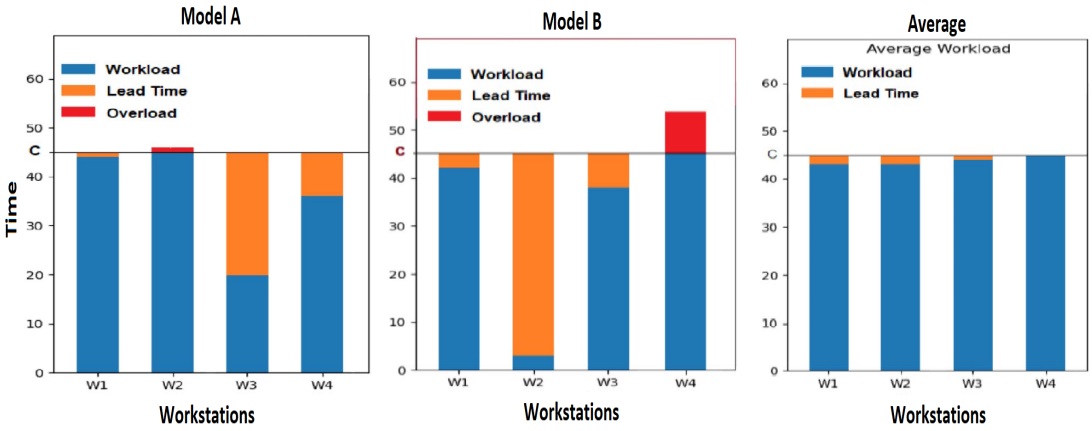

Figure 5.7: Model (A) workload, Model (B) workload, Average workload.

with a minimum cycle time (c = 45) were discovered.

In solving the proposed numerical example, the genetic algorithm was important in finding new solutions that could not be found using the proposed GRASP, and due to the presence of precedence relations constraints, the neighborhood search method was restricted because, for each solution found by the construction phase, a neighborhood solution that differed only in the positions of two tasks had to be found. However, by using genetic algorithm operators (crossover and mutation) over generations, the probability of discovering novel solutions increases.

5.6 Conclusion

The mixed model assembly line balance problem type 2 is addressed in this chapter, with the goal of determining the appropriate job assignment across workstations to reduce cycle time. To seed the initial population of the genetic algorithm, a greedy randomized adaptive search procedure based on a ranked positional weight heuristic is proposed, and a numerical example representing a mixed-model assembly line that assembles two different products is used to test the proposed hybridization. The results demonstrate the efficacy of the suggested hybridization by applying GA to improve the solutions discovered by GRASP. The proposed GRASP was trapped in the local optimal in the first stage due to the use of a fixed alpha value, which cannot help the

GRASP expand the search space, but in the second stage, the genetic algorithm starts with the GRASP solutions as an initial population to address the GRASP drawback and, as a result, avoid the local optimal problem.

Chapter 6

Maximization of the assembly line efficiency using an approach based on Genetic Algorithm

6.1 Introduction

Maximizing line efficiency is an important challenge in manufacturing systems; it is known as the assembly line balance problem type E, and it requires finding the ideal combination of workstations and cycle time that maximizes line efficiency. We proposed a genetic algorithm-based technique for dealing with SALBP type E in this contribution. To put the suggested genetic algorithm to the test, three different SALBP issues are developed, and the results are compared to those achieved by the Hybrid Reactive Greedy Randomized Adaptive Search Procedure.

This chapter is organized as follows; first, we describe the problem. Then, we discuss the proposed approach and the genetic algorithm used to solve the SLABP type 2. After that, we have a section that presents the computational experiments. Then, we discuss the obtained results. Finally, we conclude the chapter.

6.2 Problem description and mathematical formulation

n Number of tasks
w Number of workstations
c Cycle time
t_i Processing time of task i
t_{sum} The sum of all task times
t_{max} The maximum task time
I Idle time
P_i The set of predecessors of task i
$X_{ij} \in 0, 1$ 1 if task i is assigned to workstation j, 0 otherwise
c_{min} Lower bound of the cycle time
c_{max} Upper bound of the cycle time
w_{min} Lower bound of the number of workstations
w_{max} Upper bound of the number of workstations

SALBP type E is defined as follows: A single product is made on a serial assembly line by conducting a series of procedures known as tasks. The precedence relations graph represents the relationships between tasks. To be performed, each task requires a processing time known as task time, and it is assumed in SALBP type E that the nature of the task time is deterministic. Each task should be assigned to a different workstation. Each workstation might have several tasks, and the workstation time is the sum of all task times given to the same workstation. The cycle time, which is a set amount of time that each product unit spends at each workstation, must be longer than or equal to the workstation time; in other words, the workstation time cannot be longer than the cycle time. The idle time I is the duration during which the workstation is inactive and is represented by the difference between the cycle time and the workstation time. The objective in SALBP type E is to decrease the line capacity, which is defined as the product of the cycle time and the number of workstations, to maximize the line efficiency; consequently, the objective function can be represented by equation (6.1). Both cycle time and the number of workstations are unknown.

$$\min_{w} Z = c \cdot w \tag{6.1}$$

The line efficiency is defined as $E = \frac{t_{sum}}{c*w} * 100\%$. The idle time is defined as $I = (c * w) - t_{sum}$. The Objective function 6.1 is under the following constraints:

$$\sum_{j=1}^{w} X_{i,j} = 1 \;\; where\; i = 1, ..., n \tag{6.2}$$

$$\sum_{i=1}^{n} t_i \cdot X_{i,j} \leq c \;\; where\; j = 1, ..., w \tag{6.3}$$

$$\sum_{j=1}^{w} j \cdot X_{h,j} \leq \sum_{j=1}^{w} j \cdot X_{i,j} \;\; where\; i = 1, ..., n\; and\; h \in P_i \tag{6.4}$$

$$X_{i,j} \in \{0, 1\} \tag{6.5}$$

Constraint (6.2) requires task i to be assigned to just one workstation. Constraint (6.3) guarantees that the total time spent on all tasks allocated to the same workstation does not exceed the cycle time. The priority relations are imposed by constraint (6.4).

The SALBP type E can be characterized by a [W_{min}, W_{max}] interval of the possible number of workstations and/or a [c_{min}, c_{max}] interval of the potential cycle duration [47]. The cycle time must be larger than or equal to the maximum task time but less than or equal to the total of all task times:

$$t_{min} \leq c \leq t_{max}$$

As stated in [44], the ideal number of workstations may be determined when the cycle time is selected. The number of workstations lies between W_{min} and W_{max} and may be calculated as follows:

$$W_{min} = \left\lceil \frac{\sum_{i=1}^{n} t_i}{c} \right\rceil \tag{6.6}$$

$$W_{max} = \left\lceil \frac{\sum_{i=1}^{n} t_i}{t_{max}} \right\rceil \tag{6.7}$$

$$W_{min} \leq W \leq W_{max}$$

It should be noted that if the cycle time is set to tmax, the lower and upper bounds on the number of workstations will be equal.

6.3 The proposed genetic algorithm-based approach

6.3.1 The Adopted approach

The method used to solve SALBP type E is described in this section. The suggested strategy for dealing with SALBP type E is introduced first, followed by a description of the Genetic Algorithm that was used in our strategy.

There are two methods that can be utilized to establish the search space and resolve SALBP type E. The first step is to use the lower bound and upper bound [C_{min}, C_{max}] to calculate the cycle time interval. Next, find the ideal number of workstations for C_{min}, C_{max}, and any other values between them. In order to solve a SALBP type 1 sequence using this method, the optimal couple (cycle time, number of workstations) that maximizes line efficiency must be determined. This couple is therefore the best solution for SALBP type E. The second method involves using the lower bound and upper bound [W_{min}, W_{max}] to calculate the range of the potential number of workstations, and then searching for the ideal cycle time for W_{min}, W_{max} and all values in between. The optimum combination (cycle duration, number of workstations) that maximizes line efficiency is selected as the best solution for SALBP type E in this situation, which involves solving a sequence of SALBP type 2 problems.

In order to solve each SALBP type 2 and determine the best cycle time for each given number of workstations, we employed in this study the second strategy to solve SALBP type E and the genetic algorithm to solve SALBP type 2. The flowchart for the suggested method is shown in Figure 6.1.

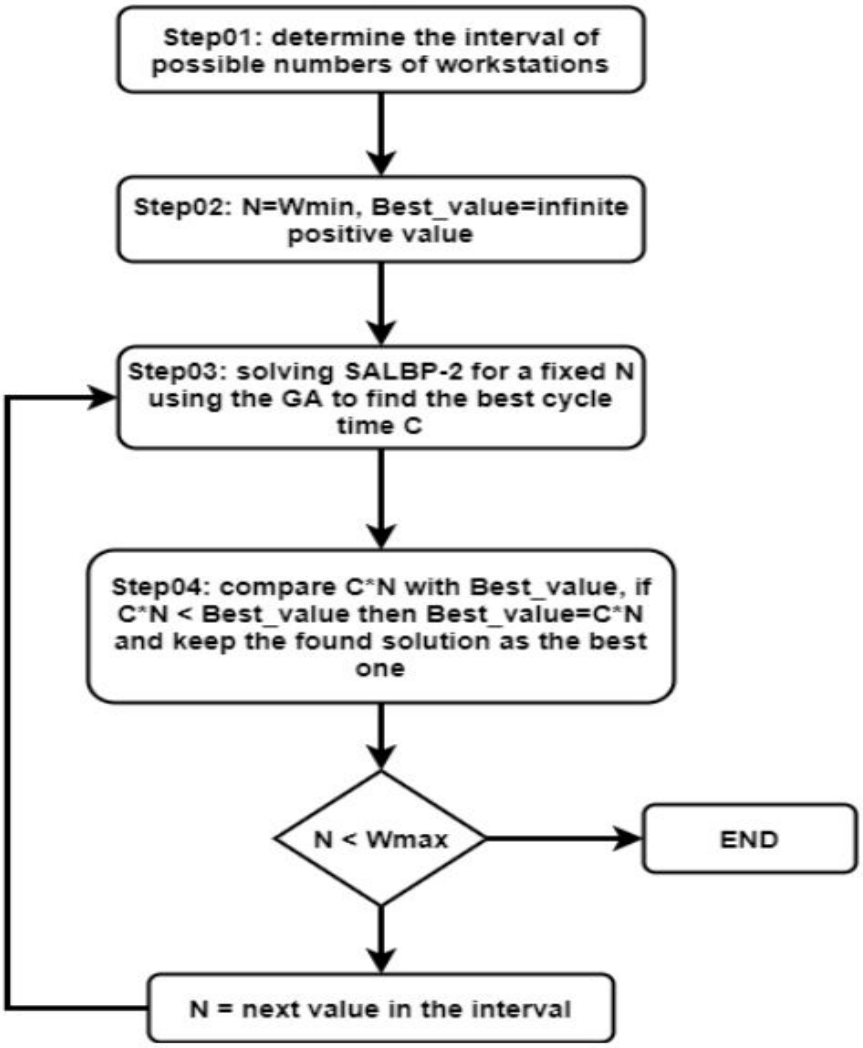

Figure 6.1: Flowchart of the proposed approach to solve the SLBP-E.

Equation (6) is employed to calculate the upper constraint on the number of workstations, and this upper bound serves as the stopping criterion. Due to the fact that real-world assembly lines require at least two workstations, we begin with $W_{min} = 2$ rather than $W_{min} = 1$ for the lower bound. When solving the ALBP, for instance, certain tasks cannot be assigned to the same workstation for safety concerns if there are negative zoning constraints that must be taken into account [167, 214].

