

DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

**AN ADAPTIVE LARGE NEIGHBOURHOOD
SEARCH ALGORITHM FOR TYPE-II
ASSEMBLY LINE BALANCING PROBLEMS**

by
Hüseyin Ali SÖNMEZ

March, 2021

İZMİR

**AN ADAPTIVE LARGE NEIGHBOURHOOD
SEARCH ALGORITHM FOR TYPE-II
ASSEMBLY LINE BALANCING PROBLEMS**

**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of
Science in Industrial Engineering, Industrial Engineering Program**

**by
Hüseyin Ali SÖNMEZ**

March, 2021

İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “**AN ADAPTIVE LARGE NEIGHBOURHOOD SEARCH ALGORITHM FOR TYPE-II ASSEMBLY LINE BALANCING PROBLEMS**” completed by **HÜSEYİN ALİ SÖNMEZ** under supervision of **ASSOC. PROF. DR. ŞENER AKPINAR** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Şener AKPINAR

Supervisor

Assoc. Prof. Dr.
Aslan Deniz KARAOĞLAN

(Jury Member)

Assist.Prof. Dr.
Şebnem DEMİRKOL AKYOL

(Jury Member)

Prof. Dr. Özgür ÖZÇELİK

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Assoc. Prof. Dr. Şener AKPINAR for the continuous support, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. He steered me in the right direction whenever he thought I needed it. I could not have imagined having a better advisor and mentor for thesis.

And also, I would like to thank my spouse, Göknil SÖNMEZ for supporting me spiritually throughout writing this thesis, continuous encouragement and my life in general.

Hüseyin Ali SÖNMEZ

AN ADAPTIVE LARGE NEIGHBOURHOOD SEARCH ALGORITHM FOR TYPE-II ASSEMBLY LINE BALANCING PROBLEMS

ABSTRACT

The need for reducing the production costs that are increasing with competition conditions and customer-oriented production strategies reveals the necessity of redesigning the production phase, especially the assembly lines in this direction. In response to this need, type-II assembly line balancing problem which is used for redesigning an existing assembly line arises.

In this study, an adaptive large neighborhood search (ALNS) algorithm is proposed for the type-II simple assembly line balancing problem (SALBP-II) and the version where this problem is enriched by considering the sequence-dependent setup times between operations (SALBPs-II). The effectiveness of the large neighborhood search (LNS) algorithm has been shown by studies on vehicle routing and scheduling problems. Vehicle routing problems aim to find the customer visit rankings that will make the minimum distance traveled as well as to find the optimum matching between a certain number of routes and a certain number of customers, while the SALBP-II aims to determine the assignment of a certain number of assembly tasks to a certain number of stations, which minimizes the cycle time. SALBPs-II aims to determine the task performing sequences that will minimize the workload at each station due to the sequence dependent setup times in addition to assign the assembly operations to the stations. This structural similarity between the vehicle routing problem and the SALBP-II and SALBPs-II reveals the potential of LNS to be an effective algorithm on both of these problems. In this respect, an ALNS algorithm for SALBP-II and SALBPs-II was developed and its performance was tested. The results show the performance of the proposed ALNS algorithm for SALBP-II and SALBPs-II is satisfactory.

Keywords: Algorithms, heuristics, assembly line balancing, optimization, large neighbourhood search

TİP-II MONTAJ HATTI Dengeleme problemleri için bir Adaptif büyük komşuluk arama algoritması

ÖZ

Her geçen gün artan rekabet koşullarının ve müşteri odaklı üretim stratejilerinin arttırdığı üretim maliyetlerini azaltmak ihtiyacı, üretim aşamalarının, özellikle montaj hatlarının bu doğrultuda yeniden tasarlanması gerekliliğini ortaya çıkarmaktadır. Bu ihtiyaca cevaben mevcut bir montaj hattının yeniden tasarlanmasında kullanılan tip-II montaj hattı dengeleme problemi ortaya çıkmaktadır.

Bu çalışmada basit tip-II montaj hattı dengeleme problemi (BMHDP-II) ile bu problemin operasyonlar arasında sıra bağımlı hazırlık sürelerinin göz önüne alınarak zenginleştirildiği versiyonu (BMHDPs-II) için bir adaptif büyük komşuluk arama (ABKA) algoritması önerilmektedir. Büyük komşuluk arama (BKA) algoritmasının etkinliği araç rotalama ve çizelgeleme problemleri üzerinde ilgili yazında yapılan çalışmalar ile ortaya konmuştur. Araç rotalama problemleri belirli sayıdaki rotalar ile belirli sayıdaki müşteriler arasındaki optimum eşleşmeyi ve toplam kat edilen mesafeyi minimum yapacak müşteri ziyaret sıralamalarını bulmayı amaçlarken, BMHDP-II belirli sayıdaki montaj operasyonlarının belirli sayıdaki istasyonlara çevrim zamanını minimum yapan atamasını bulmayı amaçlamaktadır. BMHDPs-II ise montaj operasyonlarının istasyonlara atanması yanında sıra bağımlı hazırlık süreleri nedeniyle her bir istasyondaki iş yükünü en azlayacak operasyon yürütme sıralamalarını bulmayı amaçlamaktadır. Araç rotalama problemi ile BMHDP-II ve BMHDPs-II arasındaki bu yapısal benzerlik BKA'nın bu her iki problem içinde etkin bir çözüm yöntemi olma potansiyelini ortaya çıkarmaktadır. Bu doğrultuda BMHDP-II ve BMHDPs-II için bir ABKA algoritması geliştirilmiş ve performansı test edilmiştir. Elde edilen sonuçlar önerilen ABKA algoritmasının BMHDP-II ve BMHDPs-II için tatmin edici performansını ortaya koymaktadır.

Anahtar kelimeler: Algoritmalar, sezgiseller, montaj hattı dengeleme, optimizasyon, büyük komşuluk arama

CONTENTS

	Page
M.Sc THESIS EXAMINATION RESULT FORM.....	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT.....	iv
ÖZ	v
LIST OF FIGURES	ix
LIST OF TABLES	x
 CHAPTER ONE - INTRODUCTION	 1
1.1 Importance of the Thesis	1
1.2 Framework of the Thesis	3
1.3 Outline of the Thesis	5
 CHAPTER TWO - ASSEMBLY LINE BALANCING PROBLEM.....	 6
2.1 Assembly Lines	6
2.1.1 Terminology of Assembly Line Production	6
2.1.2 Classification of Assembly Lines	9
2.1.3 Additional Features of Assembly Lines	13
2.1.4 Performance Measures of Assembly Lines	15
2.2 Simple Assembly Balancing Problem (SALBP).....	16
2.2.1 Mathematical Formulation of SALBP-II and SALBPs-II.....	17
2.2.2 Solution approaches for SALB-II.....	22

**CHAPTER THREE - AN OVERVIEW ON THE LARGE NEIGHBOURHOOD
SEARCH ALGORITHM 23**

3.1 Features of Heuristic Algorithms	23
3.2 Large Neighbourhood Search Algorithm	23
3.2.1 Removal Operator.....	26
3.2.2 Insertion Operator.....	27
3.3 Adaptive Large Neighbourhood Search Algorithm	27
3.3.1 Choosing Methods as Adaptive	28
3.4 Literature Review	28

**CHAPTER FOUR - AN ADAPTIVE LARGE NEIGHBOURHOOD SEARCH
ALGORITHM FOR SALBP-II 31**

4.1 ALNS Algorithm for SALBP-II (and SALBPs-II)	31
4.1.1 Parameter Definitions	33
4.1.2 Initial Solution	37
4.1.3 Removal Operator.....	40
4.1.3.1 Effect to Total Idle Time of Tasks to Be Removed (ETIT).....	42
4.1.3.2 Greatest Ranked Positional Weight Method (GRPW)	42
4.1.3.3 Greatest Ranked Sum of Previous Job Durations Method (GRPWP).....	43
4.1.3.4 Greatest Ranked Sum of Previous and Successors Job Durations Method (GRPWA).....	43
4.1.3.5 Random Ranking Method	43
4.1.4 Insertion Operator.....	44
4.1.4.1 Regret-2 (Regret2)	46
4.1.4.2 Longest Processing Time (LPT)	46

4.1.4.3 Shortest Processing Time (SPT)	46
4.1.4.4 Greatest Number of Immediate Successors (GNIS)	47
4.1.4.5 Greatest Number of Successors (GNS).....	47
4.1.4.6 Random Priority (RND).....	47
4.1.4.7 Smallest Task Number (STN).....	48
4.1.4.8 Greatest Ranked Positional Weight (GRPW)	48
4.1.4.9 Greatest Average Ranked Positional Weight (GARPW).....	48
4.1.4.10 Smallest Upper Bound (SUB).....	48
4.1.4.11 Smallest (Upper Bound Divided by the # of Successors) (S_UB_NS).....	49
4.1.4.12 Greatest (Processing Time Divided by the Upper Bound) (G_PT_UB).....	49
4.1.4.13 Smallest Lower Bound (SLB).....	50
4.1.4.14 Minimum Slack (MSLK).....	50
4.1.4.15 Minimum (# of Successors Divided by Task Slack) (M_NS_SLK)	50
4.1.4.16 Greatest Number of Immediate Predecessors (GNIP)	51
4.1.5 Solution Acceptance	51
4.2 Computational Study	52
4.2.1 Results of the SALBP-II.....	53
4.2.2 Results of the SALBPs-II	67
CHAPTER FIVE - CONCLUSION	79
REFERENCES.....	81

LIST OF FIGURES

	Page
Figure 2.1 Precedence Relationships	8
Figure 2.2 Classification of AL's based on model structure	10
Figure 2.3 Assembly lines: (a) single-model, (b) mixed-model, and (c) multi model.....	11
Figure 2.4 Classification of assembly line balancing literature	12
Figure 2.5 Binary integer programming model for SALBP-II	18
Figure 2.6 Binary integer programming model for SALBPs-II.....	20
Figure 2.7 Solution approaches for SALBP-II.....	22
Figure 3.1 Chosing customers to be removed.....	26
Figure 3.2 Remove customers from the routes	26
Figure 3.3 Rinsert customers to the routes.....	26
Figure 4.1 The flow of the proposed algorithm	33
Figure 4.2 Reduce Rate	36
Figure 4.3 Solution representation	39
Figure 4.4 Indexes of the tasks in station.....	39

LIST OF TABLES

	Page
Table 2.1 Notations used in this paper	17
Table 4.1 Reorder methods used in insertion operator	45
Table 4.2 Parameters of the proposed ALNS Algorithm for SALBP-II.....	54
Table 4.3 Results for the Buxey problem set	54
Table 4.4 Results for the Gunther problem set	55
Table 4.5 Results for the Kilbridge problem set	55
Table 4.6 Results for the Lutz1 problem set	56
Table 4.7 Results for the Sawyer problem set.....	56
Table 4.8 Results for the Lutz2 problem set	57
Table 4.9 Results for the Tonge problem set	58
Table 4.10 Results for the Arcus1 problem set.....	59
Table 4.11 Results for the Arcus2 problem set.....	60
Table 4.12 Results for the Warnecke problem set	61
Table 4.13 Results for the Wee-Mag problem set	62
Table 4.14 Results for the Lutz3 problem set	63
Table 4.15 Results for the Mukherje problem set.....	64
Table 4.16 Results for the Barthold problem set	65
Table 4.17 Results for the Hahn problem set.....	65
Table 4.18 Results for the Barthold2 problem set	66
Table 4.19 Summary of the computational results	67
Table 4.20 Results for the Jackson problem set with LV setup times	68
Table 4.21 Results for the Mitchell problem set with LV setup times	69
Table 4.22 Results for the Heskiaoff problem set with LV setup times	69
Table 4.23 Results for the Lutz1 problem set with LV setup times.....	70

Table 4.24 Results for the Sawyer problem set with LV setup times	70
Table 4.25 Results for the Wee-Mag problem set with LV setup times	71
Table 4.26 Results for the Tonge problem set with LV setup times	72
Table 4.27 Results for the Arcus1 problem set with LV setup times	73
Table 4.28 Summary of computational results for the SALBPs-II (LV)	73
Table 4.29 Results for the Jackson problem set with HV setup times	74
Table 4.30 Results for the Mitchell problem set with HV setup times	74
Table 4.31 Results for the Heskiaoff problem set with HV setup times	74
Table 4.32 Results for the Lutz1 problem set with HV setup times	75
Table 4.33 Results for the Arcus1 problem set with HV setup times	75
Table 4.34 Results for the Wee-Mag problem set with HV setup times	76
Table 4.35 Results for the Tonge problem set with HV setup times	77
Table 4.36 Results for the Sawyer problem set with HV setup times	77
Table 4.37 Summary of the computational results for the SALBPs-II (HV)	78

CHAPTER ONE

INTRODUCTION

1.1 Importance of the Thesis

Today, producing the same quality product with lower costs is one of the most important issues for producers. One of the easiest way to achieve this goal is to make mass production and reduce costs. The assembly lines come to the fore in the mass production systems. Assembly type production systems divide the final product obtaining process into assembly tasks and assign these tasks to workstations established on a line while considering precedence relations among them. There is flow type production in the assembly lines. Production is done by performing some tasks respectively from the first station to the last station and the final product is obtained when the last task is performed at the last station. There are precedence relationships between tasks, and tasks in the production process should be performed according to this precedence relationships.

The problem trying to equalization of the sum of the durations of the tasks performed at each workstation is the Assembly Line Balancing Problem (ALBP). In other words, the problem of deciding how to group tasks (that have processing time and precedence relationships that shows which task to do first) in workstations is called ALBP.

The AL has become an important topic with the usage in the automobile industry in the 20th century. Although the first usage of assembly lines was thought to be in the 20th century, the assembly lines were also used at ancient times. Terracotta Army located in China, which is known as the army of soldiers protecting the grave, was produced with the logic of assembly line in 200 BC. The archaeological studies revealed that the limbs and bodies of soldier statues were produced separately and subsequently assembled. However, assembly lines have become widespread with the usage in the automotive industry and moving belt innovation in the early 20th century. This change, which started in the automotive sector, because of reducing production costs significantly, has changed the production systems of all other sectors until today

and AL production has been adopted worldwide. With the usage of AL in the production process, low costed and large volume standard products have been possible to produce.

Although it provides great benefits in reducing costs, AL installations require large capital investment. The benefit of the AL to be more important than this investment depends on whether the line is balanced. It is a very important issue to find the optimum line balance both in the first installation and in the re-balancing of the assembly lines. The changes in the production process disrupt the line balance and rebalancing is required. Developing technology and increasing competition conditions can lead to this changes in the production process. Mechanization in production, the usage of new production techniques, robot usage in production, etc. can be given as examples to the effect of developing technology on the production process, while changes in the final product design, changes in the properties of the final product, the need to produce the final product with different materials or at different times etc. can be given as examples to the effect of increasing competition conditions on the production processes.

As assembly lines become more important in production systems, this issue has also started to be addressed in the literature. After Salveson (1955) published the first article on assembly line balancing, this topic has been a frequent topic of interest in the academic community (Kriengkorakot & Pianthong, 2007). In the literature, ALBP has been studied as two types. ALBP can aim minimization of the number of workstations according to the given cycle time (type 1) or minimization of the cycle time according to the given station number (type 2). For both types of ALBP, exact methods aimed at achieving directly an optimal solution or approximation strategies aimed approximation the best solution as iteratively were used. The exact methods do not give results for large volume problems due to NP Hard structure of the problem (Scholl, 1999). Many heuristic algorithms have been used as approximation strategies. Simple heuristic algorithms or meta heuristic algorithms were used as approach strategy to solve the problem. None of the approximation methods guarantee the optimum solution, however, these methods provide satisfactory results in reasonable time limits. Due to the NP hard structure of the problem, approximation strategies are

frequently studied in the literature. It cannot be said that any of these approach strategies provide the best solution on ALBP. Therefore, it is still important to develop existing solution approaches for AL or introduce new solution approaches.

Studies which aim to solve type-2 ALBP usually solve the problem iteratively by solving type-1 problem in each iteration. In this method, the algorithm determines a cycle time in each iteration and solves the problem as type-1 for determined cycle time. This method aims to minimize cycle time by iteratively solving type-1 version of the problem for the cycle time updated (re-determined) in each iteration. Differently in this study, a method has been developed to directly solve type-2 ALBP.

1.2 Framework of the Thesis

In the simplest terms, assembly line is to obtain the final product by performing some tasks on the workpiece flowing from one station to the other station respectively. In this way, standard large volume productions can be realized. Although the assembly lines are more complex in real-life conditions, the new solution methods are often studied on Simple Assembly Line Balancing Problem (SALBP). This study proposes an Adaptive Large Neighborhood Search (ALNS) algorithm for simple assembly line balancing problem. The simple assembly line balancing problem can be defined as a simple form of the assembly line balancing problem, which they are assumed that a single model production is made, there is paced flow on a straight line, there are no subassembly lines, the processing times are deterministic and there are no assignment restrictions (Baybars, 1986).

SALBP can aim minimization of the number of workstations according to the given cycle time (SALBP-I) or minimization of the cycle time according to the given station number (SALBP-II.). Finding the smallest cycle time is the most focused aim on re-balancing of lines that needs to be done due to developing technology and increasing competition conditions. Therefore, in this study, type-2 simple assembly line balancing problem (SALBP-II) was studied. Hence, the ALNS algorithm has been adopted to solve the SALBP-II problem. In order to measure the success of the developed ALNS

algorithm, the algorithm was run with the SALBP benchmark data sets retrieved <https://assembly-line-balancing.de/> and the results were evaluated.

In SALBP-II a preparation time (the time it takes to get the equipment to be used, the time it takes to return to the working position after the displacement with the moving line, etc.) may be required to perform other tasks after a task is performed. This required preparation time (setup time) is not constant and may vary according to task performed and task will be performed. In this case, sequence dependent setup times between tasks should be considered in solving of the problem. In this case, sequence -dependent setup times between tasks should be considered in problem solving. These problems are called Simple Assembly Line Balancing Problem with Sequence Dependent Setup Times (SALBPs-II). In this problem, in addition to the above assumptions, it is assumed that there are sequence dependent setup times between tasks.

The Large Neighborhood Search (LNS) algorithm, which is adapted for the solution of the SALBP-II problem, is an algorithm developed for the solution of Vehicle Routing Problems (VRP). This algorithm is used for large volume vehicle routing problems which cannot be solved by exact methods and provide very successful results (Koç & Laporte, 2018; Nagata & Bräysy 2009). After the LNS algorithm was adapted for VRP, an adaptive version of this algorithm was proposed (Ropke & Pisinger, 2006).