6.3.2 The Adopted genetic algorithm

The genetic algorithm is a meta-heuristic that begins with a set of solutions known as the initial population, and each of those answers represents an individual. This meta-heuristic employs a number of processes to produce a new population, including the generation of the initial population, the evaluation of individuals using a fitness function, the process of selection, and the use of operators (crossover and mutation). These actions are repeated for a specified number of generations. Each solution in the genetic algorithm we propose represents a set of tasks, and the precedence rules that govern the order of the tasks in the set must be observed. The name of this representation is the "priority-based encoding method," in which each gene represents a task and the position of the node in the sequence denotes the task's priority of assignment. The steps below serve as the foundation for the suggested GA to solve SALBP-2.

- **Initial population generation:** We produce the initial population at random. Every individual in the population represents a feasible solution.
- **Individual evaluation:** which is based on the fitness function that can be used to assess individuals. The goal of this suggested genetic algorithm is to solve SALBP-2, where the cycle time must be minimized. As a result, the fitness function is based on the cycle time and is provided by F = 1/C, where C is the cycle time.
- **Selection:** Using this stage, the population's top candidates are chosen. The tournament selection is employed in this GA and operates as follows: Two participants (solutions) are chosen at random from the population for each tournament iteration, and the solution with the better fitness value (the shortest cycle time) is chosen. Up until the mating pool has the required number of solutions, the selection process is repeated.
- **Crossover operator:** In this GA, two chosen solutions (parents) from the mating pool are subjected to the one-point crossover in order to produce new solutions (offspring). As seen in the picture below, repeated tasks (genes) in generated solutions are replaced with missing tasks to maintain the feasibility of solutions. Additionally, the sequence of tasks is

checked to ensure that the new solution adheres to precedence restrictions.

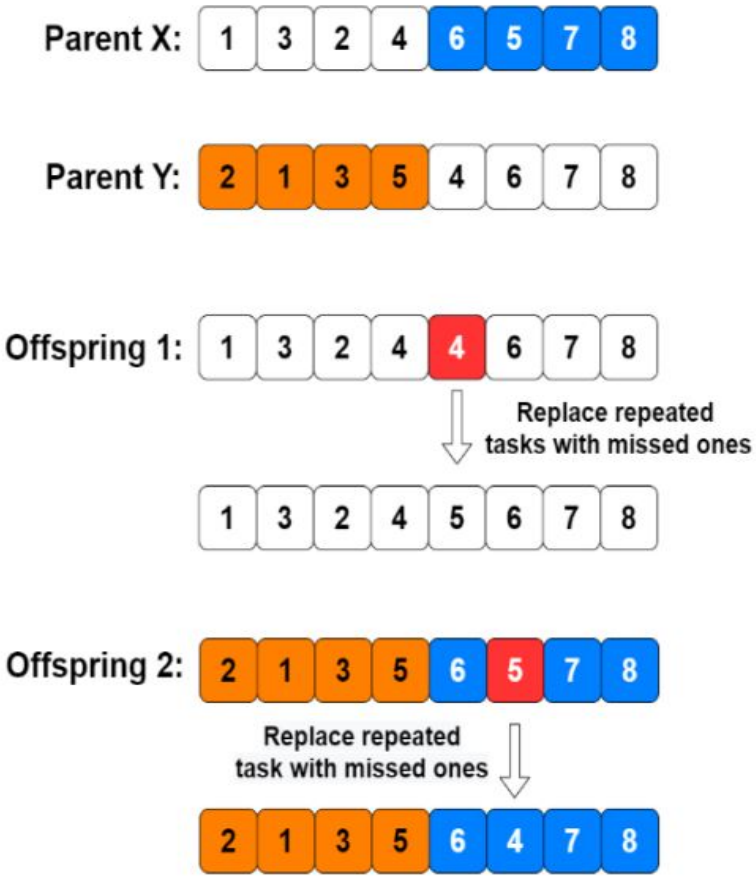

Figure 6.2: One-point crossover

- **Mutation operator:** The Swap mutation is applied in this GA. It involves exchanging the values of two tasks chosen at random from the solution. Once a new solution has been found, its viability is examined, and any necessary modifications are made. Depending on the mutation probability parameter, which is set by the programmer, the process of mutation is applied to some individuals.

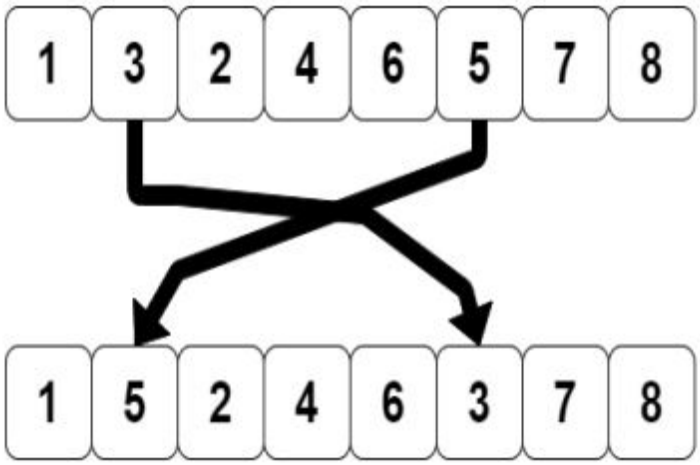

Figure 6.3: Swap mutation.

6.4 Computational experiments

We constructed three distinct problems to evaluate the suggested methodology. The first problem has 10 tasks, 9 precedence relations, and a total processing time of 23. The second problem has 14 tasks, 16 precedence relationships, and a 33-second processing time in total. Finally, there are 20 tasks, 22 precedence relations, and a total processing time of 42 in the third problem. The generated problems are displayed in Tables 6.1, 6.2, and 6.3. A computer with 8 GB of RAM and an Intel dual-core 1.70 GHz CPU is used to implement the suggested methodology. Python is the programming language used. We utilized the hybrid Reactive GRASP suggested in the next chapter to compare and assess the effectiveness of the proposed genetic algorithm. For each issue, the maximum number of workstations is determined using the formula (6). The upper bounds identified for each issue are shown in table 6.4.

We fixed the lower bound by 2 to identify the range of potential values for the number of workstations for each problem. As a result, the range of potential values for problem 1 is [W_{min} = 2, W_{max} = 6], problem 2 is [W_{min} = 2, W_{max} = 7], and problem 3 is [W_{min} = 2, W_{max} = 9]. We can calculate the number of SALB problems of type 2 that must be solved for each SALB problem of type E by establishing the intervals of possible values for the number of workstations. For instance, the interval for SALBP-E number 1 is [W_{min} = 2, W_{max} = 6], which requires that 5 SALB problems of type 2 be completed beginning at

$W_{min} = 2$. The parameters employed by the proposed evolutionary algorithm and the hybrid Reactive GRASP, respectively, to solve all problems are shown in tables 6.5 and 6.6, respectively.

Table 6.1: Problem 1

Task	Task time	Immediate predecessors
1	1	-
2	2	-
3	1	-
4	3	1
5	3	2
6	4	3
7	1	4
8	2	5
9	4	6
10	2	7, 8, 9

Table 6.2: Problem 2

Task	Task time	Immediate predecessors
1	2	-
2	3	-
3	1	1
4	5	3
5	2	1
6	2	2
7	1	5, 6
8	3	4
9	5	6
10	1	7
11	1	9
12	2	10, 11
13	2	8, 10
14	3	13, 12

Table 6.3: Problem 3

Task	Task time	Immediate predecessors
1	3	-
2	1	-
3	1	-
4	2	-
5	2	1
6	2	2
7	4	3, 4
8	5	5, 6
9	1	7
10	2	8
11	3	8
12	3	9
13	2	9
14	2	9
15	1	10
16	1	11
17	1	12
18	3	13
19	2	14
20	1	15, 16, 17, 18, 19

Table 6.4: Upper bounds used in generated problems

Problem	Upper bound on the number of workstations
1	$\lceil \frac{23}{2} \rceil = 6$
2	$\lceil \frac{33}{5} \rceil = 7$
3	$\left\lceil \frac{42}{5} \right\rceil = 9$

Table 6.5: Used parameters in the Genetic Algorithm

Parameters	Values
Number of generations (for each problem	3000
Population size (number of individuals)	20
Crossover probability	0.5
Mutation probability	0.15
Elitism (The percentage of best individuals kept to be passed to the next generation)	25 %

Table 6.6: Used parameters in the Hybrid Reactive GRASP

Parameters	Values
Alpha values	0, 0.5, 1
Max iterations	500
Max search iterations (searching for neighborhood)	20
Update period	100

6.5 Results and discussion

Tables 6.7, 6.8, and 6.9 show the results obtained after solving Problems 6.1, 6.2, and 6.3 using the suggested genetic algorithm and the hybrid reactive GRASP, respectively. The reader can see from the tables that the outcomes produced by the two algorithms are comparable. When there are 12, 8, or 6 workstations, problem 01's maximum line efficiency is 96 %. The hybrid reactive grasp outperforms the genetic algorithm in problem 01 when w = 5, which is the only distinction. When the number of workstations and the cycle time are assumed to be 2 and 11, respectively, in both methods, the highest line efficiency for problem 02 is 100 %. The greatest line efficiency for problem 03 is 100 % in three instances when the cycle time and the number of workstations are correspondingly (2 and 21), (3 and 14), and (6 and 7). We can see that when the line efficiency is 100% across all problems, there is no lost time since the overall idle time, which represents the time when workstations are idle, is equal to 0.

As a result of the data, we can draw the conclusion that the suggested genetic algorithm is just as effective as the hybrid reactive GRASP in resolving the SALBP type E's subproblem of the simple assembly line balancing problem type 2. For each issue, the optimal workstation assignments are shown

Table 6.7: Obtained results for problem 1

Problem	N-Workstations	Best cycle time		Idle time		Line efficiency %	
		GA	HRG	GA	HRG	GA	HRG
1	2	12	12	1	1	96	96
2	3	8	8	1	1	96	96
3	4	6	6	1	1	96	96
4	5	5	5	2	2	92	92
5	6	5	4	7	1	77	96

Table 6.8: Obtained results for problem 2

Problem	N-Workstations	Best cycle time		Idle time		Line efficiency %	
		GA	HRG	GA	HRG	GA	HRG
1	2	17	17	1	1	97	97
2	3	11	11	0	0	100	100
3	4	9	9	3	3	92	92
4	5	7	7	2	2	94	94
5	6	6	6	3	3	92	92
6	7	5	5	2	2	94	94

Table 6.9: Obtained results for problem 3

Problem	N-Workstations	Best cycle time		Idle time		Line efficiency %	
		GA	HRG	GA	HRG	GA	HRG
1	2	21	21	0	0	100	100
2	3	14	14	0	0	100	100
3	4	11	11	2	2	95	96
4	5	9	9	3	3	93	93
5	6	7	7	0	0	100	100
6	7	7	7	7	7	86	86
7	8	6	6	6	6	88	88
8	9	5	5	3	3	93	93

Table 6.10: The best found solutions for all problems

Problem	Best solutions/assignments	
	Genetic algorithm	Hybrid reactive GRASP
01	w1:[1,3,6], w2:[4,2] w3:[5,7,8], w4:[4,10]	w1:[3,6], w2:[2,5,1] w3:[9,8], w4:[4,7,10]
02	w1:[2,6,1,5,7,3] w2:[4,10,9] w3:[8,13,11,12,14]	w1:[2,1,3,4] w2:[6,5,9,11,7] w3:[10,12,8,13,14]
03	w1[3,4,7], w2:[1,9,2,14] w3:[6,12,5], w4:[8,10] w5:[13,11,16,15], w6:[17,19,18,20]	w1:[3,4,7], w2:[1,2,9,6] w3:[5,8], w4:[14,11,19] w5:[13,18,10], w6:[12,16,15,17,20]

in table 6.10 for each solution. The best solution is determined based on the following criteria: the solution with the highest line efficiency is chosen first, and if multiple solutions have the same line efficiency (as in problem 03), the solution with the lowest cycle time is chosen as the best option.

6.6 Conclusion

In this chapter, we presented our contribution in which The simple assembly line balance issue type E is tackled. The goal is to optimize assembly line efficiency. The primary problem is divided into multiple small assembly line balance problems of type 2 in order to discover the ideal combination of workstation number and cycle duration that maximizes total line efficiency. To solve each SALBP type 2, an accepted genetic method is presented. The suggested GA's performance is tested on three distinct problems, and the findings are compared to those of the hybrid reactive GRASP. According to the obtained findings, we concluded that the suggested genetic algorithm is efficient in addressing the basic assembly line balancing problem and can achieve the same outcomes as the efficient Hybrid Reactive GRASP.

Chapter 7

A Hybrid Approach for the Mixed-Model Assembly Line Balancing problem Type-II

7.1 Introduction

Manufacturers utilize mixed-model assembly lines to produce various versions of a single product in order to satisfy a variety of client needs without changing the assembly line's layout. Models are produced in mixed order, and the cycle time can be used to calculate how long it takes to produce one model. The mixed-model assembly line balancing problem type II (MiMALBP-II) is a well-known problem that must be solved in the step of designing the new assembly line in order to find a minimum average cycle time while taking into account all models to be produced in the assembly line. Minimizing the cycle time is a crucial issue by which the production rate of the mixed-model assembly line can be maximized. In order to solve MiMALBP-II, we suggest a hybrid approach in this study that combines the Ranked Positional Weight (RPW) and Reactive Greedy Randomized Adaptive Search Procedure (Reactive GRASP) heuristics. Three problems are used to evaluate the proposed method, and the outcomes are compared to those of the standard GRASP.

This chapter is arranged as follows: initially, we explain the mixed-model assembly line balance problem type 2. Following that, we will go through the proposed hybrid reactive GRASP. The computational experiments are completed in the next section. The results are then discussed. Finally, we conclude

the chapter.

7.2 Problem description and mathematical formulation

7.2.1 Problem description

In order to create a straight assembly line, a definite number K of workstations must be placed in a serial fashion. The goal is to determine the best cycle time C for this setup. Various models M of the same product are put together in this scenario in an interspersed order, and each model has its own precedence relations diagram G. The execution priority of each task t is determined by the precedence relations. The total number of tasks T that must be assigned to a group of workstations can be determined by combining several precedence relations diagrams into a single diagram. The workstation time (or cycle time) cannot be exceeded by the total task times allocated to a given workstation.

7.2.2 Mathematical formulation

T Number of tasks
i Task i
C Cycle time
K Number of workstations
k Workstation k
t_i Task time of task i
P_i The set of predecessors of task i
$X_{ik} \in 0, 1$ 1 if task i is assigned to workstation j, 0 otherwise

The mathematical formulation of the Simple Assembly Line Balancing Problem, which is as follows, can be employed by integrating all precedence diagrams into one diagram:

$$\min C \tag{7.1}$$

Equation (1) defines the objective function (minimizing the cycle time) and it is under the following constraints:

$$\sum_{k=1}^{K} X_{i,k} = 1 \quad where\ i = 1, ..., T \tag{7.2}$$

$$\sum_{i=1}^{T} t_i \cdot X_{i,k} \leq c \quad where\ k = 1, ..., K \tag{7.3}$$

$$\sum_{k=1}^{K} j \cdot X_{h,k} \leq \sum_{k=1}^{K} k \cdot X_{i,k} \quad where\ h \in P_i \tag{7.4}$$

$$X_{i,k} \in \{0, 1\} \tag{7.5}$$

Task i will always be assigned to a single workstation according to constraint (7.2). Constraint (7.3) ensures that the cycle time will not be exceeded by the total process times of all jobs assigned to the same workstation. The respect for task hierarchy is ensured by constraint (7.4). The integrity of the choice variables is finally constrained by constraint (7.5).

7.3 The proposed HYBRID REACTIVE GRASP

The Greedy Randomized Adaptive Search Procedure (GRASP), a multi-start meta-heuristic, is built on the foundation of two main phases: the construction phase and the local search phase. While the local search phase is used in the second stage to improve the first solution, the construction phase is utilized to create an initial solution that is not always the best one. The candidate list (CL), which includes all candidates (in our instance, tasks), and the restricted candidate list (RCL) are both employed throughout the creation process. The fixed value of the α (where $\alpha \in [0, 1]$) parameter and the costs of candidates in the CL are used to calculate a threshold value that determines which items from the CL should make up the RCL. Following that, one candidate is randomly selected from the RCL to create the partial solution, and the chosen individual is then removed from the candidate list [202].

In order to approximate solutions to the time slot assignment problem, Prais and Ribeiro [203] initially suggested the Reactive GRASP, an improvement of the original GRASP. The reactive GRASP differs from the basic GRASP in that it incorporates the idea of learning mechanisms during the construction phase and, rather than using a fixed alpha value during each iteration, selects at random an alpha value from a discrete set of possible values using selection probabilities. All selection probabilities have the same value in the initial iteration. Pi = 1/m, where i = 1,...,m (m is the number of alpha values), and the Reactive GRASP updates all selection probabilities based on prior results using equation (6) [202] after each predetermined period.

$$P_i = \frac{q_i}{\sum_{j=1}^{m} q_j} \tag{7.6}$$

Where qi = $\hat{Z}$/A i (i=1,...,m), $\hat{Z}$ is the incumbent answer and Ai is the average value of all solutions discovered using $\alpha = \alpha_i$ where i=1,...,m. If the values of $\alpha = \alpha_i$ result in the best solutions on average, the value of qi is increased. To develop the initial solution in our proposed Hybrid Reactive GRASP, we use the Ranked Positional Weight (RPW) heuristic as a greedy function in the construction phase. The CL contains all assignable jobs in the first iteration, and the best tasks are chosen from the CL based on their positional weight calculated by the RPW to generate the RCL. The threshold value must be determined to decide which tasks from the CL can be selected to form the RCL. In this stage, we incorporate certain changes to compute the threshold value; instead of utilizing the positional weight, we use its inverse (1/positional weight), as shown in the equation below:

$$TC_{th} = 1/W_{min} + \alpha(1/W_{max} - 1/W_{min}) \tag{7.7}$$

Where W_{min} and W_{max} are the minimal and maximal positional weights, respectively, of tasks that form the CL. The reason for using the inverse of the positional weight is that the Reactive GRASP uses the threshold value to determine which candidates can be selected from the CL to form the RCL in minimization problems, but in our case, we prioritize tasks having maximal positional weights, and using the inverse of the positional weight, we can maintain

the concept of the value-based mechanism used by the GRASP [201]. After the threshold value has been determined, all tasks with inverse positional weights that are less than or equal to the threshold value are chosen to construct the RCL. A task is selected at random from the RCL to create the partial solution. A change must be made by removing the chosen task from the CL at the conclusion of the iteration. New jobs that can be assigned are also taken off the list of tasks that must all be given to the CL. With this method, we combine the notion of randomization employed by the Greedy Randomized Adaptive Search Procedure with the concept of the greediness of the Ranked Positional Weight (desire for tasks having the highest positional weight).

In order to identify a better solution, we employ the local search phase, which uses the produced initial solution as a parameter. The final solution discovered by the construction phase might not be the best one. To maintain the viability of solutions, we use a neighbor search procedure in the local search phase of the proposed Hybrid reactive GRASP to try to find a neighbor assignment starting from the initial assignment discovered by the construction phase. A maximum search value that is supplied as a parameter to the local search phase controls how the proposed neighbor search procedure searches for a neighbor solution for a defined number of searches. The final result is the neighbor solution that yields the best objective value. The solution with the lowest objective value is kept as the incumbent solution by the Hybrid Reactive GRASP, which compares the objective values of the returned solution and the one that was previously held as the incumbent solution.

7.4 Computational experiments

On a computer with an Intel Core i3, 1.70 GHz, Python 3.7 was used to implement both the proposed Hybrid Reactive GRASP and the standard GRASP. The two algorithms are put to the test and compared using three different MiMALBP problems.

7.4.1 Data sets and parameters

The first issue is taken from [215], the second issue is derived from [216], and the third issue is taken from (https://assembly-line balancing.de/). Each

problem's features are shown in table 7.1; the parameters utilized in the Hybrid Reactive GRASP and the basic GRASP, respectively, to solve all issues, are shown in tables 7.2 and 7.3.

Table 7.1: Problems characteristics

Problem	Size	Number of tasks	Number of models	Number of workstations
Problem1	Small	12	2	4
Problem2	Medium	20	3	5
Problem3	Large	25	4	6

Table 7.2: Used parameters in the Hybrid Reactive GRASP

Problem	Iterations	Max search	Alpha values	Update period
Problem1	2000	15	0, 0.5, 1	100
Problem2	2000	25	0, 0.6, 1	100
Problem3	2000	35	0, 0.4, 0.9	100

Table 7.3: Used parameters in the basic GRASP

Problem	Iterations	Max search	Alpha values
Problem1	2000	15	0
Problem2	2000	25	0
Problem3	2000	35	0

7.4.2 Results and discussion

After applying the Hybrid Reactive GRASP and the standard GRASP to solve problems 1, 2, and 3, respectively, the findings are summarized in tales 7.4, 7.6, and **??**. According to Table 7.4, both algorithms discovered the same cycle time for problem 1, however, the Hybrid Reactive GRASP found more solutions than the classic GRASP did. According to table 7.6, the Hybrid Reactive GRASP outperformed the basic GRASP in terms of cycle time and produced more solutions for problem 2. Table **??** shows that the Hybrid Reactive GRASP for problem 3 achieves the ideal cycle time. The Hybrid Reactive GRASP finds 671 solutions compared to the Basic GRASP's 23 solutions. Finally, there isn't much of a difference in how long it takes for both algorithms to solve every

problem. The Hybrid Reactive GRASP expands the search space and increases the likelihood of discovering an optimal solution that cannot be discovered with the standard GRASP, which is based on a fixed alpha value, leading to the discovery of additional solutions.

Table 7.4: Obtained results for problem 1

Algorithm	Execution time (min)	Number of solutions found	Optimal cycle time
Hybrid Reactive GRASP	9:21.623	362	14.1
Basic GRASP	7:74.325	33	14.1

Table 7.5: Obtained results for problem 2

Algorithm	Execution time (min)	Number of solutions found	Optimal cycle time
Hybrid Reactive GRASP	16:20.682	671	1.5
Basic GRASP	14:50.059	47	1.6

Table 7.6: Obtained results for problem 3

Algorithm	Execution time (min)	Number of solutions found	Optimal cycle time
Hybrid Reactive GRASP	40:01.294	671	7.7
Basic GRASP	35:23.3306	23	8

7.5 Conclusion

In this final contribution, a hybrid reactive GRASP is employed to tackle the mixed-model assembly line balancing problem with the goal of lowering cycle time to maximize the assembly line's output rate. The ranking positional weight heuristic was utilized as a greedy function in the building phase, and a simple neighbor search strategy was applied to improve produced solutions in the local search phase. The suggested hybrid reactive GRASP was compared

to the basic reactive GRASP, and three distinct problems were chosen. The results revealed that the hybrid reactive GRASP outperformed the basic one in tackling medium and big tasks.