The VRP aims to find the optimum match between a certain number of routes and a certain number of customers, which will make the total distance traveled minimum. In this problem, customers are assigned to certain routes, and there is a specific ranking between the customers assigned to each route. The subject of this study, SALBP-II, deals with the problem of assigning a certain number of tasks to a certain number of stations. In the SALBPs-II problem, which takes into account the sequence-dependent setup times between tasks, the sequence among the tasks assigned to each station is also important. There is a structural similarity between assigning customers to routes and creating a sequence by assigning tasks to stations. This structural similarity suggests that the ALNS algorithm will give successful results for SALBP-II or

SALBPs-II. Previously, it has been proposed that LNS algorithm can be used for SALBP-II (Akpınar, 2017). In this study, this idea was expanded and an Adaptive Large Neighborhood Search Algorithm (ALNS) was applied to the Type-2 Assembly Line Balancing Problem.

1.3 Outline of the Thesis

The design of this study will be as follows. In the second chapter, the Assembly Line Balancing Problem is mentioned. In the third chapter, the Large Neighbourhood Search algorithm is explained. In the fourth chapter, the improved Adaptive Large Neighbourhood Search Algorithm for the type-2 Simple Assembly Line Balancing Problem is explained in detail and performance evaluation tests are executed. In the fifth chapter results are discussed.

CHAPTER TWO

ASSEMBLY LINE BALANCING PROBLEM

2.1 Assembly Lines

Assembly lines are a manufacturing process where interchangeable parts are added together to obtain the final product (Akpınar, 2009). Assembly lines are the most widely used manufacturing method in mass production because of reducing production costs, obtaining finished products in short periods, non-qualified workers can be used in the production process, making it possible to the use of robot instead of worker in production by dividing the production process into small elements.

Changing the existing production system to the assembly line production system is a long-term decision requiring large capital investment. For this reason, the balance of the assembly line is an important issue. Most of the articles dealing with the assembly line were concerned with line balancing problem. The reader may refer to the comprehensive review studies of Ghosh & Gagnon (1989), Erel & Sarin (1998), Becker & Scholl (2006), Boysen, Fliedner, & Scholl (2008), Battaïa & Dolgui (2013) and Sivasankaran & Shahabudeen (2014) about the literature on the assembly line balancing problems.

2.1.1 Terminology of Assembly Line Production

Some terms used for the assembly line are described below.

Assembly, is the process of collecting and adding various pieces together to obtain the final product.

Workpiece, is an unfinished product which is processed in the production process.

An operation (task), is the smallest divisible part of the total assembly workload. The production process is divided into pieces and the smallest workpiece that cannot be divided is called a task. Tasks cannot be divided into smaller pieces without extra

work. The time required for each task to be performed on the workpiece is called **task time (processing time)**.

Setup time, is preparation time required to perform other tasks after a task performed.

A (work) station, is the part where a certain number of tasks are performed in the production process. If the tasks are performed by the human worker, the worker who performs the tasks is called the **operator**.

Station time (Station load), is the sum of the processing times of the tasks to be performed on each station. If there is a setup time between tasks, in addition to the processing time of the tasks, the setup times between the tasks are taken into account when calculating the station time.

Cycle time, is the time between two finished products produced. Cycle time is the greatest station time. No station time can be greater than the cycle time.

Idle time, is the difference between station time and cycle time. If the time required to perform tasks assigned to a station is less than the cycle time, idle time will be occurred on that station. In order to move to the next workpiece, the station waits for the previous or next stations to finish their workload. This waiting time is called station idle time. The sum of the idle time at all stations is considered to be the **Total Idle Time (TIT)**. Ideally, the total idle time is expected to be zero.

Precedence diagram, is a diagram that shows which tasks must be completed before a task will be performed. In some cases, for technical reasons, some tasks must be completed before a task will be performed. These relationships are shown in the precedence diagram. The precedence relationship diagram consists of nodes and arcs. While nodes indicate tasks, if task i is required to be completed to perform the task j (if task i is the predecessor task of task j .), the arc drawn from the node i to the node j indicates the precedence relationship. The set of predecessors of task j , which associated with the arcs to the node j , called **Immediate predecessors** of task j . In order

to perform task j immediate predecessors and all other predecessors of task j must be completed. After task i is performed, the task j , which is connected with the arcs to the task i that can be performed, is called the **successor** of the task i . Similar to the predecessor relationship, there is a set of successors for each task, including **Immediate successors** and other successors. Figure 2.1 shows a sample Precedence diagram of a production with 10 tasks. As can be seen in this diagram, each task i can have multiple successors, and each task j may have multiple predecessors. The task 6 is the direct predecessor of the tasks 7, 8 and 9, while the tasks 7, 8 and 9 are direct predecessors of task 10.

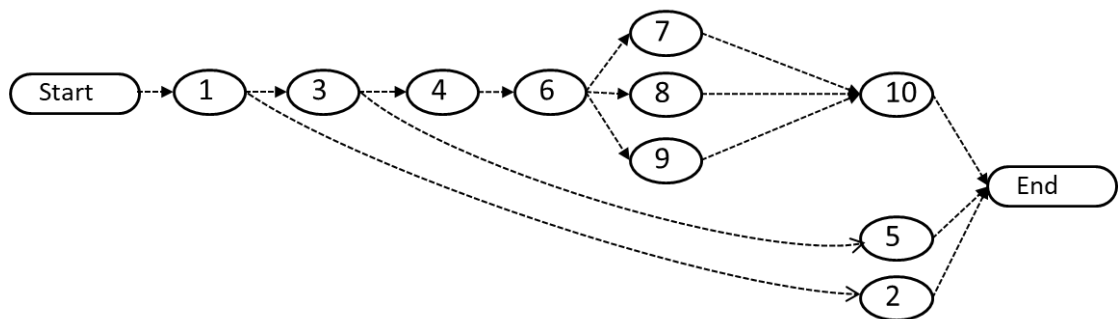


Figure 2.1 Precedence Relationships

Line balance (feasible task assignment), is a possible solution of line balancing problem. For a feasible solution, tasks must be assigned to stations without division. Each task must be assigned to a station. The assignment should be appropriate for the precedence relationships. To do this, if a task i is the predecessor of task j , the task i must be assigned to the same or previous stations which task j is assigned. If the task i is assigned to the same station with the task j , the position where task i is assigned must be before the position which task j is assigned. If there is a sequence dependent setup time between tasks, the position of tasks in a station (sequence the task at the station) becomes important, these positions do not matter if you do not have setup time. If there are not setup times it is sufficient to set the tasks at the stations without considering the positions of the tasks in the stations.

2.1.2 Classification of Assembly Lines

In the classification of the assembly lines, the structure and objectives of the model, the number of different products produced, the diversity in the processing time, the line layout oriented classification schemes are used. (Scholl, 1999; Kriengkorakot & Pianthong, 2007; Battaia & Dolgui, 2012; Boysen, Fliedner, & Scholl 2006). These classification schemes are explained below one by one.

According to the model structure and purpose, assembly line balancing problem can be examined under two main groups as **Simple Assembly Line Balancing Problem (SALBP)** and **Generalized Assembly Line Balancing Problem (GALBP)**. (Becker & Scholl, 2006; Scholl & Becker, 2006). The simple assembly line balancing problem can be defined as a simple version of assembly line balancing problem, which they are assumed that a single model is produced, there is paced flow on a straight line, there are no subassembly lines, there are no zoning restrictions (Baybars, 1986). The generalized assembly line is the assembly line balancing problem obtained by relaxation on any assumptions in the simple assembly line balancing problem. Mixed Model Assembly Line Balancing Problem (**MMALBP**), U-Type Assembly Line Balancing Problem (**UALBP**) and Assembly Line Balancing Problem with stochastic processing times can be given as examples of GALBP. SALBP is divided into 5 parts as **SALBP-I**, **SALBP-II**, **SALBP-III**, **SALBP-E** and **SALBP-F** according to the model objectives. SALBP-1 is the problem (type 1) which aims at minimizing the number of stations based on a certain cycle time. SALBP-II aims to find the lowest cycle time under a certain number of stations (type-2). SALBP-III aims to smooth workload (Hazir, Agi & Guêrin, 2019; Azizoğlu & İmat, 2017). SALBP-E tries to provide these two objectives at the same time simultaneously. SALBP-F only aims to find a feasible solution. GALBP can be classified as Multi-Mixed Model Assembly Line Balancing Problem (MMALBP), U-Type Assembly Line Balancing Problem (UALBP) and others. This classification can be seen in figure 2.2. Traditional studies and developed new methods of solution frequently referred to SALBP in literature.

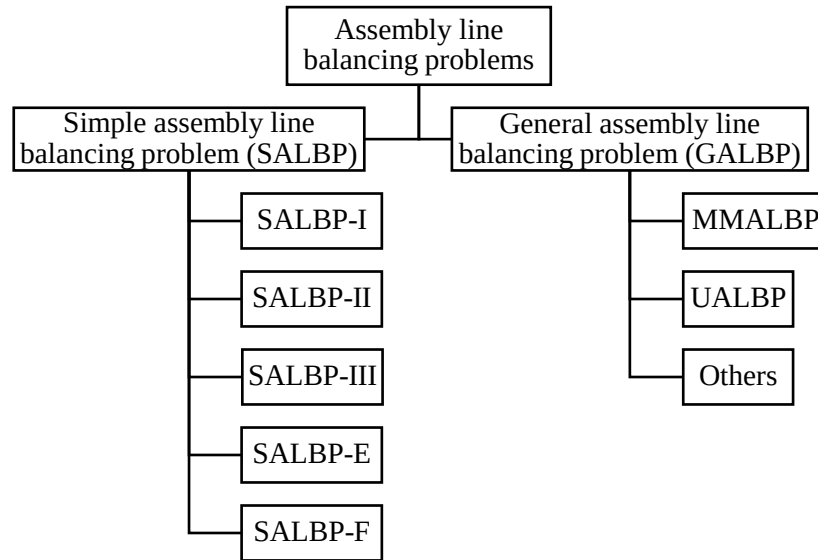


Figure 2.2 Classification of AL's based on model structure. (Becker & Scholl, 2006; Scholl & Becker, 2006)

The assembly lines can be even classified according to the number and variety of assembled models (Scholl, 1999; Ghosh & Gagnon, 1989). According to the number of different products produced, the assembly lines are divided into **Single Model Assembly Line**, **Multi-Model Assembly Line** and **Mixed Model Assembly Line** (Scholl, 1999). While there is homogeneous uniform production in single model assembly lines, it is possible to produce more than one kind of product in multi model and mixed model assembly line. In the multi-model assembly line, several models are assembled in batches in AL and a setup time may be required between batches. These three conditions can be seen in figure 2.3 (Scholl, 1999). According to this classification, SALBP is in the single model assembly line group.

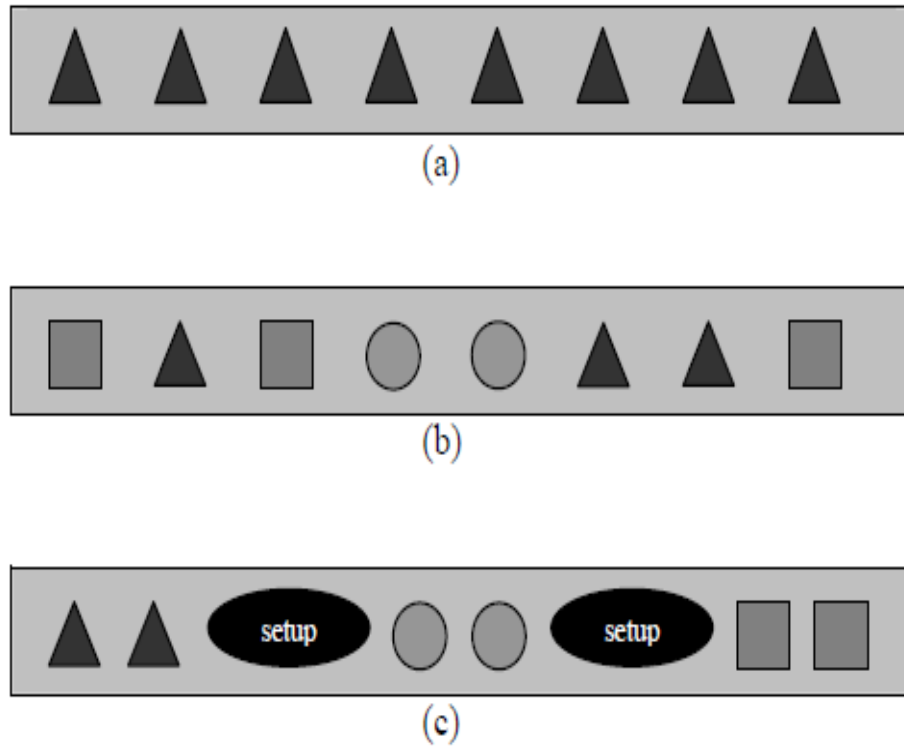


Figure 2.3 Assembly lines: (a) single-model, (b) mixed-model, and (c) multi-model

Assembly lines can be classified according to the variety of processing time. Processing times can be deterministic, stochastic, hidden or dynamic. Processing times are considered to be constant if tasks are performed by machines or by qualified and experienced personnel. Such assembly lines called assembly lines with **deterministic** task times. Task times are considered to be **stochastic** if there is a variety of processing time due to the skill, motivation and ability of the operator performing the task. Hidden process time is seen in some assembly lines used in automation. If more than one tasks are performed by automation processes in a station at the same time, the workload on that station may not simply be the sum of the processing times. Therefore, the processing time is considered as **hidden**. Finally, if the processing time changes systematically due to the learning effect etc., then the processing times are called as **dynamic**. According to this classification, SALBP is in the deterministic processing time assembly line group.

In some cases, assembly lines can be classified by looking at the number of models, variation in processing time and model structure at the same time. As it can be seen in Figure 2.4, Ghosh & Gagnon, (1989) have divided the assembly lines into 4 in this way.

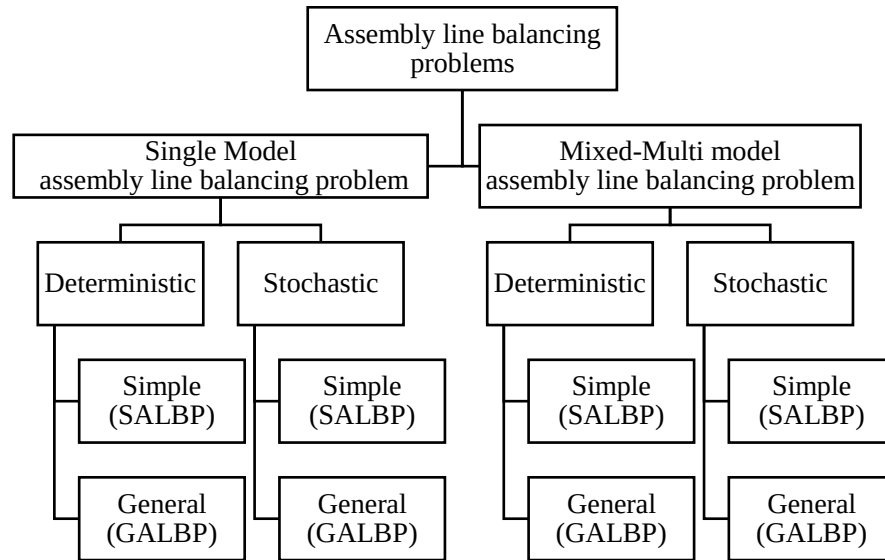


Figure 2.4 Classification of assembly line balancing literature (Ghosh & Gagnon, 1989)

Assembly lines can be classified according to their layout. Assembly lines consisting of stations lined up on a straight line are called as **straight-line assembly lines**. It is generally assumed in most of the studies that there is an operator at a station. But also Two-Sided Assembly Line Balancing Problems (TALBP), where operators work on both sides of a station (Yilmazlar et al., 2020; Bartholdi, 1993), and Multi-Manned Assembly Line Balancing Problems (M-MALBP) with multiple operators in one station (Michels, Lopes, & Magatão, 2020; A. Roshani, A. Roshani, A. Roshani, Salehi, & Esfandyari, 2013) is studied in the literature. In some cases, the assembly lines are placed in the U-shape, so that the operators standing in the middle can easily switch to multiple stations. These assembly lines are **U-type assembly lines**. When the processing time in some stations is quite large compared to other stations, these stations are duplicated in parallel and **parallel stations** are obtained. Even when necessary, the entire assembly line is replicated and a **parallel line assembly line** is obtained. Another different condition seen in the assembly line installation is the **feeder lines**. Generally, the lines where complex models are produced are divided into

subsystems so that they can be managed more easily. These subassembly lines that feed the main line are called feeder lines. According to this classification SALBP is in the straight-line assembly lines group.

2.1.3 Additional Features of Assembly Lines

The assembly lines also have some characteristics that must be addressed. These characteristics are mentioned below.

One important characteristic is the **line control** in the assembly lines. According to line independence control, the assembly lines are divided into **paced** and **asynchronous unpaced**. The flow between the stations in **paced lines** is at a constant speed. The workpiece moving between the stations usually proceeds at a constant speed with a moving belt conveyor system. A common cycle time in this case effects all stations. If the workload on a station is smaller than the cycle time, that station will remain idle when it completes its workload. In **asynchronous lines**, the workpiece moves on the station when it is finished or when it is needed at the next station. Thus, the movement of each workpiece becomes independent and the stations do not have to be restricted by cycle time. Work stations with short workloads are not blocked by stations that determine the cycle time. At SALBP, the assumption is made that the line is paced.

In some cases, assembly lines may have **assignment constraints**. In this case, the stations where some tasks can be assigned may be limited to a group of stations. Assignment constraints can be examined in terms of zoning constraints, workstation-related constraints, position-related constraints, operator-related constraints. **Zoning constraints** are constraints that indicate that different tasks must be on the same station or forbid different tasks to be on the same station. **Workstation related constraints** occur when a task can only be performed on specific stations due to technical reasons. **Position-related constraints** occur when a double-sided assembly line is used in the production process with large, non-rotatable workpieces. In this case tasks must be grouped according to the position in which they are performed. **Operator-related constraints** may occur with operator qualification and ergonomic reasons. If

performing a task requires certain qualifications, this task must be assigned to the station where the operator has sufficient skill. It may be undesirable for ergonomic reasons, to assign boring and monotonous tasks to a single operator. To avoid this, constraints can be created to assign these monotonous tasks to all operators equally.