Chapter 8

Conclusions and Perspectives

In conclusion, this thesis has delved into the intricacies of resource optimization in complex assembly lines, with a primary focus on addressing the assembly line balancing problem. Through extensive research, analysis, and the application of various optimization techniques, we have explored the challenges associated with achieving efficiency, productivity, and cost-effectiveness in these manufacturing environments. The findings of this study have provided valuable insights into the methods and strategies that can be employed to optimize assembly lines. We have demonstrated that a combination of mathematical models, heuristics, and meta-heuristics can play a crucial role in achieving a balance between workstations, reducing cycle times, and reducing energy consumption, and other resources.

Summary of Our Contributions:

In this thesis, we explored the optimization difficulties of multiple resources in various complicated assembly lines, with an emphasis on the assembly line balancing problem and its various variations. We provided the state of the art of the problem including its different classifications, and types. Furthermore, We provided an overview of the various methods used in recent years to solve the discovered ALBP versions.

The first contribution addresses the problem of energy-efficient robotic mixed-model assembly line balancing. The primary goal of resolving this issue is to reduce energy consumption in robotic assembly lines where each robot has unique properties. We proposed a memory-based cuckoo search algorithm (MBCSA) to solve this difficult problem.

Our second contribution addressed the mixed-model assembly line balancing problem type 1, which aims to minimize the number of workstations in the line. We proposed a hybrid reactive greedy randomized adaptive search procedure (HRGRASP) to solve this problem. The HRGRASP is comprised of four major steps: construction, local search, evaluation of obtained solutions, and updating of selection probabilities. During the construction phase, we used the shortest processing time heuristic to build a solution. In the local search phase, obtained solutions are replaced by their best neighbors using a simple neighborhood procedure. The proposed HRGRASP is tested on seven problems and the results are compared to other methods.

In the third contribution, we proposed a hybridization of two meta-heuristics (the genetic algorithm and the greedy randomized adaptive search procedure) to solve the mixed-model assembly line balancing problem type 2, which aims to optimize the line's cycle time. The GRASP is used in this study to generate a set of feasible solutions to the addressed problem. In the genetic algorithm, the created set serves as the first population. We used ranked positional weight (RPW) in the GRASP's construction phase to create a solution, and a neighborhood procedure was used in the local search phase to improve the solution.

In our fourth contribution, We addressed the simple assembly line balancing problem type E, which aims to optimize both the cycle time and the number of workstations at the same time. As a solution to this problem, we have proposed a genetic algorithm (GA)-based approach. To evaluate our approach, we generated three different problems, and each problem was solved several times based on the problem data. The obtained results are compared to the reactive greedy randomized adaptive search procedure's results.

In the fifth contribution, we proposed a hybridization of the ranked positional weight (RPW) heuristic and the reactive greedy randomized adaptive search procedure (RGRASP) to solve the mixed-model assembly line balancing problem type 2. The RPW is used in the RGRASP construction phase to build feasible solutions. We used a local search method to find better neighbor solutions during the RGRASP's local search phase. To put the proposed algorithm to the test, three problems of varying sizes are generated.

Perspectives and Future Directions:

Hybridization of Algorithms: The various approaches presented in this thesis, from memory-based cuckoo search to hybrid reactive greedy randomized adaptive search procedures, have shown promising results in addressing assembly line balancing problems. Future research can explore further hybridization of these algorithms, combining the strengths of different techniques to tackle even more complex and nuanced variations of assembly line balancing problems. The pursuit of novel algorithmic combinations may lead to improved solutions and broader applicability.

Energy-Efficient Manufacturing: With a growing emphasis on sustainability and energy efficiency, there is a need to delve deeper into the optimization of energy consumption in robotic assembly lines. Future work can explore additional factors that affect energy efficiency, such as dynamic workload adjustments, resource allocation, and smart scheduling. Developing methods to integrate real-time data into the assembly line balancing process can contribute to more sustainable manufacturing practices.

Data-Driven Approaches: The success of assembly line balancing is closely tied to the quality and availability of data. Future research can focus on data-driven approaches, including machine learning and predictive analytics, to optimize the decision-making process. By integrating data from various sources, such as sensor networks and historical performance records, assembly line managers can make more informed decisions for resource allocation and production planning.

Dynamic and Real-Time Balancing: Assembly lines are often subjected to dynamic changes, such as product variations, machine failures, and unexpected resource constraints. Future work should explore adaptive assembly line balancing algorithms capable of dynamically adjusting to these changes in real-time. These algorithms could make use of predictive maintenance and IoT technologies to minimize disruptions and enhance overall efficiency.

Benchmark Datasets and Evaluation Metrics: The standardization of benchmark datasets and evaluation metrics is crucial for comparing different assembly line balancing algorithms. Future research can focus on the development of comprehensive benchmark datasets that reflect real-world scenarios, enabling more rigorous comparisons and fostering healthy competition in the field.

Human-Robot Collaboration: As robotic technologies continue to advance, there is potential for greater collaboration between humans and robots on assembly lines. Future work can explore how human-robot teams can be efficiently integrated, addressing not only the allocation of tasks but also the dynamics of collaboration and coordination. The development of algorithms for optimizing human-robot assembly lines could lead to enhanced productivity and worker satisfaction.

In conclusion, the research presented in this thesis provides a solid foundation for addressing the optimization of resources in complex assembly lines. However, the field is dynamic and evolving, and there are numerous avenues for further exploration. By continuing to push the boundaries of algorithmic innovation, embracing data-driven approaches, and adapting to the changing landscape of manufacturing, researchers can contribute to the advancement of efficient and sustainable assembly line operations across various industries.

Bibliography

[1] "Assembly Line Balancing Models". 2008, pp. 477–550. DOI: 10.1007/978-1-84800-181-7_7.

[2] M. Peshkin and J.E. Colgate. "Cobots". July 1999, pp. 335–341. DOI: 10.1108/01439919910283722.

[3] R. V. Johnson. "Assembly line balancing algorithms: computation comparisons". May 1981, pp. 277–287.

[4] U. Saif, Z. Guan, B. Wang, J. Mirza, and S. Huang. "A survey on assembly lines and its types". June 2014, pp. 95–105. DOI: 10.1007/s11465-014-0302-1.

[5] M. R. Abdullah Make, M.F.F. Ab. Rashid, and M.M. Razali. "A review of two-sided assembly line balancing problem". Mar 2017, pp. 1743–1763. DOI: 10.1007/s00170-016-9158-3.

[6] N. Boysen, M. Fliedner, and A. Scholl. "A classification of assembly line balancing problems". Dec 2007, pp. 674–693. DOI: 10.1016/j.ejor.2006.10.010.

[7] S. G. Ponnambalam, P. Aravindan, and G. Mogileeswar Naidu. "A Multi-Objective Genetic Algorithm for Solving Assembly Line Balancing Problem". Apr 2000, pp. 341–352. DOI: 10.1007/s001700050166.

[8] M. Ghobakhloo. "Industry 4.0, digitization, and opportunities for sustainability". Apr 2020, p. 119869. DOI: 10.1016/j.jclepro.2019.119869.

[9] M. Jusop and M.F.F. Ab Rashid. "Optimisation of Assembly Line Balancing Type-E with Resource Constraints Using NSGA-II". July 2016, pp. 195–199. DOI: 10.4028/www.scientific.net/KEM.701.195.

[10] B. Yagmahan. "Mixed-model assembly line balancing using a multi-objective ant colony optimization approach". Sept 2011, pp. 12453–12461. DOI: 10.1016/j.eswa.2011.04.026.

[11] P. Chutima. "A comprehensive review of robotic assembly line balancing problem". Jan 2022, pp. 1–34. DOI: 10.1007/s10845-020-01641-7.

[12] L. Belkharroubi and K. Yahyaoui. "Solving the energy-efficient Robotic Mixed-Model Assembly Line balancing problem using a Memory-Based Cuckoo Search Algorithm". Sept 2022, p. 105112. DOI: 10.1016/j.engappai.2022.105112.

[13] L. Belkharroubi and K. Yahyaoui. "Solving the mixed-model assembly line balancing problem type-I using a Hybrid Reactive GRASP". Dec 2022, pp. 108–131. DOI: 10.1080/21693277.2022.2065380.

[14] L. Belkharroubi and K. Yahyaoui. "A Hybrid Grasp-genetic Algorithm for Mixed-model Assembly Line Balancing Problem Type 2". Sept 2021, pp. 424–432.

[15] L. Belkharroubi and K. Yahyaoui. "Maximization of the assembly line efficiency using an approach based on Genetic Algorithm". 2022 2nd International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET). Meknes, Morocco: IEEE, Mar. 3, 2022, pp. 1–6. DOI: 10.1109/IRASET52964.2022.9737934.

[16] L. Belkharroubi and K. Yahyaoui. "A Hybrid approach for the Mixed-Model Assembly Line Balancing problem Type II". 2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS). Cracow, Poland: IEEE, Sept. 22, 2021, pp. 329–332. DOI: 10.1109/IDAACS53288.2021.9661039.

[17] J. M. Wilson. "Henry Ford vs. assembly line balancing". Feb 2014, pp. 757–765. DOI: 10.1080/00207543.2013.836616.

[18] İ. Baybars. "A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem". Aug 1986, pp. 909–932. DOI: 10.1287/mnsc.32.8.909.

[19] A. Scholl. Balancing and sequencing of assembly lines: with 75 tables. 2nd, rev. ed. Contributions to management science. HeidelbergNewYork: Physica-Verl, 1999. 318pp.

[20] B. Rekiek, A. Dolgui, A. Delchambre, and A. Bratcu. "State of art of optimization methods for assembly line design". Jan 2002, pp. 163–174. DOI: 10.1016/S1367-5788(02)00027-5.

[21] L. Turpin. "A note on understanding cycle time". Nov 2018, pp. 113–117. DOI: 10.1016/j.ijpe.2018.09.004.

[22] T.R. Hoffmann. "Eureka: A Hybrid System for Assembly Line Balancing". Jan 1992, pp. 39–47. DOI: 10.1287/mnsc.38.1.39.

[23] I. Gräßler, D. Roesmann, C. Cappello, and E. Steffen. "Skill-based worker assignment in a manual assembly line". 2021, pp. 433–438. DOI: 10.1016/j.procir.2021.05.100.

[24] Z. Li, N. Dey, A.S. Ashour, and Q. Tang. "Discrete cuckoo search algorithms for two-sided robotic assembly line balancing problem". Nov 2018, pp. 2685–2696. DOI: 10.1007/s00521-017-2855-5. URL: http://link.springer.com/10.1007/s00521-017-2855-5 (visited on 09/20/2023).

[25] Executive Summary World Robotics 2018 Industrial Robots. 2018.

[26] Z. Li, M.N. Janardhanan, Q. Tang, and P. Nielsen. "Co-evolutionary particle swarm optimization algorithm for two-sided robotic assembly line balancing problem". Sept 2016, p. 168781401666790. DOI: 10.1177/1687814016667907.

[27] G. Boschetti, M. Faccio, M. Milanese, and R. Minto. "C-ALB (Collaborative Assembly Line Balancing): a new approach in cobot solutions". Oct 2021, pp. 3027–3042. DOI: 10.1007/s00170-021-07565-7.