Station type is another characteristic of assembly lines. According to station types, the assembly lines can be examined under the two-main heading as "degree of automation" and "station boundary". According to the **degree of automation**, the stations can be automated, semi-automated and manual. In **automated** stations, tasks are performed entirely by the machine, while in **semi-automated** stations a part of the tasks are performed by human operators, the human interaction is limited during the tasks performed by the machines. All tasks on **manual** stations are performed by people. In most of the studies, it is assumed that tasks are performed by a single operator from one side of the station. However, stations may be two-sided stations where the operators work on both sides of a station or multi-manned stations that have more than one operator in one station. The assembly lines are also called open stations and closed stations, depending on **station boundary**. If the workstation is restricted to prevent the operator from going out, it is called as **closed station** otherwise called as **open station**. Closed stations required when tasks are performed like painting, heating, sterile working etc.

Production volume is another characteristic for the assembly line production system. According to the production volume, the assembly line production system can be examined under two headings as **small batch production** in which large and expensive products are produced with low production rates and **mass production** of relatively small and inexpensive products are produced. Small-batch production is the assembly lines where heavy industrial products such as ships, airplanes, etc. are produced. Here the production time can take weeks or even months. In this production system, since the workpiece is sometimes difficult to displace, the workpiece remains stationary and the stations move around the workpiece.

2.1.4 Performance Measures of Assembly Lines

The goals of the assembly line balancing problem may vary. Capacity-based goals may be as well as cost / profit-oriented goals and social / organizational goals (Scholl, 1999). In some cases, these goals can be stated in the objective function of the problem, and they can be stated in the problem constraints in some cases. The most important objectives that can be considered in ALBP can be listed as follows.

Capacity-oriented goals can be listed as follows;

- ✓ Minimizing the number of stations used, according to a fixed cycle time
- ✓ Minimizing cycle time using a fixed number of stations
- ✓ Minimizing the weighted cycle time and weighted the number of stations
- ✓ Minimizing flow time
- ✓ Minimizing workload diversity
- ✓ Minimizing total idle time
- ✓ Minimizing the waiting time of workpieces

Cost-oriented goals can be listed as follows:

- ✓ Machine and equipment cost minimization
- ✓ Labor cost minimization
- ✓ Material cost minimization
- ✓ Idle time cost minimization
- ✓ Minimization cost of penalty for not meeting the demand,
- ✓ Incompletion cost minimization (The cost of workpiece that leaving the station before the tasks completed at that station)
- ✓ Setup cost minimization
- ✓ Inventory cost minimization

Profit-oriented goals are similar to cost-oriented objectives. Differently, the product price is also considered as a decision variable.

Social and organizational goals can be listed as follows;

- ✓ job enrichment objective means avoiding assignment of too many monotonous work to the same operator.
- ✓ job expansion objective is to increase the number of different tasks performed by an operator and to prevent the operator from specializing in only a small number of tasks.

Since many of the above-mentioned objectives are directly related to the objectives of "minimizing cycle time" and "reducing the number of stations", these two objectives are generally focused.

2.2 Simple Assembly Balancing Problem (SALBP)

Simple assembly line balancing problem (SALBP) is the simplified version of assembly line balancing problem. The following assumptions are made for the simple assembly line balancing problem:

- ✓ A single model is produced.
- ✓ A set of tasks is defined for the product.
- ✓ Precedence relationships are defined for tasks that need to be performed.
- ✓ Each task must be done on a station. Tasks cannot be split into multiple stations.
- ✓ There is a deterministic processing time for each task. The processing time of tasks is the same at each station.
- ✓ Each task can be performed on each station (each operator can perform each task and each station has equal technological equipment.)
- ✓ Stations are arranged on a straight line. There are no parallel stations, or feeder lines.
- ✓ There is a paced line flow.

In the literature, SALBP is examined under two types as type-1 and type-2 according to their goals. In the type-1 simple assembly line balancing problem

(SALBP-I), the aim is to use the least number of workstations according to the given cycle time, while in the type-2 simple assembly line balancing problem (SALBP-II) the aim is to find the smallest cycle time in the given number of stations. This study focuses on SALBP-II.

2.2.1 Mathematical Formulation of SABL P-II and SALBPs-II

The notations used in this document are given in Table 2.1.

Table 2.1 Notations used in this paper

Notation	Description	Formula
m	Number of Stations	
K	Set of Stations	
n	Number of Tasks	
N	Set of Tasks	
k	Indices for Station	
i,j	Indices for Tasks	
C	Cycle Time	
St _k	Station time of station k	
t _i	(processing time) the time required to perform the task i	
IS _i	set of immediate successors of task i	
IP _i	set of immediate predecessors of task i	
S _i	set of all successors of task i	
P _i	set of all predecessors of task i	
UB _i	(Upper Bound) the last station number that the task i can be assigned	$UB_i = n + 1 - [(t_i + \sum_{j \in S_i} t_j)/C]^+$
LB _i	(Lower Bound) the first station number that the task i can be assigned	$LB_i = [(t_i + \sum_{j \in P_i} t_j)/C]^+$

The mathematical model used for the SALBP-II solution can be seen in Figure 2.5 (Akpınar, 2009). In this model, it is assumed that there are no sequence dependent setup times.

Parameters	
$P_{ij} = \begin{cases} 1, & \text{if } i^{th} \text{ task is predecessor for } j^{th} \text{ task} \\ 0, & \text{if } i^{th} \text{ task is not predecessor for } j^{th} \text{ task} \end{cases}$	(2.1)
Decision variables	
$X_{ik} = \begin{cases} 1, & \text{if } i^{th} \text{ task assigned to } k^{th} \text{ station} \\ 0, & \text{if } i^{th} \text{ task not assigned to } k^{th} \text{ station} \end{cases}$	(2.2)
Objective function	
$Min Z = C$	(2.3)
S.t.	
$\sum_{k=1}^m X_{ik} = 1 \quad \forall i \in N$	(2.4)
$St_k \leq C \quad \forall k \in K$	(2.5)
$\sum_{i=1}^n X_{ik} * t_i = St_k \quad \forall k \in K$	(2.6)
$k * X_{jk} \geq X_{ik} * P_{ij} \quad \forall i, j \in N, \forall k \in m$	(2.7)
$X_{ik} \in \{0,1\}$	(2.8)

Figure 2.5 Binary integer programming model for SALBP-II

P_{ij} precedence relationships parameter shown at (2.1). This parameter is a matrix of 0 and 1. If task i is the predecessor of the task j , P_{ij} equals to 1 otherwise it will be 0. This parameter contains not only the immediate predecessors but also all predecessors. X_{ik} decision variable shown in (2.2) If the task i is assigned to the station k , X_{ik} will be equal to 1, otherwise it will be 0. The objective function is shown in (2.3) which is cycle time minimization. Constraint that each job should be assigned to only one station is shown in (2.4). Constraint shown in (2.5) is the constraint which ensures that the station time is smaller than the cycle time. In constraint shown in (2.6) station time

is calculated. According to this constraint, the sum of the processing times of the tasks assigned to a station gives station time of that station. Constraint shown (2.7) ensure that the assignment is appropriate for the priority relationships. According to this constraint all predecessors of a job j which assigned to the station k , must be assigned to station k or previous stations. Constraint shown in (2.8) indicates that X_{ik} decision variable is binary.

For the type-2 simple assembly line balancing problem with the sequence dependent setup times (SALBPs-II), in addition to the model for SALBP-II, it is necessary to take into account the positions of the jobs assigned to the station. The model for SALBPs-II is given in Figure 2.6 (Seyed-Alagheband, Fatemi Ghomi & Zandieh, 2011). In this model, the station time is calculated by taking into account the setup times in addition to task times of the tasks assigned to the station.

Parameters

$$P_{ij} = \begin{cases} 1, & \text{if } i^{th} \text{ task is predecessor for } j^{th} \text{ task} \\ 0, & \text{if } i^{th} \text{ task is not predecessor for } j^{th} \text{ task} \end{cases} \quad (2.9)$$

$$stp_{ij} = \text{Setup time from } i^{th} \text{ task to } j^{th} \text{ task} \quad (2.10)$$

Decision variables

$$X_{ikp} = \begin{cases} 1, & \text{if } i^{th} \text{ task assigned to } k^{th} \text{ station and } p^{th} \text{ position} \\ 0, & \text{if } i^{th} \text{ task not assigned to } k^{th} \text{ station and } p^{th} \text{ position} \end{cases} \quad (2.11)$$

$$Z_{ijk} = \begin{cases} 1, & \text{if } i^{th} \text{ task performed immediately before } j^{th} \text{ task} \\ & \text{in } k^{th} \text{ station in the same or next cycle } (i \neq j) \\ 0, & \text{else} \end{cases} \quad (2.12)$$

$$w_{ik} = \begin{cases} 1, & \text{if } i^{th} \text{ task is the last task on } k^{th} \text{ station} \\ 0, & \text{if } i^{th} \text{ task is not the last task on } k^{th} \text{ station} \end{cases} \quad (2.13)$$

Objective function

$$\text{Min } Z = C \quad (2.14)$$

S.t.

$$\sum_{k=1}^m \sum_{p=1}^n X_{ikp} = 1 \quad \forall i \in N \quad (2.15)$$

$$\sum_{i=1}^n X_{(i,k,p)} \leq 1 \quad \forall k \in K, p = 1, \dots, n \quad (2.16)$$

$$\sum_{k=1}^m X_{(i,k,p+1)} \leq \sum_{k=1}^m X_{(i,k,p)} = 1 \quad \forall i \in N, p = 1, \dots, n-1 \quad (2.17)$$

$$St_k \leq C \quad \forall k \in K \quad (2.18)$$

$$X_{jkp} * [(k-1) * n + p] \geq X_{ikp} * [(k-1) * n + p] * P_{ij} \\ \forall i, j \in N, \forall k \in K, p = 1, \dots, n \quad (2.19)$$

$$\sum_{i=1}^n \sum_{p=1}^n X_{ikp} * t_i + \sum_{i=1}^n \sum_{j=1}^n Z_{ijk} * stp_{ik} = St_k \quad \forall k \in m \quad (2.20)$$

$$X_{(i,k,p)} + X_{(j,k,p+1)} \leq 1 + Z_{(i,j,k)} \quad \forall k \in m \quad (2.21)$$

$$X_{(i,k,p)} - \sum_{j=1, j \neq i}^n X_{(j,k,p+1)} \leq w_{(i,k)} \quad \forall k \in m, \forall i \in n, \forall p = 1, \dots, n-1 \quad (2.22)$$

$$w_{(j,k)} + X_{(i,k,1)} \leq 1 + Z_{(i,j,k)} \quad \forall k \in m, \forall i, j \in m \quad (2.23)$$

$$X_{ikp} \in \{0,1\}, w_{ik} \in \{0,1\}, Z_{ijk} \in \{0,1\} \quad (2.24)$$

Figure 2.6 Binary integer programming model for SALBPs-II

The P_{ij} shown in (2.9) is the precedence relationships parameter alike the previous model. In (2.11), (2.12) and (2.13) decision variables are shown. The X_{ikp} decision variable shown in (2.11) is designed to be 1 if task i assigned to p^{th} position of the workstation k and 0 otherwise. In this model, the assumption is made that each station has virtual positions as much as the number of tasks. Hereby, when jobs are assigned to stations they are assigned to these positions respectively. The virtual positions in which the tasks will be assigned to the stations are required to calculate the setup times. Variables shown in (2.12) and (2.13) are decision variables added to the model in order to calculate setup times. Decision variable Z_{ijk} shown in (2.12) equals to 1 if the task i is performed just before the task j performed in station k (in the same or next cycle), 0 otherwise. If Z_{ijk} equals to 1, the setup time from task i to task j must be taken into account in calculating this station time. Since the after last assigned task is done in a station, the first assigned task will be done in the next cycle, the setup time from last task to first task must also be taken into account in calculating this station time. The w_{ik} decision variable shown in (2.13) equals to 1 if task i is the last task in station k and this variable (with the help of constraint shown in (2.23) ensures Z_{ijk} to be equal to 1, if task i assigned to station k as last task, and task j is the first task in station k . The objective function shown in (2.14) is cycle time minimization. Constraint shown in (2.15) ensures that each job is assigned to a position of a station. Constraint (2.16) forbids to assign multiple tasks to a position. Constraint (2.17) ensures jobs can be assigned to positions in order. This prevents the second or next positions from filling before the first position is filled in a station. Constraint (2.18) ensures that station times are smaller than the cycle time. Constraint (2.19) is the precedence relations constraint. According to this constraint, a successor task cannot be assigned to the previous station from its predecessors assigned, or to the previous position from its predecessors assigned on the same station. Constraint (2.20) calculates the station time, which is the sum of the total task time and setup times. Constraints shown in (2.21), (2.22) and (2.23), together ensures the Z_{ijk} equals to be 1 for the setup time calculation. Constraint (2.21) makes Z_{ijk} equal to 1, if the task j is assigned to the next position of task i , in station k . Constraint (2.22) ensures that the w_{ik} value equal to 1, if the task i is the last task in station k , and together with the constraint (2.23), they ensure Z_{ijk} to be equal to 1 if i is the last task and j is the first task in station k .

2.2.2 Solution approaches for SALB-II

The solution methods in the literature for SALBP-II can be examined under two headings as exact methods or heuristic approaches. Classification of solution methods can be seen in Figure 2.7. While some of the exact methods aim to solve the problem by using iterative solution strategy (Scholl, 1999; Hackman & Magazine, 1989), there is an exact method to reach a direct solution (Klein & Scholl 1996). Also, integer programming models have been developed to solve the problem (Pastor & Ferrer 2007; Pastor & Ferrer 2009). Beside the exact methods, heuristic methods developed to solve the problem also occupy a wide space in the literature. The heuristic methods developed to solve the problem include: the two-step heuristic based on linear programming (Ugurdag, Rachamadugu & Papachristou, 1997), a developed heuristic based on petri nets (Kilincci, 2010), a heuristic working by improving initial solution by trade and transfer procedures (Liu, Ong & Huang, 2003), genetic algorithm (GA) metaheuristics (Anderson & Ferris, 1994; Watanable, Hoshimoto & Nishikawa, 1995; Kim, Y. J., Kim, Y. K., & Cho, 1998; Kim, Y. K., Kim, Y. J., & Kim, Y., 1996; Goncalves & De Almeida, 2002), tabu search (TS) algorithms (Scholl, 1997; Chiang, 1998), simulated annealing (SA) metaheuristics (Henrici, 1994; Seyed-Alagheband et al., 2011), ant colony (AC) algorithms (Boysen & Fliedner, 2008; Zheng, Li, Li & Tang, 2013), particle swarm optimization (PSO) algorithm (Nearchou, 2011), differential evolution (DE) algorithm algorithms (Nearchou, 2011; Zhang, Yan, Liu & Jiang, 2016), beam search algorithm (Zhang, Yan, Liu & Jiang, 2016; Blum & Miralles, 2011; Blum, 2011) and variable neighbourhood search algorithms (Lei & Guo, 2016; Polat, Kalayci, Mutlu & Gupta, 2016).

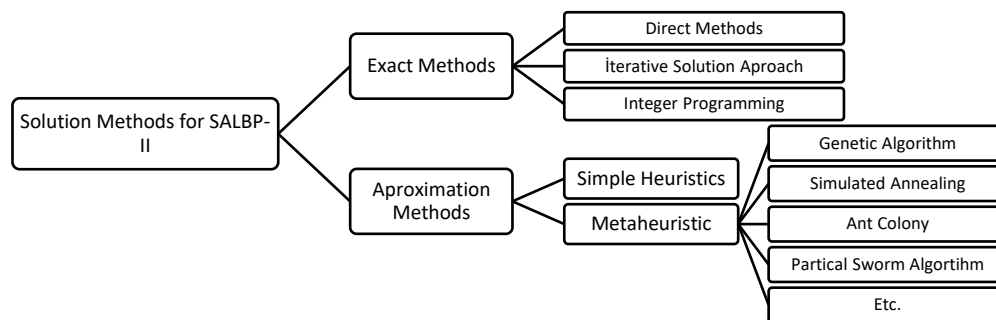


Figure 2.7 Solution approaches for SALBP-II

CHAPTER THREE

AN OVERVIEW ON THE LARGE NEIGHBOURHOOD SEARCH ALGORITHM

3.1 Features of Heuristic Algorithms

Heuristic algorithms are approximation methods that aim to approach the optimum solution instead of finding the exact optimum solution of the problem. Heuristic algorithms can be examined as two types; constructive algorithms and improvement algorithms. Constructive algorithms aim to achieve the solution directly by adding solution components one by one, while improvement algorithms start with a specified number of initial solutions and aim to approach the optimum solution by improving these solutions in an iterative manner. Initial solution is a feasible solution which generated randomly by considering some rules. In improvement algorithms certain number of new solutions are created in each iteration. The process of switching from one solution to another is about the neighborhood structure of the algorithm. In each iteration, the algorithm jumps from a solution to a neighbor solution in the solution zone. In this process, the solution which is better than the neighboring solutions is called the local optimum. Heuristic algorithms should have two important capabilities. First capability is looking for the best solution among the neighbors (intensification ability), the other one is to search for remote areas of the solution zone to avoid stuck to local optimum (diversification ability). The balance of these two abilities that can be adjusted by the neighborhood structure, determines the success of that algorithm.

3.2 Large Neighbourhood Search Algorithm

The Large Neighborhood Search (LNS) algorithm is a heuristic algorithm which search optimum solution in a large neighbourhood. LNS was introduced as a new local search algorithm for the solution of vehicle routing problems (VRP) (Shaw, 1997) and in the next, an improved version of it was presented (Shaw, 1998). The need to develop an effective and easy-to-implement method to solve all vehicle routing problems is the motivation behind the development of the LNS algorithm. Before LNS algorithm proposed in the literature, vehicle routing problems could be solved by both exact and

heuristic methods. As with many other problems, while small size problems can be solved by exact methods like mathematical model optimization, when the problem volume expands, the solution with the exact methods is facing big computational difficulties and it is impossible to solve the problem. Therefore, mostly heuristic methods are used for large volume problems. For the solution of large-volume vehicle routing problems, metaheuristics such as simulated annealing, tabu search, genetic algorithm and hybrid or extended versions of them have been successfully applied in the literature (Shaw, 1997). There are some issues encountered when adapting these metaheuristic algorithms to a problem. For example, adjusting the parameters and ensuring healthy operation is a highly demanding task. Small changes in the adaptation of the metaheuristics will seriously affect the performance of the solution method, in some cases solution obtained by an improperly adapted method can be even worse than the solution made with paper and pencil by a human (Shaw, 1997). Another issue is that metaheuristics are changing with a fixed neighborhood structure, but it is necessary to search for a variable and large neighborhood to avoid local optimizations. The large neighborhood search algorithm has been a solution to all of these problems and it has been an algorithm that has achieved successful results in vehicle routing problems.

This algorithm, which searches in a large neighborhood with removal and insertion operators, has been more successful than other metaheuristics on VRP. The neighbourhood, which is used in other metaheuristics, requires careful adjustment of the parameters in order to get rid of the local optimum and obtain good results. The LNS algorithm can be adapted more easily to the problem and can achieve more successful results thanks to the neighborhood structure where removal and insertion operators are used. An adaptive version of this algorithm was presented in 2006 (Ropke & Pisinger, 2006). In this adaptive version, multiple methods have been defined for use in removal and insertion operators and the algorithm chooses the method adaptively that will give successful results.

Large neighborhood search algorithm, like other improvement heuristics, improve the solution iteratively by reaching new solutions through a randomly generated solution. A solution is a sequence of customers and the customers are the components

of the solution. LNS uses removal and insertion operators to reach new solutions through an initial solution. In order to use these operators, firstly the number of components to be removed must be determined. This number is directly related to the diversification-intensification capability balance of the solution method. The bigger the number, the greater the capability to jump to remote solutions in solution area (diversification ability), the smaller the number, the better the ability to search the local (intensification ability). After the number of components to be removed is determined, this number of components is removed from the solution array with the removal operator and a relaxed solution is obtained. Then, components that have been removed can be added to another location in the relaxed array with the insertion operator so a new solution is reached. The LNS algorithm accepts the new solution if the new solution is better, otherwise it rejects it. This greedy procedure used to accept the new solution is due to the fact that greedy is a fast and successful procedure (Shaw, 1997).

LNS operates on a sequence which shows the assignment of customers to routes. Basically, the algorithm removes and inserts customers to the routes by removal and insertion operators. In the following illustrations, the operation of the LNS algorithm has demonstrated over a sample of 10 customers and 2 routes. In Figure 3.1, the customers to be removed from the existing solution are selected. In Figure 3.2, the selected customers are removed with the removal operator and a relaxed solution is obtained. In Figure 3.3, a new solution is obtained by adding the customers to other positions with the insertion operator. The removal and insertion operators are described in more detail below.

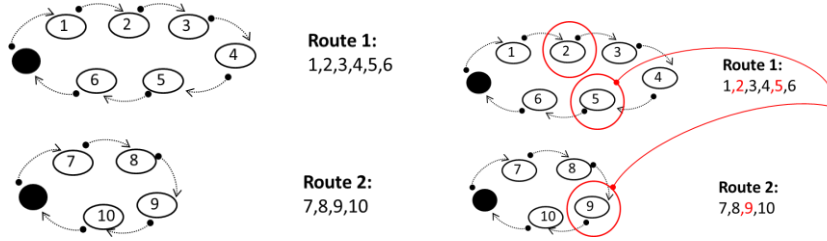


Figure 3.1 Choosing customers to be removed

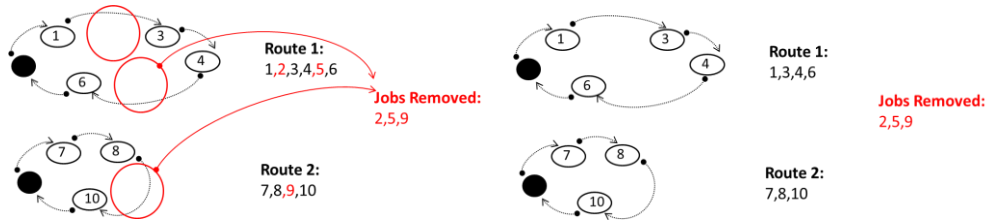


Figure 3.2 Remove customers from the routes

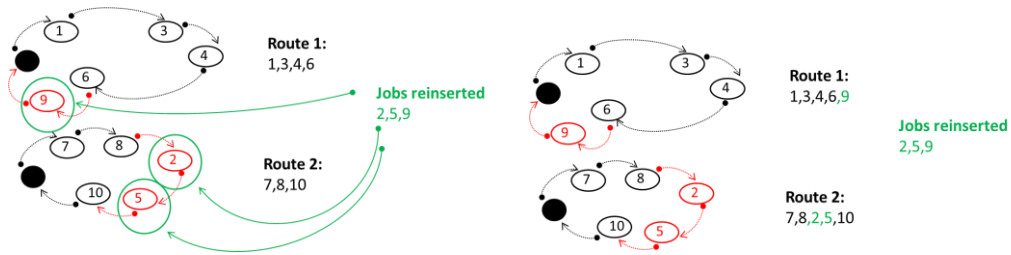


Figure 3.3 Reinsert customers to the routes

3.2.1 Removal Operator

Removal operator is the operator who determines which component to remove from the solution and obtains a relaxed solution by removing these components from the solution. A problem-specific removal operator should be used for efficient operation of the LNS algorithm. The removal operator should also have a randomness when applying a problem-specific removal rule. For vehicle routing problems, to determine components to be removed, approaches like “to remove related elements” or “to remove geographically close elements” are used. Removal of interrelated elements may result with re-insertion positions being the same (Shaw, 1997). To remove the element which most minimize the total idle time or the element has biggest position

weight etc. approaches used in removal operator while adapting LNS to ALBP. The operation of the developed algorithm will be explained in detail in chapter four.

3.2.2 Insertion Operator

The insertion is done by re-inserting the previously removed components to most appropriate insertion position. For this purpose, branch and bound method (B&B) is used and it is supported by the heuristics when B&B is insufficient (Shaw, 1997).

3.3 Adaptive Large Neighbourhood Search Algorithm

The study proposed in 2006 presents an Adaptive Large Neighbourhood Search (ALNS) Heuristic (Ropke & Pisinger, 2006). In this study, three different heuristics used in removal operator and two different insertion heuristics used in insertion heuristic. It is ensured that the algorithm to select which heuristic will be used in each iteration adaptively.

The removal operator uses three different heuristic methods: Shaw removal, worst removal and random removal. Shaw removal is the original method of LNS which developed by Shaw (1997) when the LNS algorithm first proposed (Ropke & Pisinger, 2006). This method proceeds in the logic of removing related components from the solution. Second method, the worst removal removes the components with the highest costs. While Shaw removal heuristic focused on removing the easily can be displaced components from the solution, the worst removal heuristic focused on removing components that has high capacity to improve the solution; in other words, the component that clearly placed in the wrong place in the solution. The third method is random removal. This method removes components completely random. The insertion operator uses the basic greedy heuristic and the regret heuristic. Basically, greedy heuristic is a heuristic which reinserts components back to the solution one by one from the most minimizes the cost to the less minimizes the cost. Regret heuristic reinserts components in a way that the component which most regretted if it is not reinserted now is reinserted first. Regret is measured by the increase in cost. Regret-k heuristic compares the cost of best location of a component with the cost of best k^{th}

location of this component. This cost difference means regret. For example, regret-2 compares a component's best and second-best location costs. If the component not reinserted now, this component may could not locate to its best location. Because of not inserting the component now, the component may be forced to locate its second-best position. In this case regret of a component is equal to cost differences of its best and second-best location. Regrets of all removed components are calculated and these components reinserted to the solution in descending order by their regrets.

While adapting this algorithm to solve ALBP these studies were inspired.

3.3.1 Choosing Methods as Adaptive

In the ALNS algorithm, removal and insertion heuristics that will be used in each iteration is selected by roulette wheel method (Ropke & Pisinger, 2006). In this structure each heuristic has a weight and the heuristic that will be used in each iteration is selected by roulette wheel method by using these weights. Initially, the weights of all heuristics are determined equally. While the weight of the successful heuristics increases, the weight of the unsuccessful heuristics decreases during the algorithm is performing. When the algorithm has a new solution in an iteration, if the fitness value of this solution is better than current fitness value, or even if it is not better, if this solution accepted by the algorithm this is considered a success; and the weight of the heuristics that used in removal and insertion operators is increased. The success of heuristics used in both removal and insertion operators is increased equally, because it is not possible to determine which operator is the source of success (Ropke & Pisinger, 2006). In order to update the weights in adaptive version of LNS (Ropke & Pisinger, 2006), segments include 100 iterations were determined and a weight update was made by checking the success at the end of each segment.

3.4 Literature Review

The basic logic of the large neighborhood search algorithm is to easily search the wide solution region. (Pisinger & Ropke, 2010). Destruction and repair mechanisms (removal and insertion operators) are the basis of this search capability (Pisinger &

Ropke, 2010). These mechanisms have been developed in studies on the LNS algorithm. The original algorithm presented by Shaw (1997) only accepts better solutions. However, later studies have used the solution acceptance mechanism as simulated annealing. (Ropke & Pisinger, 2006; Schrimpf, Schneider, Stamm-Wilbrandt & Dueck, 2000). Another development in the LNS algorithm is the changes in destruction and repair mechanisms. In the original algorithm has only one method for each destruction and repair mechanisms, while the later studies provided mechanisms in which multiple methods were used in an adaptive manner (Ropke & Pisinger, 2006). Various studies on the different adaptive structures of this algorithm are included in the literature (Perron, 2003; Laborie & Godard, 2007; Palpant, Artigues & Michelon, 2004; Nagata & Bräysy, 2009). The reader may refer to the comprehensive review study Ropke & Pisinger (2010) about large neighborhood algorithm. In addition to this review study, many studies have been presented about LNS and ALNS in the literature in the last 10 years. Laporte, Musmanno & Vocaturo (2010) presented an ALNS for the capacitated arc-routing problem with stochastic demands, Korsvik, Fagerholt & Laporte (2011) presented an ALNS for ship routing and scheduling with split loads, Ribeiro & Laporte (2012) presented an ALNS for the cumulative capacitated VRP, Muller, Spoorendonk & Pisinger (2012) presented a hybrid version of ALNS for lot-sizing with setup times, Kovacs, Parragh, Doerner & Hartl (2012) presented an ALNS for service technician routing and scheduling problems, Demir, Bektaş & Laporte (2012) presented an ALNS for the pollution-routing problem, Hemmelmayr, Cordeau & Crainic (2012) presented an ALNS for two-echelon vehicle routing problems arising in city logistics, Masson Lehuédé & Péton (2013) presented an ALNS search for the pickup and delivery problem with transfers, Azi, Gendreau & Potvin (2014) presented an ALNS for a VRP with multiple routes, Adulyasak, Cordeau & Jans (2012) presented an ALNS for the production routing problem, Aksen, Kaya, Salman & Tüncel (2014) presented an ALNS for a selective and periodic inventory routing problem, Belo-Filho, Amorim & Almada-Lobo (2015) presented an ALNS for the operational integrated production and distribution problem of perishable products, Luo, Qin, Zhang & Lim (2016) presented an ALNS for VRP with stochastic demands and weight-related cost, Li, Chen & Prins (2016) presented an ALNS for the pickup and delivery problem with time windows,

profits, and reserved requests, Ghilas, Demir & Van Woensel (2016) presented an ALNS for the pickup and delivery problem with time windows and scheduled lines, Dominguez, Guimarans, Juan & de la Nuez (2016) presented a LNS for the two-dimensional vehicle routing problem with backhauls, François, Arda, Crama & Laporte (2016) presented an ALNS for multi-trip VRP, Sze, Salhi & Wassan (2016) presented A hybridization of ALNS and LNS for VRP, Eskandarpour, Dejax & Péton (2017) presented a LNS for supply chain network design, Grimault, Bostel & Lehuédé (2017) presented an ALNS for the full truckload pickup and delivery problem with resource synchronization, Majidi, Hosseini-Motlagh & Ignatius (2018) presented an ALNS for pollution-routing problem with simultaneous pickup and delivery, Liu, Tao & Xie (2019) presented an ALNS for the VRP with time windows and synchronized visits, Lahyani, Gouguenheim & Coelho (2019) a hybrid version of ALNS for multi-depot open VRP, Ticha, Absi, Feillet & Quilliot (2019) presented an ALNS for the VRP with time windows, Santini (2019) presented an ALNS for the orienteering problem. Hammami, Rekik, & Coelho (2020) presented A Hybrid Adaptive Large Neighborhood Search (HALNS) for the Team Orienteering Problem. Hà et al. (2020) presented a linear programming-based Adaptive Large Neighborhood Search for the vehicle routing problem.

CHAPTER FOUR

AN ADAPTIVE LARGE NEIGHBOURHOOD SEARCH ALGORITHM FOR SALBP-II

ALNS is an algorithm that achieves very successful results for vehicle routing problems. The structural similarity between VRP and SALBP-II is the motivation of this study. In this section, first the operation of the developed algorithm is explained, second the computational study to evaluate the algorithm's performance is mentioned.

4.1 ALNS Algorithm for SALBP-II (and SALBPs-II)

In this study, the ALNS algorithm previously used for VRP was adapted for use in SALBP-II. The ALNS algorithm creates an initial solution and aims to approach the best solution by using the removal and insertion operators in each iteration. In order to use this algorithm which developed for VRP to solve the ALBP, operation of the algorithm must have been modified. For this purpose, the removal and insertion operators used in the original algorithm and the methods used in these operators are designed to be appropriate to the nature of ALBP. Under this heading, the characteristics of the developed algorithm, the meaning of the parameters used in the algorithm, the method of obtaining the initial solution, the operators used in the algorithm and the methods used in these operators and finally the solution acceptance strategy will be explained.

One of the most important factors necessary for the success of heuristic algorithms is the balance between intensification and diversification capability. ALNS algorithm provides this balance with its neighboring structure which used in removal and insertion operators. The number of components to be removed from the solution, the selection of the components to be removed and the method of reinserting these removed elements to the solution are the subjects determining the neighborhood structure. In the developed algorithm, the number of elements to be removed from the solution is determined by the parameter settings. The method of removing components and the method of reinsertion of these components to the solution are chosen between many methods as adaptive. Another factor related to the intensification and

diversification ability of the developed algorithm is the number of replicate and the minimum number of iterations (*MaxIter*) parameters. The algorithm starts to cycle through the number of replicate and the number of iterations (*MaxIter*) specified in the parameter setting. It starts with a new initial solution and iterates as much as the number of iterations in each replicate and improves the solution. Starting with different starting solutions and iterating provides to search solutions from different points of the solution space and this, develops the ability diversification ability of algorithm. As the number of replicate increases, the diversification capability of the algorithm will increase. The number of iterations is a parameter of how much we want to improve the initial solution. This parameter is related to intensification ability of the algorithm and this value directly related success of the improving to the initial solution.

Once an initial solution is created in each replicate, the objective function value is calculated and all data that the algorithm will use is updated according to this initial solution. The algorithm finds new solutions starting with the initial solution and changing the solution in each iteration. In each iteration, firstly it is determined how many tasks will be removed from the solution. This number is a random number, based on the specified parameter. When the number of tasks to be removed is determined, the method of removing tasks is determined in an adaptive manner. In the algorithm, 5 different remove methods have been defined and one of these methods is chosen as adaptive in each iteration. The definitions and selection of these methods are described in detail in 4.1.3 Removal Operator. The predetermined number of tasks removed from the solution by the chosen removal method and a relaxed solution is obtained. Temporary data that the algorithm will use on this relaxed solution is calculated. This temporary data is meaningful to find the most appropriate placement position for reinsertion of removed tasks. The operation of the removal operator is described in detail in 4.1.3 Removal Operator. The next step is to add these tasks back to the solution.

Insertion operator is part which removed tasks reinserted to the relaxed solution. For this purpose, firstly, the tasks to be reinserted to the solution must be queued for insertion. Because the insertion sequence of tasks will greatly affect the solution. The best insertion point for a task to a solution varies depending on whether another task

is inserted before it. Therefore, it is necessary to make a ranking that determines which task is to be inserted first. 16 different methods are used to make this sequence, and which method to be used is chosen as adaptive by the algorithm. The method definitions and choosing procedure are described in detail in 4.1.4 Insertion Operator. Tasks are ordered by the chosen method and they reinserted to the solution in this sequence. When a task is reinserted to the solution, it is inserted by finding the most appropriate position to be added with considering precedence relationships. The procedure for reinserting to the solution is detailed in 4.1.4 Insertion Operator. When a new solution is obtained by reinserting all tasks to the solution, the fitness value is calculated for the new solution and it is decided whether to accept or reject the solution. This procedure is described in detail in the section 4.1.5 Solution Acceptance. The flow of the algorithm shown in Figure 4.1 summarizes how the algorithm works:

```

set parameters
for i 1: replicate
    create initial solution
    calculate fitness
    for j 1: MaxIter
        determine how many jobs to remove
        choose removal method (adaptive)
        remove jobs (remove operator)
        update temporary parameters
        choose reorder method (adaptive)
        insertion (insertion position is the best position related to total idle time)
        calculate fitness and update parameters
        temperature control (is the new solution accepted)
    end iteration
end replicate
report results

```

Figure 4.1 The flow of the proposed algorithm

4.1.1 Parameter Definitions

The first step for running the algorithm is to define the parameters. The parameters to be defined are; parameters to determine how long it should work and adaptive structure parameters. According to the defined adaptive structure parameters, the data

structures required for the operation of the algorithm are generated by the algorithm. The parameters to be defined and the meanings of these parameters are explained in detail below.

First, it is determined how many iterations (*MaxIter*) and how many replicates (*replicate*) the algorithm will work. The number of iterations is the parameter that determines how many times each individual initial solution can be improved and reach a new solution. The number of replicates is the parameter of how many times the initial solution will be created. It improves the solution by making iterations as much as *MaxIter* value in each replicate. In total, we can say that solution improvement is done as many as the number of replicate is multiplied by the number of iterations (*MaxIter* * *replicate*).