[28] H. Aguilar, A. García-Villoria, and R. Pastor. "A survey of the parallel assembly lines balancing problem". Dec 2020, p. 105061. DOI: 10.1016/j.cor.2020.105061.

[29] R. Rachamadugu. "Assembly line design with incompatible task assignments". Oct 1991, pp. 469–487. DOI: 10.1016/0272-6963(91)90006-J.

[30] H. Wang and S. Hu. "Manufacturing complexity in assembly systems with hybrid configurations and its impact on throughput". 2010, pp. 53–56. DOI: 10.1016/j.cirp.2010.03.007.

[31] S. Keckl, W. Kern, A. Abou-Haydar, and E. Westkämper. "An Analytical Framework for Handling Production Time Variety at Workstations of Mixed-model Assembly Lines". 2016, pp. 201–206. DOI: 10.1016/j.procir.2015.12.080.

[32] A. Alghazi and M.E. Kurz. "Mixed model line balancing with parallel stations, zoning constraints, and ergonomics". Jan 2018, pp. 123–153. DOI: 10.1007/s10601-017-9279-9.

[33] C. Becker and A. Scholl. "A survey on problems and methods in generalized assembly line balancing". Feb 2006, pp. 694–715. DOI: 10.1016/j.ejor.2004.07.023.

[34] O. Battaïa and A. Dolgui. "Reduction approaches for a generalized line balancing problem". Oct 2012, pp. 2337–2345. DOI: 10.1016/j.cor.2011.11.022.

[35] P. Bryan. "A STUDY ON GENERAL ASSEMBLY LINE BALANCING MODELING METHODS AND TECHNIQUES". PhD thesis. Clemson University. 257pp.

[36] M. Jusop and M.F.F. Ab Rashid. "A review on simple assembly line balancing type-e problem". Dec 2015, p. 012005. DOI: 10.1088/1757-899X/100/1/012005.

[37] Y. Kara, T. Paksoy, and C.-T. Chang. "Binary fuzzy goal programming approach to single model straight and U-shaped assembly line balancing". June 2009, pp. 335–347. DOI: 10.1016/j.ejor.2008.01.003.

[38] M. Vilà and J. Pereira. "An enumeration procedure for the assembly line balancing problem based on branching by non-decreasing idle time". Aug 2013, pp. 106–113. DOI: 10.1016/j.ejor.2013.03.003.

[39] R. Klein and A. Scholl. "Maximizing the production rate in simple assembly line balancing—A branch and bound procedure". June 1996, pp. 367–385. DOI: 10.1016/0377-2217(95)00047-X.

[40] İ. Baybars. "An efficient heuristic method for the simple assembly line balancing problem". Jan 1986, pp. 149–166. DOI: 10.1080/00207548608919719.

[41] B. W. Pearce, K. Antani, L. Mears, K. Funk, M.E. Mayorga, and M.E. Kurz. "An effective integer program for a general assembly line balancing problem with parallel workers and additional assignment restrictions". Jan 2019, pp. 180–192. DOI: 10.1016/j.jmsy.2018.12.011.

[42] R. Pastor and L. Ferrer. "An improved mathematical program to solve the simple assembly line balancing problem". June 2009, pp. 2943–2959. DOI: 10.1080/00207540701713832.

[43] E. C. Sewell and S.H. Jacobson. "A Branch, Bound, and Remember Algorithm for the Simple Assembly Line Balancing Problem". Aug 2012, pp. 433–442. DOI: 10.1287/ijoc.1110.0462.

[44] N.-C. Wei and I.-M. Chao. "A solution procedure for type E simple assembly line balancing problem". Oct 2011, pp. 824–830. DOI: 10.1016/j.cie.2011.05.015.

[45] O. Kilincci. "A Petri net-based heuristic for simple assembly line balancing problem of type 2". Jan 2010, pp. 329–338. DOI: 10.1007/s00170-009-2082-z.

[46] C. Blum. "Beam-ACO for Simple Assembly Line Balancing". Nov 2008, pp. 618–627. DOI: 10.1287/ijoc.1080.0271.

[47] R. Esmaeilbeigi, B. Naderi, and P. Charkhgard. "The type E simple assembly line balancing problem: A mixed integer linear programming formulation". Dec 2015, pp. 168–177. DOI: 10.1016/j.cor.2015.05.017.

[48] O. Kilincci and G.M. Bayhan. "P-invariant-based algorithm for simple assembly line balancing problem of type-1". May 2008, pp. 400–409. DOI: 10.1007/s00170-007-0975-2.

[49] H.-y. Zhang. "An improved immune algorithm for simple assembly line balancing problem of type 1". Dec 2017, pp. 317–326. DOI: 10.1177/1748301817710924.

[50] H. Zhang, Q. Yan, Y. Liu, and Z. Jiang. "An integer-coded differential evolution algorithm for simple assembly line balancing problem of type 2". Aug 2016, pp. 246–261. DOI: 10.1108/AA-11-2015-089.

[51] J. Pereira. "Empirical evaluation of lower bounding methods for the simple assembly line balancing problem". June 3 2015, pp. 3327–3340. DOI: 10.1080/00207543.2014.980014.

[52] J. F. Gonçalves and J.R. De Almeida. "A Hybrid Genetic Algorithm for Assembly Line Balancing". 2002, pp. 629–642. DOI: 10.1023/A:1020377910258.

[53] H. Gökçen and K. Agˇpak. "A goal programming approach to simple U-line balancing problem". June 2006, pp. 577–585. DOI: 10.1016/j.ejor.2004.09.021.

[54] J. Dou, J. Li, and C. Su. "A novel feasible task sequence-oriented discrete particle swarm algorithm for simple assembly line balancing problem of type 1". Dec. 2013, pp. 2445–2457. DOI: 10.1007/s00170-013-5216-2.

[55] G. Jirasirilerd, R. Pitakaso, K. Sethanan, S. Kaewman, W. Sirirak, and M. Kosacka-Olejnik. "Simple Assembly Line Balancing Problem Type 2 By Variable Neighborhood Strategy Adaptive Search: A Case Study Garment Industry". Mar 2020, p. 21. DOI: 10.3390/joitmc6010021.

[56] M. J. Saltzman and I. Baybars. "A two-process implicit enumeration algorithm for the simple assembly line balancing problem". Oct 1987, pp. 118–129. DOI: 10.1016/0377-2217(87)90276-1.

[57] C. G. S. Sikora, T.C. Lopes, D. Schibelbain, and L. Magatão. "Integer-based formulation for the simple assembly line balancing problem with multiple identical tasks". Feb 2017, pp. 134–144. DOI: 10.1016/j.cie.2016.12.026.

[58] O. A. Arık, E. Köse, and J. Forrest. "Simple assembly line balancing problem of Type 1 with grey demand and grey task durations". Oct 2019, pp. 401–414. DOI: 10.1108/GS-05-2019-0011.

[59] S. V. Ravelo. "Approximation algorithms for simple assembly line balancing problems". Mar 2022, pp. 432–443. DOI: 10.1007/s10878-021-00778-2.

[60] Emrani Noushabadi, Bahalke, Dolatkhahi, Dolatkhahi, and Makui. "Simple assembly line balancing problem under task deterioration". 2011, pp. 583–592. DOI: 10.5267/j.ijiec.2011.02.003.

[61] A. Baskar and M. Anthony Xavior. "Heuristics based on Slope Indices for Simple Type I Assembly Line Balancing Problems and Analyzing for a Few Performance Measures". 2020, pp. 3171–3180. DOI: 10.1016/j.matpr.2020.03.454.

[62] B. Toklu and U. Özcan. "A fuzzy goal programming model for the simple U-line balancing problem with multiple objectives". Mar 2008, pp. 191–204. DOI: 10.1080/03052150701651642.

[63] M. Lalaoui and A.E. Afia. "A Fuzzy generalized simulated annealing for a simple assembly line balancing problem". 2018, pp. 600–605. DOI: 10.1016/j.ifacol.2018.11.489.

[64] M. Arikan. "A Tabu Search Algorithm for Type-2 U-Shaped Simple Assembly Line Balancing Problem". Vol. 144. Lecture Notes on Data Engineering and Communications Technologies. Cham: Springer International Publishing, 2022, pp. 435–449. DOI: 10.1007/978-3-031-10388-9_32.

[65] J. I. Van Zante-de Fokkert and T.G. De Kok. "The mixed and multi model line balancing problem: a comparison". Aug 1997, pp. 399–412. DOI: 10.1016/S0377-2217(96)00162-2.

[66] A. Noorul Haq, K. Rengarajan, and J. Jayaprakash. "A hybrid genetic algorithm approach to mixed-model assembly line balancing". Mar. 2006, pp. 337–341. DOI: 10.1007/s00170-004-2373-3.

[67] A. Mamun, A. Khaled, S. Ali, and M. Chowdhury. "A heuristic approach for balancing mixed-model assembly line of type I using genetic algorithm". Sept 2012, pp. 5106–5116. DOI: 10.1080/00207543.2011.643830.

[68] S. Akpınar, G. Mirac Bayhan, and A. Baykasoglu. "Hybridizing ant colony optimization via genetic algorithm for mixed-model assembly line balancing problem with sequence dependent setup times between tasks". Jan 2013, pp. 574–589. DOI: 10.1016/j.asoc.2012.07.024.

[69] P. Sadeghi, R.D. Rebelo, and J.S. Ferreira. "Balancing mixed-model assembly systems in the footwear industry with a variable neighborhood descent method". July 2018, pp. 161–176. DOI: 10.1016/j.cie.2018.05.020.

[70] D. Thiruvady, A. Nazari, and A. Elmi. "An Ant Colony Optimisation Based Heuristic for Mixed-model Assembly Line Balancing with Setups". 2020 IEEE Congress on Evolutionary Computation (CEC). Glasgow, UK: IEEE, July 2020, pp. 1–8. DOI: 10.1109/CEC48606.2020.9185757.

[71] M. Lalaoui and A.E. Afia. "A versatile generalized simulated annealing using type-2 fuzzy controller for the mixed-model assembly line balancing problem". 2019, pp. 2804–2809. DOI: 10.1016/j.ifacol.2019.11.633.

[72] J.-H. Zhang, A.-P. Li, and X.-M. Liu. "Hybrid genetic algorithm for a type-II robust mixed-model assembly line balancing problem with interval task times". June 2019, pp. 117–132. DOI: 10.1007/s40436-019-00256-3.

[73] H. Gökçen and E. Erel. "BINARY INTEGER FORMULATION FOR MIXED-MODEL ASSEMBLY LINE BALANCING PROBLEM". Apr 1998, pp. 451–461. DOI: 10.1016/S0360-8352(97)00142-3.

[74] S. Choudhary and S. Agrawal. "Multiobjective Mixed Model Assembly Line Balancing Problem". 2015, pp. 655–663. DOI: 10.1007/978-3-319-11218-3_58.

[75] E. Erel and H. Gokcen. "Shortest-route formulation of mixed-model assembly line balancing problem". July 1999, pp. 194–204. DOI: 10.1016/S0377-2217(98)00115-5.

[76] H. Gokcen and E. Erel. "A goal programming approach to mixed-model assembly line balancing problem". Jan. 1997, pp. 177–185. DOI: 10.1016/S0925-5273(96)00069-2.

[77] P. Sivasankaran and P.M. Shahabudeen. "Heuristics for Mixed Model Assembly Line Balancing Problem with Sequencing". 2016, pp. 41–65. DOI: 10.4236/iim.2016.83005.

[78] Y. Zhang, L.-f. Tao, and F. Ju. "Balancing of mixed-model assembly line based on ant colony optimization algorithm". 2011 IEEE 18th International Conference on Industrial Engineering and Engineering Management (EM2011). Changchun, China: IEEE, Sept. 2011, pp. 898–901. DOI: 10.1109/ICIEEM.2011.6035302.

[79] W. Zhang and M. Gen. "An efficient multiobjective genetic algorithm for mixed-model assembly line balancing problem considering demand ratio-based cycle time". June 2011, pp. 367–378. DOI: 10.1007/s10845-009-0295-5.

[80] M. M. Razali, M.F.F. Ab. Rashid, and M.R.A. Make. "Mathematical Modelling of Mixed-Model Assembly Line Balancing Problem with Resources Constraints". Nov 2016, p. 012002. DOI: 10.1088/1757-899X/160/1/012002.

[81] X. Liu, X. Yang, and M. Lei. "Optimisation of mixed-model assembly line balancing problem under uncertain demand". Apr 2021, pp. 214–227. DOI: 10.1016/j.jmsy.2021.02.019.

[82] L. M. Liao, C.J. Huang, and J.H. Huang. "Applying multi-agent approach to mixed-model assembly line balancing". 2012 IEEE 6th International Conference on Management of Innovation & Technology (ICMIT2012). Bali, Indonesia: IEEE, June 2012, pp. 684–688. DOI: 10.1109/ICMIT.2012.6225889.

[83] N. V. Hop. "A heuristic solution for fuzzy mixed-model line balancing problem". Feb 2006, pp. 798–810. DOI: 10.1016/j.ejor.2004.07.029.

[84] D. Krenczyk, B. Skolud, and A. Herok. "A Heuristic and Simulation Hybrid Approach for Mixed and Multi-Model Assembly Line Balancing". Vol. 637, 2018, pp. 99–108. DOI: 10.1007/978-3-319-64465-3_10.

[85] J. C. Chen, Y.-Y. Chen, T.-L. Chen, and Y.-H. Kuo. "Applying two-phase adaptive genetic algorithm to solve multi-model assembly line balancing problems in TFT-LCD module process". July 2019, pp. 86–99. DOI: 10.1016/j.jmsy.2019.05.009.

[86] Hao Yu and Wei Shi. "A genetic algorithm for Multi-Model Assembly Line Balancing Problem". 2013 IEEE International Symposium on Assembly and Manufacturing (ISAM). Xi'an, China: IEEE, July 2013, pp. 369–371. DOI: 10.1109/ISAM.2013.6643482.

[87] A. Jafari Asl, M. Solimanpur, and R. Shankar. "Multi-objective multi-model assembly line balancing problem: a quantitative study in engine manufacturing industry". Sept 2019, pp. 603–627. DOI: 10.1007/s12597-019-00387-y.

[88] J. Pereira. "Modelling and solving a cost-oriented resource-constrained multi-model assembly line balancing problem". June 2018, pp. 3994–4016. DOI: 10.1080/00207543.2018.1427899.

[89] Z. Li, M.N. Janardhanan, and S.G. Ponnambalam. "Cost-oriented robotic assembly line balancing problem with setup times: multi-objective algorithms". Apr 2021, pp. 989–1007. DOI: 10.1007/s10845-020-01598-7.

[90] B.-q. Sun and L. Wang. "An estimation of distribution algorithm with branch-and-bound based knowledge for robotic assembly line balancing". June 2021. DOI: 10.1007/s40747-020-00166-z.

[91] Z. Abidin Çil, S. Mete, and K.Ağpak. "A Goal Programming Approach for Robotic Assembly Line Balancing Problem". 2016, pp. 938–942. DOI: 10.1016/j.ifacol.2016.07.896.

[92] Z. Li, M.N. Janardhanan, P. Nielsen, and Q. Tang. "Mathematical models and simulated annealing algorithms for the robotic assembly line balancing problem". Oct 2018, pp. 420–436. DOI: 10.1108/AA-09-2017-115.

[93] L. Borba, M. Ritt, and C. Miralles. "Exact and heuristic methods for solving the Robotic Assembly Line Balancing Problem". Oct 2018, pp. 146–156. DOI: 10.1016/j.ejor.2018.03.011.

[94] Z. A. Cil, S. Mete, and E. Ozceylan. "An efficient heuristic algorithm for solving robotic assembly line balancing problem". 8th International Conference on Information Technology (ICIT). Amman, Jordan: IEEE, May 2017, pp. 412–417. DOI: 10.1109/ICITECH.2017.8080035.

[95] S. Daoud, L. Amodeo, F. Yalaoui, H. Chehade, and P. Duperray. "New mathematical model to solve robotic assembly lines balancing". Volumes 45.6, May 2012, pp. 1353–1358. DOI: 10 .3182/20120523-3-RO-2023.00183.

[96] S. Daoud, H. Chehade, F. Yalaoui, and L. Amodeo. "Solving a robotic assembly line balancing problem using efficient hybrid methods". June 2014, pp. 235–259. DOI: 10.1007/s10732-014-9239-0.

[97] J. Gao, L. Sun, L. Wang, and M. Gen. "An efficient approach for type II robotic assembly line balancing problems". Apr. 2009, pp. 1065–1080. DOI: 10.1016/j.cie.2008.09.027.

[98] M. N. Janardhanan, Z. Li, G. Bocewicz, Z. Banaszak, and P. Nielsen. "Metaheuristic algorithms for balancing robotic assembly lines with sequence-dependent robot setup times". Jan 2019, pp. 256–270. DOI: 10.1016/j.apm.2018.08.016.

[99] H. Kim and S. Park. "A strong cutting plane algorithm for the robotic assembly line balancing problem". Aug 1995, pp. 2311–2323. DOI: 10.1080/00207549508904817.

[100] G. Levitin, J. Rubinovitz, and B. Shnits. "A genetic algorithm for robotic assembly line balancing". Feb 2006, pp. 811–825. DOI: 10.1016/j.ejor.2004.07.030.

[101] M. Rabbani, S. Z. B. Behbahan, and H. Farrokhi-Asl. "The Collaboration of Human-Robot in Mixed-Model Four-Sided Assembly Line Balancing Problem". Oct 2020, pp. 71–81. DOI: 10.1007/s10846-020-01177-1.

[102] J. Mukund Nilakantan and S. Ponnambalam. "An efficient PSO for type II robotic assembly line balancing problem" 2012 IEEE International Conference on Automation Science and Engineering (CASE 2012). Seoul, Korea (South): IEEE, Aug. 2012, pp. 600–605. DOI: 10.1109/CoASE.2012.6386398.

[103] J. Mukund Nilakantan, S. G. Ponnambalam, N. Jawahar, and G. Kanagaraj. "Bio-inspired search algorithms to solve robotic assembly line balancing problems". Aug 2015, pp. 1379–1393. DOI: 10.1007/s00521-014-1811-x.

[104] J. M. Nilakantan, S. G. Ponnambalam, and P. Nielsen. "Energy-Efficient Straight Robotic Assembly Line Using Metaheuristic Algorithms". 2018, pp. 803–814. DOI: 10.1007/978-981-10-5687-1_72.

[105] J. Pereira, M. Ritt, and Ó. C. Vásquez. "A memetic algorithm for the cost-oriented robotic assembly line balancing problem". Nov 2018, pp. 249–261. DOI: 10.1016/j.cor.2018.07.001.

[106] Y. Chi, Z. Qiao, Y. Li, M. Li, and Y. Zou. "Type-1 Robotic Assembly Line Balancing Problem That Considers Energy Consumption and Cross-Station Design". Nov 2022, p. 218. DOI: 10.3390/systems10060218.

[107] A. Yoosefelahi, M. Aminnayeri, H. Mosadegh, and H. D. Ardakani. "Type II robotic assembly line balancing problem: An evolution strategies algorithm for a multi-objective model". Apr 2012, pp. 139–151. DOI: 10.1016/j.jmsy.2011.10.002.

[108] B. Zhou and Q. Wu. "An improved immune clonal selection algorithm for bi-objective robotic assembly line balancing problems considering time and space constraints". July 2019, pp. 1868–1892. DOI: 10.1108/EC-11-2018-0512. (Visited on 09/21/2023).

[109] M. Pınarba¸sı. "New chance-constrained models for U-type stochastic assembly line balancing problem". July 2021, pp. 9559–9573. DOI: 10.1007/s00500-021-05921-z.

[110] Ö. F. Yılmaz. "An integrated bi-objective U-shaped assembly line balancing and parts feeding problem: optimization model and exact solution method". Sept 2022, pp. 679–696. DOI: 10.1007/s10472-020-09718-y.

[111] M. Şahin and T. Kellegöz. "An efficient grouping genetic algorithm for U-shaped assembly line balancing problems with maximizing production rate". Sept 2017, pp. 213–229. DOI: 10.1007/s12293-017-0239-0.

[112] Y. Li, X. Hu, X. Tang, and I. Kucukkoc. "Type-1 U-shaped Assembly Line Balancing under uncertain task time". 2019, pp. 992–997. DOI: 10.1016/j.ifacol.2019.11.324.

[113] Z. Zhang, Q. Tang, D. Han, and X. Qian. "An enhanced multi-objective JAYA algorithm for U-shaped assembly line balancing considering preventive maintenance scenarios". Oct 2021, pp. 6146–6165. DOI: 10.1080/00207543.2020.1804639.

[114] B. Zhang and L. Xu. "An improved flower pollination algorithm for solving a Type-II U-shaped assembly line balancing problem with energy consideration". Sept 2020, pp. 847–856. DOI: 10.1108/AA-07-2019-0144.

[115] A. Nourmohammadi, M. Fathi, M. Zandieh, and M. Ghobakhloo. "A Water-Flow Like Algorithm for Solving U-Shaped Assembly Line Balancing Problems". 2019, pp. 129824–129833. DOI: 10.1109/ACCESS.2019.2939724.

[116] Z. Zhang, Q. Tang, D. Han, and Z. Li. "Enhanced migrating birds optimization algorithm for U-shaped assembly line balancing problems with workers assignment". Nov 2019, pp. 7501–7515. DOI: 10.1007/s00521-018-3596-9.

[117] D. Ajenblit and R. Wainwright. "Applying genetic algorithms to the U-shaped assembly line balancing problem". 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence. Anchorage, AK, USA: IEEE, 1998, pp. 96–101. DOI: 10.1109/ICEC.1998.699329.

[118] S. Avikal, R. Jain, P. Mishra, and H. Yadav. "A heuristic approach for U-shaped assembly line balancing to improve labor productivity". Apr 2013, pp. 895–901. DOI: 10.1016/j.cie.2013.01.001.

[119] A. Fattahi, S. Elaoud, E. Sadeqi Azer, and M. Turkay. "A novel integer programming formulation with logic cuts for the U-shaped assembly line balancing problem". Mar 2014, pp. 1318–1333. DOI: 10.1080/00207543.2013.832489.

[120] M. Ghadiri Nejad, A. Husseinzadeh Kashan, and S. M. Shavarani. "A novel competitive hybrid approach based on grouping evolution strategy algorithm for solving U-shaped assembly line balancing problems". Oct 2018, pp. 555–566. DOI: 10.1007/s11740-018-0836-x.

[121] R. K. Hwang, H. Katayama, and M. Gen. "U-shaped assembly line balancing problem with genetical algorithm". Aug 2008, pp. 4637–4649. DOI: 10.1080/00207540701247906.