Secondly, adaptive structure parameters are entered. It was mentioned that the algorithm chooses the removal method in removal operator and the sequencing method in insertion operator as adaptively. The algorithm takes into account the success of the methods in making this selection. In each iteration, the algorithm chooses the methods with the roulette wheel method according to the success of the methods. The selection probability of each method in this selection is the ratio of the success number of that method to the sum of success numbers of all methods. The probability of choosing a method is calculated according to the formula (4.1). Success number of methods are updated in each iteration. If a new solution is accepted in an iteration, the success number of the methods used in that iteration is increased by one. Thus, as the number of iterations increases, the likelihood of choosing methods that are successful for the problem solved is increased. When the new solution is accepted, it is not known whether this success is caused by the removal operator or by the insertion operator (Ropke & Pisinger, 2006). Therefore, the success number of methods used in both the removal operator and the insertion operator is increased by one. In Roulette Wheel selection, random selection is made by proportioning these success numbers in a roulette wheel. Thereby the method with high success is more likely to be selected. If the initial success number of methods are zero and if a method is selected in the first iterations of the first replicate, the success number of this method will be several times the number of other methods. In order to prevent this situation, a large number of

success numbers is initially assigned to each method so that it is prevented to be a large increase in the probability of selecting the same method in the early several iterations. This large number is a certain rate of the maximum number of iteration. This first parameter of adaptive structure is weight of adaptability (w_{adapt}) which between 0 and 1. The initial success numbers of the methods are calculated according to the formula shown in (4.2) using this parameter. For example, in a solution with 1000 iterations, if the adaptation weight is 0.5, the success of each method will be 500 at the beginning. Thus, the success of a method in the first iterations will not seriously affect the selection of that method in the selection of the wheel. As this parameter becomes smaller, selection of the method according to their success situation occurs when this parameter chosen bigger ignoring success of the methods situation occurs in the first replicate. Once the first replicate has been completed, the initial success numbers added in the first step are extracted and the actual success rates of the methods will be used in roulette wheel. From the second replicate, the algorithm makes selection according to the actual number of success of the methods and this parameter has no effect. Therefore, the first replicate acts like a warm-up period for the determination of suitable methods for the problem.

$$Possibility\ to\ choose\ method_r = \frac{Success\ Number\ of\ method_r}{\sum_{r=1}^{Num\ of\ methods} Success\ Number\ of\ method_r} \quad (4.1)$$

$$Initial\ Success\ Number = w_{adapt} * MaxIter \quad (4.2)$$

Another parameter for the adaptive structure is memory. While a method has been quite successful in some of the iterative progress, it may not be a success anymore. A memory function has been added in the algorithm to prevent the selection of this method again with its old successes. The determined i_memory and r_memory memory parameters which smaller than the number of iterations means; if a removal method has not been achieved a success in the last r_memory time it selected, the selection rate of this method will be reduced randomly. Similarly, if an insertion method is not successful in i_memory times it selected, the selection ratio of this method will be reduced randomly. Random reduce is performed by multiplying the selection rate of the method by a random number between 0 and 1. While this random number is

selected from a wide range and relatively closer to the 1 in first iterations, it is selected from a narrow range and very close to 0 and towards the latest iterations. This random number selection to perform the random reduce, utilizes from the specially developed Reduce Rate vector. Figure 4.2 shows an example of how this reduce rate vector is formed for 1000 iteration number. According to this figure, in the first iterations, 50%-100% of the success of the method is used in the roulette wheel, whereas in the last iterations almost 0% of its success is used in roulette wheel, if there is a reduction. The reason why this process is designed is to prevent the re-selection of methods that do not succeed towards the last iterations. As seen in the graph, the reduce rate has a randomness, but also has a decreasing randomness range and generally an exponentially decreasing trend.

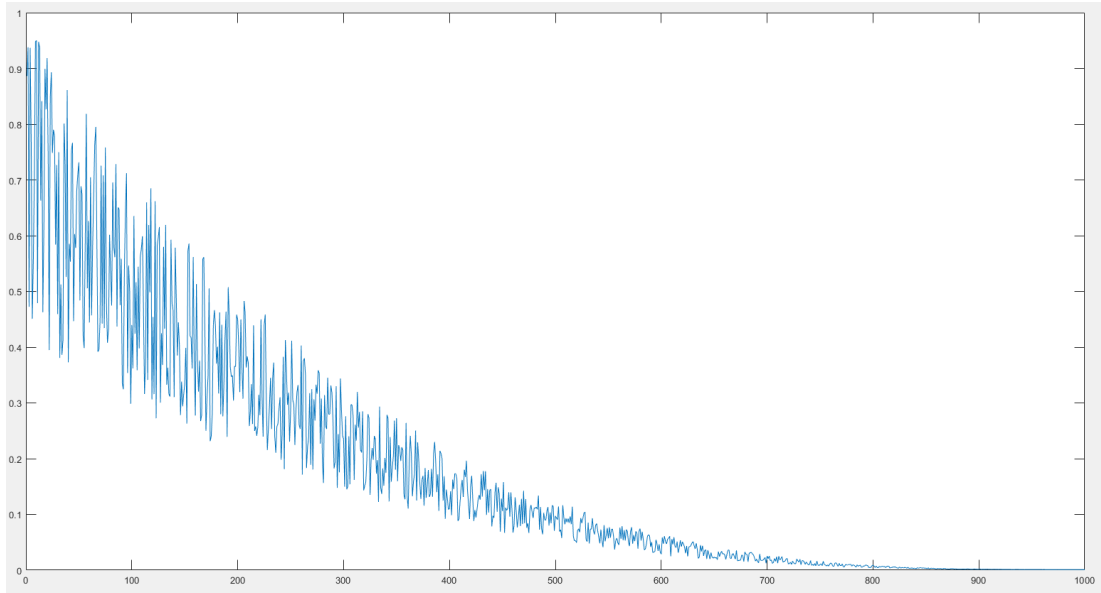


Figure 4.2 Reduce Rate

Finally, the parameters that determine how the algorithm works in general are determined. First, the number of methods to be used in the removal operator ($\#r_pol$) and the number of methods to be used in the insertion operator ($\#i_pol$) are determined. These methods must be defined. In the developed algorithm, 5 different removal methods are used, while 16 different before insertion sequencing methods are used. Determining number of methods is important because of creating and updating data structures that will be used in processes such as, process of keeping method successes,

process about memory, and method selection processes. The removal ratio parameter (*ratio_rmv*) determined which is used to determine the maximum number of tasks that can be removed at the removal operator. The number of tasks to be removed (q) is determined randomly in each iteration. However, the ratio of the maximum number of tasks that can be removed to the total number of tasks determined by this parameter. The p parameter used in the removal operator is determined. This parameter is used to add randomness to the process to determine which tasks will be removed by the selected remove method. This parameter is described in detail under 4.1.3 Removal Operator title below. For the new solution acceptance process, the temperature (T) and the cooling rate (cr) parameters are determined. The temperature and cooling rate will be described in 4.1.5 solution acceptance. And finally, whether is the problem contains setup times or not (*issetup*) is determined.

4.1.2 Initial Solution

This algorithm starts with an initial solution and improves this solution. A difference while adapting this algorithm to SALBP-II is that precedence relationships should be considered when creating the initial solution. For this reason, the first step in creating the initial solution is to create a random sequence between tasks that matches the precedence relationships. Once this sequence has been created, the tasks will be assigned to the stations respectively. This sequence has been created by adding tasks to the sequence one by one with a constructive method. This process continues until all tasks are added to the sequence. While adding each task to the sequence, the predecessors of that task are checked. If all of its predecessors are added to the sequence before it, the selected task is added to the sequence, otherwise by selecting another task this control is re-done. When this process is completed, a random sequence is obtained that matches the priority relationships between the tasks.

The final step in creating the initial solution is assigning tasks in the sequence to the stations respectively. For this purpose, an initial solution can be created by assigning tasks randomly to the stations, assigning an equal number of tasks to the stations or assigning tasks with an assignment method to provide close station times including randomness. It is important to obtain a relatively good initial solution to

achieve good results. Because of the precedence relationships is a fact of the problem, bad initial solutions are difficult to be improved. When the assignment is made by random assignment, it is possible to obtain an initial solution where some stations are empty. The improvement of such as initial solution is difficult because. In some cases, the station remains empty due to precedence relations no matter how many iterations done. In the equal number of assignments to the stations since the task times are not equal, serious differences may occur between the station times. The improvement of such as initial solution is also difficult because of the precedence relations. At the end of the experiments, it was determined that the most successful method was the method of assigning tasks to the stations with the close station times including randomness. In order to be able to assign with this method, firstly an average station time is calculated. The average station time is sum of the theoretical cycle time obtained by dividing the total workload to number of stations and the average setup time if the problem includes setup times as shown in (4.3). The average setup time is obtained by dividing the sum of smallest $n-m + 1$ setup time to m . Then the maximum deviation amount from the average time allowed in the stations is determined. This maximum deviation as can be seen (4.4) is the maximum difference between the task times or the minimum task time. Once a randomness parameter τ (*tau*) is also determined, the tasks are assigned to the stations in order. After a task assigned to current station if the station time this station less than sum of average station time and a randomly indicated amount of max deviation amount shown in (4.5), the task assigned to this station, otherwise the task assigned to the next station. Thus, both station times are made close and a random assignment is made.

$$C_{ort} = \frac{\sum_{i=1}^n t_i}{m} + \frac{sum(m - m + 1)_{lowest\ setup\ times}}{m} \quad (4.3)$$

$$dev_{max} = \max[\max(t_i) - \min(t_i), \min(t_i)] \quad (4.4)$$

$$\begin{cases} \text{Add to current station ,} & \text{if } S_i \leq C_{ort} + dev_{max} * rand^{\tau} \\ \text{Next Station,} & \text{else} \end{cases} \quad (4.5)$$

Solution Representation

The solution is displayed in a matrix with the stations in each row respectively, while a second matrix is kept indicating which task is in which station. The reason why this solution representation is preferred is to be able to easily check the priority relationships that should be done repeatedly in each iteration, and also to make the removal operations easily from the current solution. Figure 4.3 and figure 4.4 show a solution representation and a table shows index of the tasks in stations respectively which belongs to 10 tasks and a 4-station sample.

	Position 1	Position 2	Position 3	Position 4	Position 5	Position 6	Position 7	Position 8	Position 9	Position 10	The total number of tasks in the station
Station 1	1	2	0	0	0	0	0	0	0	0	2
Station 2	3	4	5	0	0	0	0	0	0	0	3
Station 3	6	7	8	0	0	0	0	0	0	0	3
Station 4	9	10	0	0	0	0	0	0	0	0	2

Figure 4.3 Solution representation

A 4 x 11 size solution representation matrix is created for 4 stations and 10 tasks. The last column is to keep the total number of tasks in each station.

	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8	Task 9	Task 10
Station 1	1	1	0	0	0	0	0	0	0	0
Station 2	0	0	1	1	1	0	0	0	0	0
Station 3	0	0	0	0	0	1	1	1	0	0
Station 4	0	0	0	0	0	0	0	0	1	1

Figure 4.4 Indexes of the tasks in station

As can be seen from the figure 4.4, it is easy to find which of the tasks are assigned to each station.

After obtaining the initial solution, the fitness value of the solution is calculated. The algorithm considers the total idle time as a fitness value and attempts to minimize

this value. The total idle time is the sum of the idle waiting times from station time to cycle time at each station. Station times are the sum of task times on each station. If there is a sequence dependent setup times, station times calculated by adding setup times between the tasks to the task times at each of the station. Since the first task assigned to that station will be performed after the last task assigned to a station, it should be noted that the setup time from the last task to the first task is also calculated when calculating the station time.

4.1.3 Removal Operator

Removal operator is the part of the algorithm which some components of the solution extracted from the solution. At the removal operator, first how many tasks will be removed from the solution is decided. This decision is related to the removal ratio parameter described in 4.1.1 parameter definitions. The number of tasks to be removed (q) determined randomly a range from 1 to multiplication of the total number of tasks (n) by the removal ratio ($ratio_rmvl$). After determining the number of tasks to be removed(q), a sequence created by removal method. Tasks will be removed are chosen from this sequence.

Once the number of tasks to be removed(q) is determined, it is decided by which method to remove. In order to make this decision, first of all the total success number of each method and memory parameters are used. After determining a temporary selection rate according to the number of successes of each method, the temporary selection rates of the methods, which need to be reduced, are reduced by checking the memory. To make a memory check, the success of each method in the last use is considered. For each method, the success of using this method in the last r_memory times is checked. If a method has not been successful in the last r_memory times used, the probability of selecting this method is reduced, and if the method has achieved even one success, the temporary selection rate used as it. Reducing the temporary selection rate will not affect future iterations, but will only affect the temporal selection rate data to be used for selection in the current iteration. It means if a method which has not been successful in the last r_memory times has been selected and succeeded, at the next iteration the temporary selection rate will be determined according to the

number of successes that not reduced. Similarly, if this method does not achieve success, at the next iteration, the temporary selection rate which recalculated according to the number of successes will be reduced, rather than the reduced temporary selection rate data in the previous iteration. The process of reducing this selection rate is described in heading 4.1.1 Parameter Definitions. After the memory is checked and temporary success rates is determined, the method will be used in the removal operator is chosen by the roulette wheel method according to these success rates. A sequence of tasks to be removed is obtained by sorting tasks according to this chosen method.

Once a sequence has been obtained with any of the removal methods, q tasks are selected to remove from this sequence. In this selection, the rank in the sequence for each removal is determined by the formula (4.6). This p parameter, which is assigned at the beginning of the algorithm, will increase the probability of selecting the first tasks if it equals to a big value, whereas if the p parameter is 1, then a random selection will be made. The reason for using the p parameter is to make a selection includes randomness rather than being restrict to the order of the methods.

$$rank = n * rand^p \quad (4.6)$$

In the removal process, after number of tasks to be removed and in the next the tasks to be removed determined, these tasks are removed from the solution and a relaxed solution is obtained. After this step, a new solution is obtained by reinserting this tasks to the solution with the reinserting operator.

The ALNS algorithm proposed for VRP used 3 different methods to remove related components (Shaw removal), remove the costliest components (Worst removal) and remove components randomly (Random removal) (Ropke & Pisinger, 2006). Five different removal methods used in this developed algorithm; positional weight methods (detailed in 4.1.3.2, 4.1.3.3 and 4.1.3.4) in response to the Shaw removal, effect on total idle time method (detailed in 4.1.3.1) in response to the worst removal and random removal method in response random removal. The methods used in the removal operator are described below.

4.1.3.1 Effect to Total Idle Time of Tasks to Be Removed (ETIT)

The method of removing the tasks by looking at the effect to total idle time of the task to be removed is a method that works as greedy. In this method, for each task in the current solution, it is calculated how much total idle time will be decrease if this task removed. For this calculation, after each task has been removed from the solution, new total idle time must have calculated after station times and cycle time are calculated. The decrease in total idle time is positive effect while increase of it is a negative effect. For this reason, all the tasks in the solution are sorted from the most minimizes that the total idle time to the less minimizes it (even in some cases from less maximizes to the most maximizes) and subtraction is performed according to this sequence. This method was inspired by the worst removal of the original algorithm.

4.1.3.2 Greatest Ranked Positional Weight Method (GRPW)

In this method, positional weight (PW) of each task is calculated with the formula shown in (4.7) and these tasks are sorted from large to small according to their positional weight. In this method the task with the greatest sum of the processing times of his and his successors is the first task in sequence. As a result, a sequence based on the priority relationship and the processing times is created.

This method and the two other methods to be explained thereafter obtain a sequence by considering only precedence relationships and processing times. These three methods are inspired from Shaw removal. If only the first described ETIT method was used, the algorithm is then stuck to the precedence relationships and could not improve the solution. In this case the station times of the first and last stations remain relatively small, while the station times of the other middle stations remaining large. Algorithm removes the tasks in middle stations but it cannot insert this tasks to the first or last stations due to precedence relationships. The usage of such a method was found to be appropriate, because if a predecessor of a removed task is on the same station with that task, the removed task could not be reinserted to the previous stations. The use of this method solves the problem of the fact that the first stations to be empty that can occur by using only the ETIT method.

$$PW = t_i + \sum_{j \in S_i}^i t_j \quad (4.7)$$

4.1.3.3 Greatest Ranked Sum of Previous Job Durations Method (GRPWP)

This method works similarly to the method described in the previous headings (GRPW). The different aspect is that this method considering predecessors of the task instead of successors shown in (4.8). If the successor of a task is on the same station, it is not possible to insert this removed task forward stations. This causes the last stations to remain empty. Similar to the case described in the previous headings, this method is a method used to prevent the last station to be empty.

$$PWP = t_i + \sum_{j \in P_i}^i t_j \quad (4.8)$$

4.1.3.4 Greatest Ranked Sum of Previous and Successors Job Durations Method (GRPWA)

This method is the combination of two methods described in the previous two headings. It is aimed to prevent both first and last stations to remain empty by using a single method. In this method, a sequence is created by sum of the values obtained in the previous two methods with formula shown in (4.9).

$$PWA = t_i + \sum_{j \in S_i}^i t_j + \sum_{j \in P_i}^i t_j \quad (4.9)$$

4.1.3.5 Random Ranking Method

In this method the tasks to be removed are selected completely randomly. It is used to add randomness to the removal operator and to prevent stuck to the local optimum.

4.1.4 Insertion Operator

Insertion operator is the part of the algorithm which extracted components reinserted to the relaxed solution so a new solution obtained. In the insertion operator, firstly the tasks are sorted and then they reinserted to the most appropriate positions in which they can be placed respectively. Since the most appropriate reinsertion position for a task will vary if another task is inserted to the solution before it, it is important to know in which sequence the tasks will be reinserted to the relaxed solution. 16 methods were used to sort these removed tasks among themselves. The first of these methods is the regret method used in the adaptive version of the LNS algorithm. (Ropke & Pisinger, 2006). The other 15 methods are commonly used methods in assembly line balancing problems. These methods are compiled by Baykasoglu (Baykasoglu, 2005). Choosing one of these methods is done in a similar way with the choosing removal methods described in 4.1.3 Removal Operator heading. At the same way, success and memory data are kept for each method. For each method, the temporary selection rate is defined by using the number of successes. In each iteration, the algorithm checks the memory and according to this, it reduces temporary selection rates of the methods which should be reduced. After that sorting method is selected by roulette wheel using temporary selection rates obtained. The methods used herein can be seen collectively in Table 4.1.

Table 4.1 Reorder methods used in insertion operator

No	Rule Name	Symbol	Definition
1	Regret-2	Regret2	$\Delta(TIT_{best}) - \Delta(TIT_{second})$
2	Longest Processing Time	LPT	t_i
3	Shortest Processing Time	SPT	t_i
4	Greatest Number of Immediate Successors	GNIS	$ IS_i $
5	Greatest Number of Successors	GNS	$ S_i $
6	Random Priority	RND	<i>random</i>
7	Smallest Task Number	STN	i
8	Greatest Ranked Positional Weight	GRPW	$t_i + \sum_{j \in S_i}^i t_j$
9	Greatest Average Ranked Positional Weight	GARPW	$\left(t_i + \sum_{j \in S_i} t_j \right) / (S_i + 1)$
10	Smallest Upper Bound	SUB	$N + 1 - \left[(t_i + \sum_{j \in S_i} t_j) / C \right]^+$
11	Smallest (Upper Bound Divided by the Number of Successors)	S_UB_NS	$UB_i / (S_i + 1)$
12	Greatest (Processing Time Divided by the Upper Bound)	G_PT_UB	t_i / UB_i
13	Smallest Lower Bound	SLB	$\left[(t_i + \sum_{j \in P_i} t_j) / C \right]^+$
14	Minimum Slack	MSLK	$UB_i - LB_i$
15	Minimum (Number of Successors Divided by Task Slack)	M_NS_SLK	$ S_i / (UB_i - LB_i)$
15	Greatest Number of Immediate Predecessors	GNIP	$ IP_i $

After the sorting method (which will be used in insertion operator) is determined, reinsertion is performed by finding the most appropriate position for the task to be inserted. When re-insertion is performed, the precedence relationship is checked to find the feasible insertion interval where is after all its predecessors and before all its successors. By controlling each position in this range, the most appropriate insertion position is determined. In order to find the most appropriate position, the test is performed for all positions in feasible insertion interval and the total idle time is calculated for each position. As a result, the insertion position which the most

minimizes the total idle time is selected. The methods used to sort the tasks (to be inserted) among themselves are described in detail below.

4.1.4.1 *Regret-2 (Regret2)*

The Regret method sorts tasks according to their regrets from biggest to the smallest. The regret value of a task is about how much we regret if we don't insert this job first. Regret value of the regret-k method is, the difference between the fitness value with a task's best insertion position and the fitness value with this task's best k^{th} insertion position. In this study regret-2 method is used. According to this method for each task it discovered the best insertion position and the second-best insertion position of this task. The total idle time of the solution in these two cases are calculated. The difference in total idle times between these two cases gives the regret value as shown (4.10). These regret values sorted descending and insertion sequence is obtained.

$$\text{Regret2} = \Delta(\text{TIT}_{\text{best}}) - \Delta(\text{TIT}_{\text{second}}) \quad (4.10)$$

4.1.4.2 *Longest Processing Time (LPT)*

This method sorts the tasks according to their processing times (PT) (4.11) descending order. the task with the longest processing time will be first task in the sequence obtained. The basic logic here is the expectation of the insertion the tasks with long processing times primarily will make the station times closer to each other. Although it is not given priority to insert short process-time tasks in the most appropriate positions, it is expected that because of shorter processing times they less likely to disturb line balance.

$$PT = t_i \quad (4.11)$$

4.1.4.3 *Shortest Processing Time (SPT)*

This method sorts the tasks according to their processing times (4.11) ascending order. This is the opposite of the LPT method described in 4.1.4.2. The logic here is to

insert as many tasks as possible in the most appropriate position because of their short processing times. Even if non-prioritized long process-time tasks are not inserted in the most appropriate positions, it is expected to create a solution that is easily improved for further iterations.

4.1.4.4 Greatest Number of Immediate Successors (GNIS)

This method, which sort tasks by descending order of their number of immediate successors (NIS), considers the precedence relationships. If a task with a great number of immediate successors (4.12) is located at the last stations, the successors of this task will always have to be inserted at the last stations due to precedence relations and therefore there will be great imbalance will be on the line. In order to prevent this situation, tasks with great number of immediate successors are given priority, so that these tasks are inserted in appropriate positions instead of being inserted in inappropriate positions at the last stations. The aim of this method is to prevent imbalance due to priority relations.

$$NIS = |IS_i| \quad (4.12)$$

4.1.4.5 Greatest Number of Successors (GNS)

This method, which is similar to the GNIS method described in 4.1.4.4, differs from GNIS method by considering all successors (4.13) instead of immediate successors. In this method, a descending sequence is made according to the number of all successors (NS). The general logic is the same as the GNIS method.

$$NS = |S_i| \quad (4.13)$$

4.1.4.6 Random Priority (RND)

Random priority method sorts the tasks (to be inserted) among themselves randomly. This method is used for adding to randomness and not stick to fixed rules.

4.1.4.7 Smallest Task Number (STN)

This method sorts tasks by the increasing task numbers (TN) (4.14).

$$TN = i \quad (4.14)$$

4.1.4.8 Greatest Ranked Positional Weight (GRPW)

This method sorts the tasks by considering the processing times with precedence relationships. For each task, a value (PW) is obtained with sum of the processing time of that task and the processing times all its successors (4.15). Tasks are sorted descending according to these values. This method is similar to the GNS method described in 4.1.4.5. differently, it is based on the logic that the processing times of the successors should be taken into account instead of the successor number.

$$PW = t_i + \sum_{j \in S_i}^i t_j \quad (4.15)$$

4.1.4.9 Greatest Average Ranked Positional Weight (GARPW)

This method is similar to the GRPW method described in 4.1.4.8. Differently, the total duration calculated, divided by one more than number of successors (4.16). Thus, the average processing time (ARPW) for that task and successors is calculated. The tasks are sorted descending according to this value obtained.

$$ARPW = \left(t_i + \sum_{j \in S_i} t_j \right) / (|S_i| + 1) \quad (4.16)$$

4.1.4.10 Smallest Upper Bound (SUB)

This method calculates the last station number (Upper bound) of the task can be assigned. This calculation is done for all task to be reinserted. In order to calculate this value, firstly lower bound is calculated by dividing the positional weight value by

theoretical cycle time (4.20). When calculating the lower bound value, the fractions are rounded to an upper integer. upper bound value is calculated by subtracting the lower bound value from the number of stations and adding 1 (4.17). The tasks are sorted by ascending order according to their upper bound values. In this method, it is intended to prevent imbalanced assignment due to precedence relationships by assigning tasks with the smallest upper bound values primarily. If the last station number of a task is smaller than other tasks, this task must assign before other tasks.

$$UB_i = N + 1 - \left\lceil (t_i + \sum_{j \in S_i} t_j) / C \right\rceil \quad (4.17)$$

4.1.4.11 Smallest (Upper Bound Divided by the # of Successors) (S_UB_NS)

This method is similar to the SUB method described in 4.4.10. Differently, calculated upper bound value divided by number of successors of the task (4.18). A sequence is obtained by sorting the tasks ascending according to these values (UB_NS).

$$UB_{NS} = UB_i / (|S_i| + 1) \quad (4.18)$$

4.1.4.12 Greatest (Processing Time Divided by the Upper Bound) (G_PT_UB)

In this method, a value (PT_UB) is calculated by dividing the process time by the upper bound value (4.19). A sequence is obtained by sorting the tasks descending order according to these values. This method is in a sense a combination of the LPT described in 4.1.4.2 and the SUB methods described in 4.1.4.10. On the one hand it considers the biggest processing time, on the other hand it considers the smallest upper bound while obtaining a sequence.

$$PT_{UB} = t_i / UB_i \quad (4.19)$$

4.1.4.13 Smallest Lower Bound (SLB)

This method calculates the smallest station number (Lower bound) of the task can be assigned. Lower bound is calculated by dividing the positional weight value by theoretical cycle time (4.20). A sequence is obtained by sorting the tasks ascending order according to these values. In this method, it is intended to prevent imbalanced assignment due to precedence relationships by assigning tasks with the smallest lower bound values primarily.

$$LB = \left[(t_i + \sum_{j \in P_i} t_j) / C \right]^+ \quad (4.20)$$

4.1.4.14 Minimum Slack (MSLK)

In this method, the interval (SLK) between the last station which the task can be assigned and the first station to which it can be assigned is calculated. This calculation is done by subtraction lower bound from upper bound (4.21). A sequence is obtained by sorting the tasks ascending order according to these values. In this method, it is intended to prevent imbalanced assignment due to precedence relationships by assigning tasks with large interval of assignment.

$$SLK = UB_i - LB_i \quad (4.21)$$

4.1.4.15 Minimum (# of Successors Divided by Task Slack) (M_NS_SLK)

In this method, a value (NS_SLK) obtained by dividing the total number of successors by the minimum slack value (4.22). A sequence is obtained by sorting the tasks ascending order according to these values. In this method, it is taken into consideration that the task with a maximum number of successes should be assigned first, while the task with the minimum number of slack must be assigned first. This method is a combination of the GNS method described in 4.1.4.5 and the MSLK method described in 4.1.4.14.

$$NS_SLK = |S_i| / (UB_i - LB_i) \quad (4.22)$$

4.1.4.16 Greatest Number of Immediate Predecessors (GNIP)

Although this method is similar to the GNIS method described in 4.1.4.4, it has the opposite logic. The GNIP method sorts the tasks in a descending order according to their number of immediate predecessors (NIP) (4.23). If a task with a great number of immediate predecessors is located at the first stations, the predecessors of this task will always have to be inserted at the first stations due to precedence relations and therefore there will be great imbalance will be on the line. In order to prevent this situation, tasks with great number of immediate predecessors are given priority, so that these tasks are inserted in appropriate positions instead of being inserted in inappropriate positions at the first stations. The aim of this method is to prevent imbalance due to priority relations.

$$NIP = |IP_i| \quad (4.23)$$

4.1.5 Solution Acceptance

In each iteration, the removal and insertion operators work once to reach a new solution. The fitness value is calculated for this new solution reached. Fitness value of the solution is the total idle time in that solution. Fitness value of the new solution is compared to the fitness value of the current solution. If the new solution is better than the current solution, the new solution is accepted. In this case, the new solution is compared with the best solution and if new solution better than best solution, it is kept as best solution.

If the new solution is not better than the current solution a control is done as in the simulated annealing algorithm. In order to perform this control, firstly the temperature (T) is decreased as far as the cooling ratio (cr) in each iteration (4.24) and after that a random number generated. If the condition given (4.25) provided the solution accepted, otherwise rejected.

$$T_{new} = T_{current} * cr \quad (4.24)$$

$$P_{ij} = \begin{cases} \text{Accept,} & \text{if } rand < e^{-\left(\frac{-(new\ cycle\ time - current\ cycle\ time)}{Temperature}\right)} \\ \text{Reject,} & \text{else} \end{cases} \quad (4.25)$$

If the new solution is accepted, necessary data updates are made for the next iteration. These updates include accepting the new solution as the current solution, accepting the new solution fitness value as the current solution fitness value, updating the data such as station time and total idle time etc. When the new solution is accepted, the methods used in removal and insertion operators are considered successful and the success numbers of them are increased one. The reason increasing the success numbers of the methods used in both removal and insertion operators is that the success does not belong only to the removal operator or to the insertion operator, it must belong to the method pair used in both operators. Also, the memory parameters of the methods are updated before the new solution. While the data that indicates they have succeeded in the last time they used for the methods kept in the memory, the oldest data is removed from the memory.

If the new solution is not accepted, the next iteration is continued with the current solution. In this case, the memory parameter of the methods used is updated. The oldest memory data is also forgotten while the data that indicates the methods failed in the last use are kept in the memory.

When all iterations and replicates are completed, the results are written to the screen and an excel file.

4.2 Computational Study

The presented ALNS algorithm was run for SALBP-II without setup times and for SALBPs-II problems with setup times and its performance was evaluated. The results obtained for SALBP-II were compared with the other algorithms that solve this problem and the performance of the developed algorithm was evaluated. In order to do this, the algorithm was run by using Benchmark Data Sets by Scholl (1993)

Retrieved June 14, 2017 from <https://assembly-line-balancing.de/>. The current benchmark data set for SALBP-II includes a total of 302 instances and two groups as data set 1 and data set 2. The first data set has 128 instances; 9 precedence relationship diagrams with the number of tasks vary from 29 to 111 and the problem of assigning this precedence relationship diagrams to the different number of stations. The second data set has 174 instances; 8 precedence relationship diagrams with the number of tasks vary from 53 to 297 and the problem of assigning this precedence relationship diagrams to the different number of stations. In these data sets, optimum cycle time c^* is given for each case obtained by assignment to the different number of stations of this total 17 precedence relationship diagrams. To solve these benchmark problems the developed ALNS algorithm was coded on matlab2018b and run using an Intel Core (TM) i7-4710MQ CPU 2.50 GHz computer. The results obtained and the success of algorithm comparison with previous studies are presented below.

4.2.1 Results of the SALBP-II

The developed ALNS algorithm was run 10 times for each problem because the majority of the algorithms to be compared presented the results of 10 runs (Akpınar, 2017). The parameters entered for the developed ALNS algorithm are summarized in Table 4.2. These parameters were decided by running the algorithm on the sample problems. As a performance measure $c\%dev$ which described below is used.

- **$c\%dev$:** The average relative deviation from the known optimum solution (previously known minimal cycle time c^*) in percentage; calculated as $((c - c^*)/c^*)100$; where c is the best cycle time achieved by the solution algorithm.

Table 4.2 Parameters of the proposed ALNS Algorithm for SALBP-II

Parameter	Definition	Value
replicate	Number of repeat runs with a new initial solution	10
MaxIter	Maximum number of iterations	100*n
p	Parameter that provides randomness in the removal operator	3
T	Initial temperature	100
cr	Cooling ratio	0.99975
ratio_rem	Ratio of removal tasks	0.6
q	Number of tasks to be removed	$1 \leq randi() \leq ratio_rem * n$
i_memory	Memory for insertion policies	100
r_memory	Memory for removal policies	100
#r_pol	Number of removal policy used in removal operator	5
#i_pol	Number of reorder policy used in insertion operator	16
wadapt	Weight of adaptive	0.5
issetup	Is the setup time be taken into account in the calculation	0
τ	Tao	3

Buxey precedence relationship diagram contains 29 tasks and task times vary from 1 to 25. 8 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 7 to 14. The results obtained is shown in table 4.3. The algorithm finds 7 optimum results of 8 instances at Buxey precedence diagram. The average of c%dev is 0.27% in this precedence diagram.

Table 4.3 Results for the Buxey problem set

#Station	Optimum Value	Result	c%dev
7	47	48	2.13
8	41	41	0.00
9	37	37	0.00
10	34	34	0.00
11	32	32	0.00
12	28	28	0.00
13	27	27	0.00
14	25	25	0.00
Average			0.27

Gunther precedence relationship diagram contains 35 tasks and task times vary from 1 to 40. 10 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 6 to 15. The results obtained is shown in table 4.4.

Table 4.4 Results for the Gunther problem set

#Station	Optimum Value	Result	c%dev
6	84	84	0.00
7	72	72	0.00
8	63	63	0.00
9	54	54	0.00
10	50	50	0.00
11	48	48	0.00
12	44	44	0.00
13	42	42	0.00
14	40	40	0.00
15	40	40	0.00
Average			0.00

Kilbridge precedence relationship diagram contains 45 tasks and task times vary from 3 to 55. 9 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 5 to 11. The results obtained is shown in table 4.5.

Table 4.5 Results for the Kilbridge problem set

#Station	Optimum Value	Result	c%dev
3	184	184	0.00
4	138	138	0.00
5	111	111	0.00
6	92	92	0.00
7	79	79	0.00
8	69	69	0.00
9	62	62	0.00
10	56	56	0.00
11	55	55	0.00
Average			0.00

Lutz1 precedence relationship diagram contains 32 tasks and task times vary from 100 to 1400. 5 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 8 to 12. The results obtained is shown in table 4.6.

Table 4.6 Results for the Lutz1 problem set

#Station	Optimum Value	Result	c%dev
8	1860	1860	0.00
9	1638	1638	0.00
10	1526	1526	0.00
11	1400	1400	0.00
12	1400	1400	0.00
Average			0.00

The algorithm solves all of instances optimally at Gunther, Kilbridge and Lutz1 precedence diagrams.

Sawyer precedence relationship diagram contains 30 tasks and task times vary from 1 to 25. 8 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 7 to 14. The results obtained is shown in table 4.7.

Table 4.7 Results for the Sawyer problem set

#Station	Optimum Value	Result	c%dev
7	47	48	2.13
8	41	41	0.00
9	37	37	0.00
10	34	34	0.00
11	31	31	0.00
12	28	28	0.00
13	26	26	0.00
14	25	25	0.00
Average			0.27

The algorithm finds 7 optimum results of 8 instances at Sawyer precedence diagram. The average of c%dev is 0.27% in this precedence diagram.

Lutz2 precedence relationship diagram contains 89 tasks and task times vary from 1 to 10. 20 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 9 to 28. The results obtained is shown in table 4.8.

Table 4.8 Results for the Lutz2 problem set

#Station	Optimum Value	Result	c%dev
9	54	54	0.00
10	49	49	0.00
11	45	45	0.00
12	41	41	0.00
13	38	38	0.00
14	35	35	0.00
15	33	33	0.00
16	31	31	0.00
17	29	30	3.45
18	28	28	0.00
19	26	26	0.00
20	25	25	0.00
21	24	24	0.00
22	23	23	0.00
23	22	22	0.00
24	21	21	0.00
25	20	20	0.00
26	19	20	5.26
27	19	19	0.00
28	18	18	0.00
Average			0.44

The algorithm finds 18 optimum results of 20 instances at Lutz2 precedence diagram. The average of c%dev is 0.44% in this precedence diagram.

Tonge precedence relationship diagram contains 70 tasks and task times vary from 1 to 156. 23 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 3 to 25. The results obtained is shown in table 4.9.

Table 4.9 Results for the Tonge problem set

#Station	Optimum Value	Result	c%dev
3	1170	1170	0.00
4	878	878	0.00
5	702	702	0.00
6	585	585	0.00
7	502	502	0.00
8	439	440	0.23
9	391	391	0.00
10	352	352	0.00
11	320	320	0.00
12	294	295	0.34
13	271	272	0.37
14	251	253	0.80
15	235	237	0.85
16	221	222	0.45
17	208	209	0.48
18	196	198	1.02
19	186	188	1.08
20	177	178	0.56
21	170	172	1.18
22	162	162	0.00
23	156	156	0.00
24	156	156	0.00
25	156	156	0.00
Average			0.32

The algorithm finds 11 optimum results of 23 instances at Tonge precedence diagram. The average of c%dev is 0.32% in this precedence diagram.

Arcus1 precedence relationship diagram contains 83 tasks and task times vary from 233 to 3691. 20 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 3 to 22. The results obtained is shown in table 4.10.

Table 4.10 Results for the Arcus1 problem set

#Station	Optimum Value	Result	c%dev
3	25236	25238	0.01
4	18927	18940	0.07
5	15142	15161	0.13
6	12620	12640	0.16
7	10826	10846	0.18
8	9554	9591	0.39
9	8499	8528	0.34
10	7580	7654	0.98
11	7084	7091	0.10
12	6412	6448	0.56
13	5864	5935	1.21
14	5441	5472	0.57
15	5104	5142	0.74
16	4850	4904	1.11
17	4516	4581	1.44
18	4317	4387	1.62
19	4068	4131	1.55
20	3882	3930	1.24
21	3691	3730	1.06
22	3691	3691	0.00
Average			0.67

The algorithm finds 1 optimum results of 20 instances at Arcus1 precedence diagram. The average of c%dev is 0.67% in this precedence diagram.

Arcus2 precedence relationship diagram contains 111 tasks and task times vary from 10 to 5689. 25 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 3 to 27. The results obtained is shown in table 4.11.

Table 4.11 Results for the Arcus2 problem set

#Station	Optimum Value	Result	c%dev
3	50133	50133	0.00
4	37600	37600	0.00
5	30080	30080	0.00
6	25067	25068	0.00
7	21486	21489	0.01
8	18800	18802	0.01
9	16711	16716	0.03
10	15040	15045	0.03
11	13673	13679	0.04
12	12534	12540	0.05
13	11570	11579	0.08
14	10747	10763	0.15
15	10035	10047	0.12
16	9412	9424	0.13
17	8855	8884	0.33
18	8377	8405	0.33
19	7922	7952	0.38
20	7524	7545	0.28
21	7187	7289	1.42
22	6856	6914	0.85
23	6560	6614	0.82
24	6282	6327	0.72
25	6101	6125	0.39
26	5855	5939	1.43
27	5689	5690	0.02
Average			0.31

The algorithm finds 4 optimum results of 25 instances at Arcus2 precedence diagram. The average of c%dev is 0.31% in this precedence diagram.