[122] V. Jonnalagedda and B. Dabade. "Application of Simple Genetic Algorithm to U-Shaped Assembly Line Balancing Problem of Type II". 2014, pp. 6168–6173. DOI: 10.3182/20140824-6-ZA-1003.01769.

[123] M. Li, Q. Tang, Q. Zheng, X. Xia, and C. Floudas. "Rules-based heuristic approach for the U-shaped assembly line balancing problem". Aug 2017, pp. 423–439. DOI: 10.1016/j.apm.2016.12.031.

[124] Z. Li, I. Kucukkoc, and Z. Zhang. "Branch, bound and remember algorithm for U-shaped assembly line balancing problem". Oct 2018, pp. 24–35. DOI: 10.1016/j.cie.2018.06.037.

[125] Z. Li, M. N. Janardhanan, and H. F. Rahman. "Enhanced beam search heuristic for U-shaped assembly line balancing problems". Apr 2021, pp. 594–608. DOI: 10.1080/0305215X.2020.1741569.

[126] P. Sresracoo, N. Kriengkorakot, P. Kriengkorakot, and K. Chantarasamai. "U-Shaped Assembly Line Balancing by Using Differential Evolution Algorithm". Dec 2018, p. 79. DOI: 10.3390/mca23040079.

[127] M. Khorram, M. Eghtesadifard, and S. Niroomand. "Hybrid meta-heuristic algorithms for U-shaped assembly line balancing problem with equipment and worker allocations". Mar 2022, pp. 2241–2258. DOI: 10.1007/s00500-021-06472-z.

[128] D. Ogan and M. Azizoglu. "A branch and bound method for the line balancing problem in U-shaped assembly lines with equipment requirements". July 2015, pp. 46–54. DOI: 10.1016/j.jmsy.2015.02.007.

[129] Z. Zhang and W. Cheng. "An Exact Method for U-Shaped Assembly Line Balancing Problem". 2010 2nd International Workshop on Intelligent Systems and Applications (ISA). Wuhan, China: IEEE, May 2010, pp. 1–4. DOI: 10.1109/IWISA.2010.5473379.

[130] Z. Q. Zhang and W. M. Cheng. "Solving Fuzzy U-Shaped Line Balancing Problem with Exact Method". June 2010, pp. 1046–1051. DOI: 10.4028/www.scientific.net/AMM.26-28.1046.

[131] Z. Li, I. Kucukkoc, and J. M. Nilakantan. "Comprehensive review and evaluation of heuristics and meta-heuristics for two-sided assembly line balancing problem". Aug 2017, pp. 146–161. DOI: 10.1016/j.cor.2017.03.002.

[132] U. Özcan and B. Toklu. "A tabu search algorithm for two-sided assembly line balancing". Aug 2009, pp. 822–829. DOI: 10.1007/s00170-008-1753-5.

[133] U. Özcan and B. Toklu. "Multiple-criteria decision-making in two-sided assembly line balancing: A goal programming and a fuzzy goal programming models". June 2009, pp. 1955–1965. DOI: 10.1016/j.cor.2008.06.009.

[134] L. Özbakır and P. Tapkan. "Bee colony intelligence in zone constrained two-sided assembly line balancing problem". Sept 2011, pp. 11947–11957. DOI: 10.1016/j.eswa.2011.03.089.

[135] D. Kizilay and Z. A. Çil. "Constraint programming approach for multi-objective two-sided assembly line balancing problem with multi-operator stations". Aug 2021, pp. 1315–1330. DOI: 10.1080/0305215X.2020.1786081.

[136] A. Baykasoglu and T. Dereli. "Two-sided assembly line balancing using an ant-colony-based heuristic". Mar 2008, pp. 582–588. DOI: 10.1007/s00170-006-0861-3.

[137] X. Duan, B. Wu, Y. Hu, J. Liu, and J. Xiong. "An improved artificial bee colony algorithm with MaxTF heuristic rule for two-sided assembly line balancing problem". June 2019, pp. 241–253. DOI: 10.1007/s11465-018-0518-6.

[138] M. Gansterer and R. F. Hartl. "One-and two-sided assembly line balancing problems with real-world constraints". Apr 2018, pp. 3025–3042. DOI: 10.1080/00207543.2017.1394599.

[139] H.-Y. Kang and A. H. I. Lee. "An evolutionary genetic algorithm for a multi-objective two-sided assembly line balancing problem: a case study of automotive manufacturing operations". Jan 2023, pp. 66–88. DOI: 10.1080/16843703.2022.2079062.

[140] D. Khorasanian, S. R. Hejazi, and G. Moslehi. "Two-sided assembly line balancing considering the relationships between tasks". Dec 2013, pp. 1096–1105. DOI: 10.1016/j.cie.2013.08.006.

[141] Y. K. Kim, Y. Kim, and Y. J. Kim. "Two-sided assembly line balancing: A genetic algorithm approach". Jan 2000, pp. 44–53. DOI: 10.1080/095372800232478.

[142] Y. K. Kim, W. S. Song, and J. H. Kim. "A mathematical model and a genetic algorithm for two-sided assembly line balancing". Mar 2009, pp. 853–865. DOI: 10.1016/j.cor.2007.11.003.

[143] D. Lei and X. Guo. "Variable neighborhood search for the second type of two-sided assembly line balancing problem". Aug 2016, pp. 183–188. DOI: 10.1016/j.cor.2016.03.003.

[144] Z. Li, Q. Tang, and L. Zhang. "Two-sided assembly line balancing problem of type I: Improvements, a simple algorithm and a comprehensive study". Mar 2017, pp. 78–93. DOI: 10.1016/j.cor.2016.10.006.

[145] Z. Li, I. Kucukkoc, and Z. Zhang. "Branch, bound and remember algorithm for two-sided assembly line balancing problem". Aug 2020, pp. 896–905. DOI: 10.1016/j.ejor.2020.01.032.

[146] Y. Li, I. Kucukkoc, and X. Tang. "Two-sided assembly line balancing that considers uncertain task time attributes and incompatible task sets". Mar 2021, pp. 1736–1756. DOI: 10.1080/00207543.2020.1724344.

[147] A. Roshani, P. Fattahi, A. Roshani, M. Salehi, and A. Roshani. "Cost-oriented two-sided assembly line balancing problem: A simulated annealing approach". Aug 2012, pp. 689–715. DOI: 10.1080/0951192X.2012.664786.

[148] R. B. Taha, A. K. El-Kharbotly, Y. M. Sadek, and N. H. Afia. "A Genetic Algorithm for solving two-sided assembly line balancing problems". Sept 2011, pp. 227–240. DOI: 10.1016/j.asej.2011.10.003.

[149] Q. Tang, Z. Li, and L. Zhang. "An effective discrete artificial bee colony algorithm with idle time reduction techniques for two-sided assembly line balancing problem of type-II". July 2016, pp. 146–156. DOI: 10.1016/j.cie.2016.05.004.

[150] P. Tapkan, L. Özbakır, and A. Baykasoğlu. "Bees Algorithm for constrained fuzzy multi-objective two-sided assembly line balancing problem". Aug 2012, pp. 1039–1049. DOI: 10.1007/s11590-011-0344-9.

[151] P. Tapkan, L. Ozbakir, and A. Baykasoglu. "Modeling and solving constrained two-sided assembly line balancing problem via bee algorithms". Nov 2012, pp. 3343–3355. DOI: 10.1016/j.asoc.2012.06.003.

[152] M. S. Yang, L. Ba, Y. Liu, H. Y. Zheng, J. T. Yan, X. Q. Gao, and J. M. Xiao. "An Improved Genetic Simulated Annealing Algorithm for Stochastic Two-Sided Assembly Line Balancing Problem". Mar 2019, pp. 175–186. DOI: 10.2507/IJSIMM18(1)CO4.

[153] G. Tunçel and D. Aydın. "Two-sided assembly line balancing using teaching–learning based optimization algorithm". Aug 2014, pp. 291–299. DOI: 10.1016/j.cie.2014.06.006.

[154] B. Wang, Z. Guan, D. Li, C. Zhang, and L. Chen. "Two-sided assembly line balancing with operator number and task constraints: a hybrid imperialist competitive algorithm". Sept 2014, pp. 791–805. DOI: 10.1007/s00170-014-5816-5.

[155] E.-F. Wu, Y. Jin, J.-S. Bao, and X.-F. Hu. "A branch-and-bound algorithm for two-sided assembly line balancing". Nov 2008, pp. 1009–1015. DOI: 10.1007/s00170-007-1286-3.

[156] Y. Zhong, Z. Deng, and K. Xu. "An effective artificial fish swarm optimization algorithm for two-sided assembly line balancing problems". Dec 2019, p. 106121. DOI: 10.1016/j.cie.2019.106121.

[157] A. Lusa. "A survey of the literature on the multiple or parallel assembly line balancing problem". 2008, p. 50. DOI: 10.1504/EJIE.2008.016329.

[158] L. Özbakır and G. Seçme. "A hyper-heuristic approach for stochastic parallel assembly line balancing problems with equipment costs". Mar 2022, pp. 577–614. DOI: 10.1007/s12351-020-00561-x.

[159] U. Özcan. "Balancing stochastic parallel assembly lines". Nov 2018, pp. 109–122. DOI: 10.1016/j.cor.2018.05.006.

[160] A. Baykasoğlu, L. Özbakir, L. Görkemli, and B. Görkemli. "Multi-colony ant algorithm for parallel assembly line balancing with fuzzy parameters". 2012, pp. 283–295. DOI: 10.3233/IFS-2012-0520.

[161] J. F. Bard. "Assembly line balancing with parallel workstations and dead time". June 1989, pp. 1005–1018. DOI: 10.1080/00207548908942604.

[162] A. Baykasoglu, L. Ozbakur, L. Gorkemli, and B. Gorkemli. "Balancing parallel assembly lines via Ant Colony Optimization". July 2009, pp. 506–511. DOI: 10.1109/ICCIE.2009.5223867.

[163] Y. Kara, H. Gökçen, and Y. Atasagun. "Balancing parallel assembly lines with precise and fuzzy goals" Mar 2010, pp. 1685–1703. DOI: 10.1080/00207540802534715.

[164] L. Ozbakir, A. Baykasoglu, B. Gorkemli, and L. Gorkemli. "Multiple-colony ant algorithm for parallel assembly line balancing problem". Apr 2011, pp. 3186–3198. DOI: 10.1016/j.asoc.2010.12.021.

[165] Z. A. Çil, S. Mete, E. Özceylan, and K. Ağpak. "A beam search approach for solving type II robotic parallel assembly line balancing problem". Dec 2017, pp. 129–138. DOI: 10.1016/j.asoc.2017.07.062.

[166] Z. A. Çil, Z. Li, S. Mete, and E. Özceylan. "Mathematical model and bee algorithms for mixed-model assembly line balancing problem with physical human–robot collaboration". Aug 2020, p. 106394. DOI: 10.1016/j.asoc.2020.106394.

[167] S. Akpınar and G. Mirac Bayhan. "A hybrid genetic algorithm for mixed model assembly line balancing problem with parallel workstations and zoning constraints". Apr 2011, pp. 449–457. DOI: 10.1016/j.engappai.2010.08.006.

[168] K. Ağpak, M. F. Yeğül, and H. Gökçen. "Two-sided U-type assembly line balancing problem". Sept 2012, pp. 5035–5047. DOI: 10.1080/00207543.2011.631599.

[169] W. Yang and W. Cheng. "Modelling and solving mixed-model two-sided assembly line balancing problem with sequence-dependent setup time". Nov 2020), pp. 6638–6659. DOI: 10.1080/00207543.2019.1683255.

[170] Z. A. Çil, S. Mete, and K. Ağpak. "Analysis of the type II robotic mixed-model assembly line balancing problem". June 2017, pp. 990–1009. DOI: 10.1080/0305215X.2016.1230208.

[171] U. Özcan, T. Kellegöz, and B. Toklu. "A genetic algorithm for the stochastic mixed-model U-line balancing and sequencing problem". Mar 2011, pp. 1605–1626. DOI: 10.1080/00207541003690090.

[172] P. Chutima and P. Chimklai. "Multi-objective two-sided mixed-model assembly line balancing using particle swarm optimization with negative knowledge". Feb 2012, pp. 39–55. DOI: 10.1016/j.cie.2011.08.015.

[173] G. R. Esmaeilian, S. Sulaiman, N. Ismail, M. Hamedi, and M. M. H. M. Ahmad. "A tabu search approach for mixed-model parallel assembly line balancing problem (type II)". 2011, p. 407. DOI: 10.1504/IJISE.2011.041803.

[174] Z. Zhang, Q. Tang, Z. Li, and L. Zhang. "Modelling and optimization of energy-efficient U-shaped robotic assembly line balancing problems". Sept 2019, pp. 5520–5537. DOI: 10.1080/00207543.2018.1530479.

[175] D. Huang, Z. Mao, K. Fang, and B. Yuan. "Combinatorial Benders decomposition for mixed-model two-sided assembly line balancing problem". Apr 2022, pp. 2598–2624. DOI: 10.1080/00207543.2021.1901152.

[176] M. Kammer Christensen, M. N. Janardhanan, and P. Nielsen. "Heuristics for solving a multi-model robotic assembly line balancing problem". Jan 2017, pp. 410–424. DOI: 10.1080/21693277.2017.1403977.

[177] I. Kucukkoc and D. Z. Zhang. "Type-E parallel two-sided assembly line balancing problem: Mathematical model and ant colony optimization based approach with optimized parameters". June 2015, pp. 56–69. DOI: 10.1016/j.cie.2014.12.037.

[178] I. Kucukkoc and D. Z. Zhang. "Mixed-model parallel two-sided assembly line balancing problem: A flexible agent-based ant colony optimization approach". July 2016, pp. 58–72. DOI: 10.1016/j.cie.2016.04.001.

[179] Z. Li, M. N. Janardhanan, A. S. Ashour, and N. Dey. "Mathematical models and migrating birds optimization for robotic U-shaped assembly line balancing problem". Dec 2019, pp. 9095–9111. DOI: 10.1007/s00521-018-3957-4.

[180] Z. Li, M. N. Janardhanan, Q. Tang, and S. Ponnambalam. "Model and metaheuristics for robotic two-sided assembly line balancing problems with setup times". Nov 2019, p. 100567. DOI: 10.1016/j.swevo.2019.100567.

[181] Z. Li, M. N. Janardhanan, Q. Tang, and P. Nielsen. "Local search methods for type I mixed-model two-sided assembly line balancing problems". Mar 2021, pp. 111–130. DOI: 10.1007/s12293-020-00319-0.

[182] N. Manavizadeh, N.-s. Hosseini, M. Rabbani, and F. Jolai. "A Simulated Annealing algorithm for a mixed model assembly U-line balancing type-I problem considering human efficiency and Just-In-Time approach". Feb 2013, pp. 669–685. DOI: 10.1016/j.cie.2012.11.010.

[183] J. Mukund Nilakantan and S. Ponnambalam. "Robotic U-shaped assembly line balancing using particle swarm optimization". Feb 2016, pp. 231–252. DOI: 10.1080/0305215X.2014.998664.

[184] M. Rabbani, Z. Mousavi, and H. Farrokhi-Asl. "Multi-objective meta-heuristics for solving a type II robotic mixed-model assembly line balancing problem". Oct 2016, pp. 472–484. DOI: 10.1080/21681015.2015.1126656.

[185] M. Rabbani, R. Siadatian, H. Farrokhi-Asl, and N. Manavizadeh. "Multi-objective optimization algorithms for mixed-model assembly line balancing problem with parallel workstations". Dec 2016. Ed. by Z. Zhou, p. 1158903. DOI: 10.1080/23311916.2016.1158903.

[186] Z. Zhang, Q. Tang, and L. Zhang. "Mathematical model and grey wolf optimization for low-carbon and low-noise U-shaped robotic assembly line balancing problem". Apr 2019, pp. 744–756. DOI: 10.1016/j.jclepro.2019.01.030.

[187] M. F. Yegul, K. Agpak, and M. Yavuz. "A NEW ALGORITHM FOR U-SHAPED TWO-SIDED ASSEMBLY LINE BALANCING". June 2010, pp. 225–241. DOI: 10.1139/tcsme-2010-0014.

[188] A. Yadav and S. Agrawal. "Mathematical model for robotic two-sided assembly line balancing problem with zoning constraints". Feb 2022, pp. 395–408. DOI: 10.1007/s13198-021-01284-8.

[189] A. Yadav, R. Kulhary, R. Nishad, and S. Agrawal. "Parallel two-sided assembly line balancing with tools and tasks sharing". Nov 2019, pp. 833–846. DOI: 10.1108/AA-02-2018-025.

[190] P. Tapkan, L. Özbakır, and A. Baykasoglu. "Bee algorithms for parallel two-sided assembly line balancing problem with walking times". Feb 2016, pp. 275–291. DOI: 10.1016/j.asoc.2015.11.017.

[191] D. Sparling and J. Miltenburg. "The mixed-model U-line balancing problem". Feb 1998, pp. 485–501. DOI: 10.1080/002075498193859.

[192] A. Roshani and F. Ghazi Nezami. "Mixed-model multi-manned assembly line balancing problem: a mathematical model and a simulated annealing approach". Feb 2017, pp. 34–50. DOI: 10.1108/AA-02-2016-016.

[193] P. Chutima and N. Yothaboriban. "Multi-objective mixed-model parallel assembly line balancing with a fuzzy adaptive biogeography-based algorithm". 2017, p. 90. DOI: 10.1504/IJISE.2017.083182.

[194] Z. Zhang and W. Cheng. "Improved Heuristic Procedure for Mixed-Model U-line Balancing Problem with Fuzzy Times". 2015, pp. 395–406. DOI: 10.1007/978-3-662-44674-4_37.

[195] Z. Zhang, Q. Tang, and M. Chica. "A robust MILP and gene expression programming based on heuristic rules for mixed-model multi-manned assembly line balancing". Sept 2021, p. 107513. DOI: 10.1016/j.asoc.2021.107513.

[196] J. Mukund Nilakantan, G. Q. Huang, and S. Ponnambalam. "An investigation on minimizing cycle time and total energy consumption in robotic assembly line systems". Mar 2015, pp. 311–325. DOI: 10.1016/j.jclepro.2014.11.041.

[197] X.-S. Yang and Suash Deb. "Cuckoo Search via Lévy flights". 2009, pp. 210–214. DOI: 10.1109/NABIC.2009.5393690.

[198] K.-L. Du and M. N. S. Swamy. Search and Optimization by Metaheuristics. Cham: Springer International Publishing, 2016. DOI: 10.1007/978-3-319-41192-7.

[199] X.-S. Yang. Nature-inspired optimization algorithms. First edition. OCLC: ocn866615538. Amsterdam; Boston: Elsevier, 2014. 263 pp.

[200] J. M. Nilakantan, S. G. Ponnambalam, and G. Q. Huang. "Minimizing energy consumption in a U-shaped robotic assembly line." 2015 International Conference on Advanced Mechatronic Systems (ICAMechS). Beijing, China: IEEE, Aug. 2015, pp. 119–124. DOI: 10.1109/ICAMechS.2015.7287140.

[201] R. Martí, P. M. Pardalos, and M. G. C. Resende, eds. Handbook of Heuristics. Cham: Springer International Publishing, 2018. DOI: 10.1007/978-3-319-07124-4.

[202] F. Glover and G. A. Kochenberger. Handbook of metaheuristics. International series in operations research & management science 57. Boston Dordrecht London: Kluwer Academic Publ, 2003.

[203] M. Prais and C. C. Ribeiro. "ReactiveGRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment". Aug 2000, pp. 164–176. DOI: 10.1287/ijoc.12.3.164.12639.

[204] R. Alvarez-Valdes, F. Parreño, and J. Tamarit. "ReactiveGRASP for the strip-packing problem". Apr 2008, pp. 1065–1083. DOI: 10.1016/j.cor.2006.07.004.

[205] Y. Deng and J. F. Bard. "A reactiveGRASP with path relinking for capacitated clustering". Apr 2011, pp. 119–152. DOI: 10.1007/s10732-010-9129-z.

[206] T. S. Jaikishan and R. Patil. "A ReactiveGRASP Heuristic Algorithm for Vehicle Routing Problem with Release Date and Due Date Incurring Inventory Holding Cost and Tardiness Cost." 2019 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM). Macao, Macao: IEEE, Dec. 2019, pp. 1393–1397. DOI: 10.1109/IEEM44572.2019.8978851.

[207] J. Bautista, R. Suarez, M. Mateo, and R. Companys. "Local search heuristics for the assembly line balancing problem with incompatibilities between tasks". 2000 ICRA. IEEE International Conference on Robotics and Automation. Vol. 3. San Francisco, CA, USA: IEEE, 2000, pp. 2404–2409. DOI: 10.1109/ROBOT.2000.846387.

[208] M. Chica, O. Cordón, S. Damas, and J. Bautista. "A Multiobjective GRASP for the 1/3 Variant of the Time and Space Assembly Line Balancing Problem". 2010, pp. 656–665. DOI: 10.1007/978-3-642-13033-5_67.

[209] F. F. Boctor. "A Multiple-Rule Heuristic for Assembly Line Balancing". Jan 1995, p. 62. DOI: 10.2307/2583836.

[210] A. V. Raj, J. Mathew, P. Jose, and G. Sivan. "Optimization of Cycle Time in an Assembly Line Balancing Problem". 2016, pp. 1146–1153. DOI: 10.1016/j.protcy.2016.08.231.

[211] H. Du, Z. Wang, W. Zhan, and J. Guo. "Elitism and Distance Strategy for Selection of Evolutionary Algorithms". 2018, pp. 44531–44541. DOI: 10.1109/ACCESS.2018.2861760.

[212] M. G. C. Resende and C. C. Ribeiro. "Greedy Randomized Adaptive Search Procedures: Advances and Extensions". 2019, pp. 169–220. DOI: 10.1007/978-3-319-91086-4_6.

[213] R. O. Edokpia and F. Owu. "Assembly Line Re-Balancing Using Ranked Positional Weight Technique and Longest Operating Time Technique: A Comparative Analysis". Sept 2013, pp. 568–578. DOI: 10.4028/www.scientific.net/AMR.824.568.

[214] N. Boysen, M. Kiel, and A. Scholl. "Sequencing mixed-model assembly lines to minimize the number of work overload situations". Aug 2011, pp. 4735–4760. DOI: 10.1080/00207543.2010.507607.

[215] W. Grzechca, ed. Assembly Line - Theory and Practice. InTech, Aug. 17, 2011. DOI: 10.5772/824.

[216] N. T. Thomopoulos. Assembly Line Planning and Control. Cham: Springer International Publishing, 2014. DOI: 10.1007/978-3-319-01399-2.