Warnecke precedence relationship diagram contains 58 tasks and task times vary from 7 to 53. 27 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 3 to 29. The results obtained is shown in table 4.12.

Table 4.12 Results for the Warnecke problem set

#Station	Optimum Value	Result	c%dev
3	516	516	0.00
4	387	387	0.00
5	310	310	0.00
6	258	258	0.00
7	222	222	0.00
8	194	195	0.52
9	172	173	0.58
10	155	157	1.29
11	142	143	0.70
12	130	131	0.77
13	120	122	1.67
14	111	112	0.90
15	104	106	1.92
16	98	99	1.02
17	92	95	3.26
18	87	89	2.30
19	84	85	1.19
20	79	81	2.53
21	76	77	1.32
22	73	74	1.37
23	69	72	4.35
24	66	68	3.03
25	64	66	3.13
26	64	64	0.00
27	60	63	5.00
28	59	60	1.69
29	56	57	1.79
Average			1.49

The algorithm finds 6 optimum results of 27 instances at Warnecke precedence diagram. The average of c%dev is 1.49% in this precedence diagram.

Wee-Mag precedence relationship diagram contains 75 tasks and task times vary from 2 to 27. 28 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 3 to 30. The results obtained is shown in table 4.13.

Table 4.13 Results for the Wee-Mag problem set

#Station	Optimum Value	Result	c%dev
3	500	500	0.00
4	375	375	0.00
5	300	300	0.00
6	250	250	0.00
7	215	215	0.00
8	188	188	0.00
9	167	167	0.00
10	150	150	0.00
11	137	137	0.00
12	125	126	0.80
13	116	116	0.00
14	108	109	0.93
15	100	101	1.00
16	94	95	1.06
17	89	90	1.12
18	87	88	1.15
19	85	85	0.00
20	77	77	0.00
21	72	73	1.39
22	69	69	0.00
23	67	68	1.49
24	66	67	1.52
25	65	66	1.54
26	65	65	0.00
27	65	65	0.00
28	64	64	0.00
29	63	63	0.00
30	56	56	0.00
Average			0.43

The algorithm finds 18 optimum results of 28 instances at Wee-Mag precedence diagram. The average of c%dev is 0.43% in this precedence diagram.

Lutz3 precedence relationship diagram contains 89 tasks and task times vary from 1 to 74. 21 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 3 to 23. The results obtained is shown in table 4.14.

Table 4.14 Results for the Lutz3 problem set

#Station	Optimum Value	Result	c%dev
3	548	548	0.00
4	411	411	0.00
5	329	329	0.00
6	275	275	0.00
7	236	236	0.00
8	207	207	0.00
9	184	185	0.54
10	165	166	0.61
11	151	151	0.00
12	138	138	0.00
13	128	128	0.00
14	118	119	0.85
15	110	112	1.82
16	105	106	0.95
17	98	99	1.02
18	93	94	1.08
19	89	90	1.12
20	85	86	1.18
21	80	82	2.50
22	76	78	2.63
23	74	75	1.35
Average			0.75

The algorithm finds 9 optimum results of 21 instances at Lutz3 precedence diagram. The average of c%dev is 0.75% in this precedence diagram.

Mukherje precedence relationship diagram contains 94 tasks and task times vary from 8 to 171. 24 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 3 to 26. The results obtained is shown in table 4.15.

Table 4.15 Results for the Mukherje problem set

#Station	Optimum Value	Result	c%dev
3	1403	1403	0.00
4	1052	1052	0.00
5	844	844	0.00
6	704	704	0.00
7	621	621	0.00
8	532	533	0.19
9	471	471	0.00
10	424	424	0.00
11	391	391	0.00
12	358	358	0.00
13	325	326	0.31
14	311	312	0.32
15	288	289	0.35
16	268	270	0.75
17	251	251	0.00
18	239	240	0.42
19	226	226	0.00
20	220	222	0.91
21	208	209	0.48
22	200	201	0.50
23	189	190	0.53
24	179	180	0.56
25	172	174	1.16
26	171	171	0.00
Average			0.27

The algorithm finds 12 optimum results of 24 instances at Mukherje precedence diagram. The average of c%dev is 0.27% in this precedence diagram.

Barthold precedence relationship diagram contains 148 tasks and task times vary from 3 to 383. 13 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (#Station) varies from 3 to 15. The results obtained is shown in table 4.16.

Table 4.16 Results for the Barthold problem set

#Station	Optimum Value	Result	c%dev
3	1878	1878	0.00
4	1409	1409	0.00
5	1127	1127	0.00
6	939	939	0.00
7	805	805	0.00
8	705	705	0.00
9	626	627	0.16
10	564	564	0.00
11	513	513	0.00
12	470	470	0.00
13	434	435	0.23
14	403	405	0.50
15	383	383	0.00
Average			0.07

The algorithm finds 10 optimum results of 13 instances at Barthold precedence diagram. The average of c%dev is 0.07% in this precedence diagram.

Hahn precedence relationship diagram contains 53 tasks and task times vary from 40 to 1775. 8 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (#Station) varies from 3 to 10. The results obtained is shown in table 4.17.

Table 4.17 Results for the Hahn problem set

#Station	Optimum Value	Result	c%dev
3	4787	4787	0.00
4	3677	3677	0.00
5	2823	2823	0.00
6	2400	2400	0.00
7	2336	2336	0.00
8	1907	1907	0.00
9	1827	1827	0.00
10	1775	1775	0.00
Average			0.00

The algorithm solves optimally all of 8 instances at Hahn precedence diagram.

Barthol2 precedence relationship diagram contains 148 tasks and task times vary from 1 to 83.15 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 27 to 41. The results obtained is shown in table 4.18.

Table 4.18 Results for the Barthold2 problem set

#Station	Optimum Value	Result	c%dev
27	157	160	1.91
28	152	155	1.97
29	146	149	2.05
30	142	145	2.11
31	137	140	2.19
32	133	134	0.75
33	129	131	1.55
34	125	128	2.40
35	121	124	2.48
36	118	121	2.54
37	115	117	1.74
38	112	115	2.68
39	109	112	2.75
40	106	112	0.00
41	104	111	0.00
Average			1.81

The algorithm finds 2 optimum results of 15 instances at Barthold2 precedence diagram. The average of c%dev is 1.81% in this precedence diagram.

The developed SALBP-II algorithm was compared with the previously developed 15 different algorithms to solve the SALBP-II problem, as was done by Akpınar (2017) and the non-adaptive version of this algorithm (Akpınar, 2017). In these 16 algorithms there are three versions of the genetic algorithm [pGA (Kim et al., 1996), rGA_prb and rGA_rks (Goncalves & De Almeida, 2002)], four different versions of the differential evolution algorithm [DE_prb and DE_rks (Nearchou, 2007), IDEA_rd2 and IDEA_opt (Zhang, Yan, Liu & Jiang, 2016)], petri net based three heuristic

algorithm [PNA_for, PNA_back and PNA_bid (Kilincici, 2010)], four versions of PSO [PSO1, PSO2, PSO3 and PSO4 (Nearchou, 2011)], the iterative beam search algorithm [IBS (Blum & Miralles, 2011)] and the previously developed non-adaptive version of this algorithm [LNS (Akpınar, 2017)]. The results obtained by running the ALNS algorithm and the results of 16 different algorithms listed by Akpınar (2017) are presented in the table 4.19.

Table 4.19 Summary of the computational results

Problem Name	n	pGA	rGA_rks	rGA_prb	DE_prb	DE_rks	IDEA_rd2	IDEA_apt	PNA_for	PNA_back	PNA_bid	PSO1	PSO2	PSO3	PSO4	IBS	LNS	ALNS
Data Set 1																		
Buxey	29	3.26	2.15	2.46	2.69	1.16	2.21	3.07	3.07	2.92	3.76	1.96	5.88	1.55	5.01	0.0000	0.55	0.27
Sawyer	30	2.61	4.51	3.67	3.52	2.27	3.34	3.52	4.00	4.39	5.24	2.24	6.62	1.60	5.61	0.0000	1.22	0.27
Lutz1	32	0.32	0.71	0.00	0.35	0.32	0.35	0.71	4.09	2.93	4.09	-	-	-	-	0.0000	0.32	0.00
Gunther	35	0.64	1.71	1.21	1.02	0.14	1.02	1.27	1.27	2.99	1.27	0.87	3.88	0.64	3.92	0.0000	0.42	0.00
Kilbridge	45	1.38	0.96	0.80	0.60	0.66	0.60	0.60	2.19	1.46	1.91	0.80	1.64	0.68	1.78	0.0000	0.07	0.00
Tonge	70	2.75	3.63	3.63	2.07	1.88	2.07	1.78	2.53	3.41	2.92	2.91	5.99	2.03	5.82	0.0000	1.03	0.32
Arcus1	83	1.65	1.96	1.48	0.83	0.99	0.98	0.78	2.47	2.57	2.25	1.42	3.20	0.70	3.04	0.0287	0.70	0.67
Lutz2	89	3.13	3.76	2.84	2.54	3.08	2.64	1.99	2.99	4.44	3.92	-	-	-	-	0.0000	0.85	0.44
Arcus2	111	5.02	5.27	5.11	4.98	4.96	4.98	4.64	2.06	4.61	2.25	3.79	6.67	2.02	5.77	0.0066	1.82	0.31
Average		2.31	2.74	2.36	2.07	1.72	2.02	2.04	2.74	3.30	3.07	2.00	4.84	1.32	4.42	0.0039	0.78	0.25
Data Set 2																		
Hahn	53	0.00	1.27	0.00	0.00	0.00	0.00	0.00	2.52	2.08	2.61	-	-	-	-	0.0000	0.50	0.00
Warnecke	58	4.29	6.40	5.25	3.31	3.74	3.98	3.51	5.57	5.90	6.01	-	-	-	-	0.0579	0.35	1.49
Wee-Mag	75	2.61	2.37	2.44	1.39	1.23	1.63	1.39	1.56	1.63	1.57	-	-	-	-	0.0000	0.00	0.43
Lutz3	89	2.54	3.20	2.19	1.58	1.68	2.88	1.57	2.59	2.88	2.94	-	-	-	-	0.0000	0.78	0.75
Mukherje	94	-	-	-	-	-	-	-	1.04	1.41	1.21	-	-	-	-	0.0000	0.64	0.27
Barthold1	148	0.85	0.76	0.58	0.24	0.26	0.46	0.26	1.02	0.46	0.48	-	-	-	-	0.0000	0.64	0.07
Barthold2	148	9.64	9.77	9.88	7.37	6.85	4.79	4.79	3.97	4.79	4.32					0.0000	2.14	1.81
Average		4.30	5.08	4.38	3.39	3.32	3.41	2.97	2.69	2.76	2.74					0.0076	1.02	0.69

As seen in Table 4.19, the developed algorithm has achieved very successful results. In data set 1 ALNS showed more successful results than other algorithms except IBS algorithm. In Date set 2 LNS algorithm is seems like more successful than ALNS in Warnecke and Wee-Mag precedence relationship diagrams in addition to the IBS algorithm but in the rest ALNS algorithm is more successful. These results showed that the developed algorithm was successful for the SALBP-II problem.

4.2.2 Results of the SALBPs-II

SALBPs-II has two different setup time data: low variability (LV) and high variability (HV). The SALBPs-II problem is solved by using these data sets. Since

there is no known optimum value because of the preparation times in this problem, the results are compared with lower bound as in the literature. (Seyed-Alagheband et al., 2011). At the SALBPs-II problem setup times are added to the optimum cycle times known for SALBPs-II. The lower bound is calculated by the formula given (4.26) by adding the setup time value to the optimum time known for SALBP-II. The theoretical cycle time is used when the optimum cycle time is not given which is required for this calculation.

$$LB = Opt + \frac{\sum(N - m + 1)_{\text{lowest setup times}}}{m} \quad (4.26)$$

After obtaining the results for SALBP-II, SALBPs-II problems were solved with the algorithm developed. The algorithm is operated according to Arcus1, Arcus2, Heskia, Jackson, Lutz1, Mitchell, Sawyer, Tonge and Wee-Mag precedence relationship diagrams. In this operation, the parameters used for the SALBP-II problem are used exactly. Only the setup parameter is entered 1 to take account of setup time. The results obtained for each precedence diagram and summary of average results is presented below.

Jackson precedence relationship diagram contains 11 tasks and task times vary from 1 to 7. 3 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 3 to 5. The results obtained is shown in table 4.20.

Table 4.20 Results for the Jackson problem set with LV setup times

#Station	Lower Bound	Result	c%dev
3	16.23	15.37	5.57
4	12.29	11.53	6.64
5	10.26	9.22	11.33
Average			7.85

With low variability setup times at the Jackson precedence diagram the average deviation from the lower bound is %7.85.

Mitchell precedence relationship diagram contains 21 tasks and task times vary from 1 to 13. 5 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 3 to 7. The results obtained is shown in table 4.21. With low variability setup times at the Mitchell precedence diagram the average deviation from the lower bound is %4.22.

Table 4.21 Results for the Mitchell problem set with LV setup times

#Station	Lower Bound	Result	c%dev
3	35.75	35.03	2.06
4	27.49	26.27	4.64
5	21.47	21.01	2.17
6	18.26	17.51	4.28
7	16.20	15.01	7.95
Average			4.22

Heskiaoff precedence relationship diagram contains 28 tasks and task times vary from 1 to 108. 7 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 3 to 9. The results obtained is shown in table 4.22. With low variability setup times at the Heskiaoff precedence diagram the average deviation from the lower bound is %0.77.

Table 4.22 Results for the Heskiaoff problem set with LV setup times

#Station	Lower Bound	Result	c%dev
3	342.28	341.36	0.27
4	256.60	256.02	0.23
5	205.48	204.81	0.33
6	171.39	170.68	0.42
7	147.24	146.29	0.65
8	129.23	128.00	0.96
9	116.25	113.78	2.17
Average			0.72

Lutz1 precedence relationship diagram contains 32 tasks and task times vary from 100 to 1400. 5 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 8 to 12. The results obtained is shown in table 4.23. With low variability setup times at the Lutz1 precedence diagram the average deviation from the lower bound is %1.70.

Table 4.23 Results for the Lutz1 problem set with LV setup times

#Station	Lower Bound	Result	c%dev
8	1919.00	1860.00	3.17
9	1677.00	1638.00	2.38
10	1571.00	1526.00	2.95
11	1400.00	1400.00	0.00
12	1400.00	1400.00	0.00
Average			1.70

Sawyer precedence relationship diagram contains 30 tasks and task times vary from 1 to 25. 8 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 7 to 14. The results obtained is shown in table 4.24. With low variability setup times at the Sawyer precedence diagram the average deviation from the lower bound is %1.49.

Table 4.24 Results for the Sawyer problem set with LV setup times

#Station	Lower Bound	Result	c%dev
7	48.37	47.01	2.89
8	41.40	41.01	0.95
9	37.40	37.01	1.06
10	34.07	34.01	0.19
11	32.29	31.00	4.15
12	28.25	28.00	0.88
13	26.32	26.00	1.22
14	25.15	25.00	0.59
Average			1.49

Wee-Mag precedence relationship diagram contains 75 tasks and task times vary from 2 to 27. 28 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 3 to 30. The results obtained is shown in table 4.25. With low variability setup times at the Wee-Mag precedence diagram the average deviation from the lower bound is %1.01.

Table 4.25 Results for the Wee-Mag problem set with LV setup times

#Station	Lower Bound	Result	c%dev
3	502.78	500.02	0.55
4	377.98	375.01	0.79
5	302.76	300.01	0.92
6	252.69	250.01	1.07
7	216.70	215.00	0.79
8	189.91	188.00	1.02
9	168.89	167.00	1.13
10	152.06	150.00	1.37
11	138.38	137.00	1.01
12	126.72	125.00	1.38
13	117.28	116.00	1.10
14	109.67	108.00	1.55
15	101.73	100.00	1.73
16	95.25	94.00	1.33
17	89.93	89.00	1.04
18	87.49	87.00	0.56
19	85.38	85.00	0.45
20	77.58	77.00	0.75
21	73.01	72.00	1.40
22	70.31	69.00	1.90
23	67.75	67.00	1.12
24	66.61	66.00	0.92
25	66.27	65.00	1.95
26	65.48	65.00	0.74
27	65.21	65.00	0.32
28	64.39	64.00	0.61
29	63.12	63.00	0.19
30	56.28	56.00	0.50
Average			1.01

Tonge precedence relationship diagram contains 70 tasks and task times vary from 1 to 156. 23 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 3 to 25. The results obtained is shown in table 4.26. With low variability setup times at the Tonge precedence diagram the average deviation from the lower bound is %0.66.

Table 4.26 Results for the Tonge problem set with LV setup times

#Station	Lower Bound	Result	c%dev
3	1172.29	1170.00	0.20
4	879.33	878.00	0.15
5	703.73	702.00	0.25
6	586.94	585.00	0.33
7	503.56	502.00	0.31
8	441.43	439.00	0.55
9	391.91	391.00	0.23
10	353.55	352.00	0.44
11	321.11	320.00	0.35
12	294.61	294.00	0.21
13	274.22	271.00	1.19
14	253.76	251.00	1.10
15	237.53	235.00	1.08
16	223.65	221.00	1.20
17	209.45	208.00	0.70
18	199.46	196.00	1.77
19	187.32	186.00	0.71
20	179.51	177.00	1.42
21	173.35	170.00	1.97
22	163.27	162.00	0.78
23	156.48	156.00	0.31
24	156.00	156.00	0.00
25	156.00	156.00	0.00
Average			0.66

Arcus1 precedence relationship diagram contains 83 tasks and task times vary from 233 to 3691. 16 different instances of this precedence relationship diagram are solved with developed algorithm that number of stations (*#Station*) varies from 3 to 18. The results obtained is shown in table 4.27. With low variability setup times at the Arcus1 precedence diagram the average deviation from the lower bound is %3.62.

Table 4.27 Results for the Arcus1 problem set with LV setup times

#Station	Lower Bound	Result	c%dev
3	25767.00	25242.33	2.08
4	19397.00	18931.50	2.46
5	15617.00	15145.40	3.11
6	13052.00	12622.67	3.40
7	11230.00	10828.14	3.71
8	9866.00	9555.75	3.25
9	8717.00	8500.44	2.55
10	7885.00	7581.20	4.01
11	7227.00	7085.00	2.00
12	6669.00	6412.83	3.99
13	6099.00	5864.69	4.00
14	5668.00	5441.57	4.16
15	5364.00	5104.47	5.08
16	5082.00	4850.38	4.78
17	4766.00	4516.29	5.53
18	4485.00	4317.22	3.89
Average			3.62

Summary of all precedence diagram solved with LV setup times is presented in table 4.28.

Table 4.28 Summary of computational results for the SALBPs-II (LV)

Data	Average c%dev
Arcus1	3.62
Heskiaoff	0.72
Jackson	7.85
Lutz1	1.70
Mitchell	4.22
Sawyer	1.49
Tonge	0.66
Wee-Mag	1.01
Average	2.66

As can be seen in the table 4.28 the developed algorithm has been solved the LV problems with %2.66 average deviation from the lower bound.

The algorithm is operated with the same precedence relationship diagrams with same station number for high variability (HV) setup times. The results for Jackson, Mitchell, Heskiaoff, Lutz1, Arcus1, Wee-Mag, Tonge and Sawyer precedence relationship diagrams is shown in table 4.29-4.36 respectively.

Table 4.29 Results for the Jackson problem set with HV setup times

#Station	Lower Bound	Result	c%dev
3	15.45	16.98	9.88
4	11.58	13.49	16.54
5	9.25	10.75	16.24
Average			14.22

Table 4.30 Results for the Mitchell problem set with HV setup times

#Station	Lower Bound	Result	c%dev
3	35.12	37.13	5.71
4	26.33	28.92	9.83
5	21.06	22.17	5.27
6	17.55	19.14	9.09
7	15.03	17.00	13.07
Average			8.60

Table 4.31 Results for the Heskiaoff problem set with HV setup times

#Station	Lower Bound	Result	c%dev
3	341.47	343.00	0.45
4	256.09	257.73	0.64
5	204.87	206.71	0.90
6	170.72	172.26	0.90
7	146.33	147.87	1.06
8	128.03	129.74	1.33
9	113.80	117.84	3.55
Average			1.26

Table 4.32 Results for the Lutz1 problem set with HV setup times

#Station	Lower Bound	Result	c%dev
8	1861.25	1983.00	6.54
9	1639.00	1751.00	6.83
10	1526.80	1632.00	6.89
11	1400.64	1448.00	3.38
12	1400.50	1400.00	-0.04
Average			4.72

As seen in table 4.32, the algorithm found a better solution than lower bound for Lutz1 precedence diagram with 12 stations. One task with 1400 processing time stands alone in a station and this task determines cycle time. When calculating lower bound, the calculated lower setup time (0.64) was added to the known best solution (1400). This is why lower bound is calculated higher than expected.

Table 4.33 Results for the Arcus1 problem set with HV setup times

#Station	Lower Bound	Result	c%dev
3	25263.33	26925.00	6.58
4	18947.00	20342.00	7.36
5	15157.60	16288.00	7.46
6	12632.67	13548.00	7.25
7	10836.57	11825.00	9.12
8	9563.00	10455.00	9.33
9	8506.78	9264.00	8.90
10	7586.80	8318.00	9.64
11	7090.00	7580.00	6.91
12	6417.33	7187.00	11.99
13	5868.77	6510.00	10.93
14	5445.29	5976.00	9.75
15	5107.87	5579.00	9.22
16	4853.50	5329.00	9.80
17	4519.18	5022.00	11.13
18	4319.89	4820.00	11.58
Average			9.18

Table 4.34 Results for the Wee-Mag problem set with HV setup times

#Station	Lower Bound	Result	c%dev
3	500.24	508.66	1.68
4	375.18	383.99	2.35
5	300.14	308.17	2.68
6	250.11	257.70	3.03
7	215.09	221.05	2.77
8	188.08	194.42	3.37
9	167.07	172.56	3.29
10	150.06	155.85	3.86
11	137.05	141.81	3.47
12	125.05	130.00	3.96
13	116.04	120.30	3.67
14	108.04	112.20	3.85
15	100.03	104.56	4.53
16	94.03	97.64	3.84
17	89.03	92.40	3.79
18	87.02	88.83	2.08
19	85.02	86.84	2.14
20	77.02	79.21	2.84
21	72.02	74.64	3.64
22	69.02	71.93	4.22
23	67.01	69.72	4.04
24	66.01	68.55	3.84
25	65.01	67.52	3.86
26	65.01	66.49	2.27
27	65.01	66.17	1.78
28	64.01	65.09	1.69
29	63.01	63.19	0.29
30	56.01	57.49	2.64
Average			3.05

Table 4.35 Results for the Tonge problem set with HV setup times

#Station	Lower Bound	Result	c%dev
3	1170.12	1175.16	0.43
4	878.09	882.05	0.45
5	702.07	706.22	0.59
6	585.06	589.45	0.75
7	502.05	505.25	0.64
8	439.04	442.75	0.85
9	391.03	394.48	0.88
10	352.03	355.18	0.90
11	320.03	322.99	0.93
12	294.02	296.98	1.01
13	271.02	274.32	1.22
14	251.02	253.97	1.18
15	235.02	238.44	1.46
16	221.01	224.08	1.39
17	208.01	211.49	1.67
18	196.01	199.58	1.82
19	186.01	189.69	1.98
20	177.01	183.19	3.49
21	170.01	175.62	3.30
22	162.01	165.69	2.27
23	156.01	158.52	1.61
24	156.01	156.00	0.00
25	156.01	156.00	0.00
Average			1.25

Table 4.36 Results for the Sawyer problem set with HV setup times

#Station	Lower Bound	Result	c%dev
7	47.04	49.05	4.28
8	41.03	42.14	2.70
9	37.03	38.50	3.98
10	34.02	34.85	2.44
11	31.02	32.45	4.62
12	28.01	29.00	3.52
13	26.01	26.92	3.49
14	25.01	25.42	1.64
Average			3.33

Summary of all precedence diagram solved with LV setup times is presented in table 4.37.

Table 4.37 Summary of the computational results for the SALBPs-II (HV)

Data	Average c%dev
Arcus1	9.18
Heskiaoff	1.26
Jackson	14.22
Lutz1	4.72
Mitchell	8.60
Sawyer	3.33
Tonge	1.25
Wee-Mag	3.05
Average	5.43

As can be seen in the table 4.28 the developed algorithm has been solved the problems with high variability setup times with %5.43 average deviation from the lower bound.

CHAPTER FIVE

CONCLUSION

Reducing costs for manufacturers is one of the most important issues today. The reason for this is the increasing competition conditions with the developing technology. One of the best ways to achieve this goal is to focus on assembly lines that are used very often in production. Therefore, Assembly Line Balancing Problems (ALBP) covers a wide range in the literature. Although the most recent studies in the literature have focused on finding solutions to current life problems under various classifications by developing existing methods, newly developed algorithms are often studied on this simplified version of the problem. For this reason, in this study, Simple Assembly Line Balancing Problem (SALBP) has been studied and an Adaptive Large Neighborhood Search (ALNS) algorithm has been developed for Type Two Simple Assembly Line Balancing Problems(SALBP-II).

Large Neighborhood Search (LNS) algorithm is an algorithm firstly presented for Vehicle Routing Problem (Shaw, 1997). This algorithm has demonstrated more successful results than other algorithms due to its neighborhood structure on VRP. In VRP, a certain number of customers are assigned to a certain number of warehouses and routes are created. In SALBP-II, assignments are made in such a way that a certain number of tasks are performed at a certain number of stations. There are structural similarities between these two problems, only difference is in Assembly Line Balancing Problem, precedence relationship should be considered. The motivation of this study is that with this structural similarity, the great neighborhood search algorithm will be successful for ALBP.

The LNS algorithm is an algorithm consisting of removal and insertion operators. While adapting this algorithm to the ALBP, removal and insertion operators have been restructured by considering the precedence relationship. The algorithm was inspired by the adaptive version of LNS previously presented for VRP (Ropke & Pisinger, 2006). In this adaptive structure, 5 different methods for the removal operator and 16 different methods for the insertion operator have been defined. The algorithm basically starts with an initial solution. In each iteration, in the removal operator, a relaxed

solution obtained by removing some solution components from the solution. In the insertion operator, the removed solution components are added to the different locations in the relaxed solution and the new solution is obtained. The new solution adoption process is similar to the Simulated Annealing algorithm. The success of the methods used in the removal and insertion operators are kept and in this way the algorithm is provided for selecting successful methods.

The developed algorithm was tested on benchmark problems. As can be seen from the results listed in chapter four, the algorithm has achieved very successful results. In this study, the ALNS algorithm has been proven suitable for ALBP. The success of the algorithm on the SALBP is an indicator of the success of this algorithm on other ALB problems. This success of the ALNS algorithm, which can be applied easily for assembly line balancing problems, which is an important topic for reducing costs, will also shed light on future studies.

REFERENCES

- Adulyasak, Y., Cordeau, J. F., & Jans, R. (2012). Optimization-based adaptive large neighborhood search for the production routing problem. *Transportation Science*, 48(1), 20-45.
- Akpınar, Ş. (2009). *A hybrid genetic algorithm for mixed-model assembly line balancing problem with parallel workstation assignment*. Master Thesis, Dokuz Eylül University, İzmir.
- Akpınar, Ş. (2017). Large neighbourhood search algorithm for type-II assembly line balancing problem. *Pamukkale University Journal of Engineering Sciences*, 23(4), 444-450.
- Aksen, D., Kaya, O., Salman, F. S., & Tüncel, Ö. (2014). An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem. *European Journal of Operational Research*, 239(2), 413-426.
- Anderson, E. J., & Ferris, M. C. (1994). Genetic algorithms for combinatorial optimization: the assemble line balancing problem. *ORSA Journal on Computing*, 6(2), 161-173.
- Azi, N., Gendreau, M., & Potvin, J. Y. (2014). An adaptive large neighborhood search for a vehicle routing problem with multiple routes. *Computers & Operations Research*, 41, 167-173.
- Azizoğlu, M., & Imat, S. (2018). Workload smoothing in simple assembly line balancing. *Computers & Operations Research*, 89, 51-57.
- Bartholdi, J. J. (1993). Balancing two-sided assembly lines: a case study. *International Journal of Production Research*, 31(10), 2447-2461.

- Battaia, O., & Dolgui, A. (2013). A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, 142(2), 259-277.
- Baybars, I. (1986). A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, 32(8), 909-932.
- Baykasoglu, A. (2006). Multi-rule multi-objective simulated annealing algorithm for straight and U type assembly line balancing problems. *Journal of Intelligent Manufacturing*, 17(2), 217-232.
- Becker, C., & Scholl, A. (2006). A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, 168(3), 694-715.
- Belo-Filho, M. A. F., Amorim, P., & Almada-Lobo, B. (2015). An adaptive large neighbourhood search for the operational integrated production and distribution problem of perishable products. *International Journal of Production Research*, 53(20), 6040-6058.
- Blum, C. (2010). Iterative beam search for simple assembly line balancing with a fixed number of work stations. *arXiv preprint arXiv:1012.3273*, 1-19.
- Blum, C., & Miralles, C. (2011). On solving the assembly line worker assignment and balancing problem via beam search. *Computers & Operations Research*, 38(1), 328-339.
- Boysen, N., & Fliedner, M. (2008). A versatile algorithm for assembly line balancing. *European Journal of Operational Research*, 184(1), 39-56.
- Boysen, N., Fliedner, M., & Scholl, A. (2007). A classification of assembly line balancing problems. *European Journal of Operational Research*, 183(2), 674-693.

- Boysen, N., Fliedner, M., & Scholl, A. (2008). Assembly line balancing: Which model to use when? *International Journal of Production Economics*, 111(2), 509-528.
- Chiang, W. C. (1998). The application of a tabu search metaheuristic to the assembly line balancing problem. *Annals of Operations Research*, 77, 209-227.
- Demir, E., Bektaş, T., & Laporte, G. (2012). An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research*, 223(2), 346-359.
- Dominguez, O., Guimarans, D., Juan, A. A., & de la Nuez, I. (2016). A biased-randomised large neighbourhood search for the two-dimensional vehicle routing problem with backhauls. *European Journal of Operational Research*, 255(2), 442-462.
- Erel, E., & Sarin, S. C. (1998). A survey of the assembly line balancing procedures. *Production Planning & Control*, 9(5), 414-434.
- Eskandarpour, M., Dejax, P., & Péton, O. (2017). A large neighborhood search heuristic for supply chain network design. *Computers & Operations Research*, 80, 23-37.
- François, V., Arda, Y., Crama, Y., & Laporte, G. (2016). Large neighborhood search for multi-trip vehicle routing. *European Journal of Operational Research*, 255(2), 422-441.
- Ghilas, V., Demir, E., & Van Woensel, T. (2016). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows and scheduled lines. *Computers & Operations Research*, 72, 12-30.
- Ghosh, S., & Gagnon, R. J. (1989). A comprehensive literature review and analysis of the design, balancing and scheduling of assembly systems. *The International Journal of Production Research*, 27(4), 637-670.

- Gonçalves, J. F., & De Almeida, J. R. (2002). A hybrid genetic algorithm for assembly line balancing. *Journal of Heuristics*, 8(6), 629-642.
- Grimault, A., Bostel, N., & Lehuédé, F. (2017). An adaptive large neighborhood search for the full truckload pickup and delivery problem with resource synchronization. *Computers & Operations Research*, 88, 1-14.
- Hà, M. H., Nguyen, T. D., Duy, T. N., Pham, H. G., Do, T., & Rousseau, L. M. (2020). A new constraint programming model and a linear programming-based adaptive large neighborhood search for the vehicle routing problem with synchronization constraints. *Computers & Operations Research*, 124, 105085.
- Hackman, S. T., Magazine, M. J., & Wee, T. S. (1989). Fast, effective algorithms for simple assembly line balancing problems. *Operations Research*, 37(6), 916-924.
- Hammami, F., Rekik, M., & Coelho, L. C. (2020). A hybrid adaptive large neighborhood search heuristic for the team orienteering problem. *Computers & Operations Research*, 123, 105034.
- Hazır, Ö., Agi, M., & Guérin, J. (2019). A Problem Specific Heuristic for Smoothing the Workload in Simple Assembly Lines. *IFAC-PapersOnLine*, 52(13), 2562-2565
- Heinrici, A. (1994). A comparison between simulated annealing and tabu search with an example from the production planning. In *Operations Research Proceedings 1993* (498-503). Berlin, Heidelberg: Springer.
- Hemmelmayr, V. C., Cordeau, J. F., & Crainic, T. G. (2012). An adaptive large neighborhood search heuristic for two-echelon vehicle routing problems arising in city logistics. *Computers & Operations Research*, 39(12), 3215-3228.
- Kilinci, O. (2010). A Petri net-based heuristic for simple assembly line balancing problem of type 2. *The International Journal of Advanced Manufacturing Technology*, 46(1-4), 329-338.

- Kim, Y. J., Kim, Y. K., & Cho, Y. (1998). A heuristic-based genetic algorithm for workload smoothing in assembly lines. *Computers & Operations Research*, 25(2), 99-111.
- Kim, Y. K., Kim, Y. J., & Kim, Y. (1996). Genetic algorithms for assembly line balancing with various objectives. *Computers & Industrial Engineering*, 30(3), 397-409.
- Klein, R., & Scholl, A. (1996). Maximizing the production rate in simple assembly line balancing—a branch and bound procedure. *European Journal of Operational Research*, 91(2), 367-385.
- Koç, Ç., & Laporte, G. (2018). Vehicle routing with backhauls: Review and research perspectives. *Computers & Operations Research*, 91, 79-91.
- Korsvik, J. E., Fagerholt, K., & Laporte, G. (2011). A large neighbourhood search heuristic for ship routing and scheduling with split loads. *Computers & Operations Research*, 38(2), 474-483.
- Kovacs, A. A., Parragh, S. N., Doerner, K. F., & Hartl, R. F. (2012). Adaptive large neighborhood search for service technician routing and scheduling problems. *Journal of Scheduling*, 15(5), 579-600.
- Kriengkorakot, N., & Pianthong, N. (2007). The assembly line balancing problem: review articles. *Engineering and Applied Science Research*, 34(2), 133-140.
- Laborie, P., & Godard, D. (2007). Self-adapting large neighborhood search: Application to single-mode scheduling problems. *Proceedings MISTA-07, Paris*, 8, 276-284.
- Lahyani, R., Gouguenheim, A. L., & Coelho, L. C. (2019). A hybrid adaptive large neighbourhood search for multi-depot open vehicle routing problems. *International Journal of Production Research*, 1-14.

- Laporte, G., Musmanno, R., & Vocaturo, F. (2010). An adaptive large neighbourhood search heuristic for the capacitated arc-routing problem with stochastic demands. *Transportation Science*, 44(1), 125-135.
- Lei, D., & Guo, X. (2016). Variable neighborhood search for the second type of two-sided assembly line balancing problem. *Computers & Operations Research*, 72, 183-188.
- Li, Y., Chen, H., & Prins, C. (2016). Adaptive large neighborhood search for the pickup and delivery problem with time windows, profits, and reserved requests. *European Journal of Operational Research*, 252(1), 27-38.
- Liu, R., Tao, Y., & Xie, X. (2019). An adaptive large neighborhood search heuristic for the vehicle routing problem with time windows and synchronized visits. *Computers & Operations Research*, 101, 250-262.
- Liu, S. B., Ong, H. L., & Huang, H. C. (2003). Two bi-directional heuristics for the assembly line type II problem. *The International Journal of Advanced Manufacturing Technology*, 22(9-10), 656-661.
- Luo, Z., Qin, H., Zhang, D., & Lim, A. (2016). Adaptive large neighborhood search heuristics for the vehicle routing problem with stochastic demands and weight-related cost. *Transportation Research Part E: Logistics and Transportation Review*, 85, 69-89.
- Majidi, S., Hosseini-Motlagh, S. M., & Ignatius, J. (2018). Adaptive large neighborhood search heuristic for pollution-routing problem with simultaneous pickup and delivery. *Soft Computing*, 22(9), 2851-2865.
- Masson, R., Lehuédé, F., & Péton, O. (2013). An adaptive large neighborhood search for the pickup and delivery problem with transfers. *Transportation Science*, 47(3), 344-355.

- Michels, A. S., Lopes, T. C., & Magatão, L. (2020). An exact method with decomposition techniques and combinatorial Benders' cuts for the type-2 multi-manned assembly line balancing problem. *Operations Research Perspectives*, 7, 100163.
- Muller, L. F., Spoorendonk, S., & Pisinger, D. (2012). A hybrid adaptive large neighborhood search heuristic for lot-sizing with setup times. *European Journal of Operational Research*, 218(3), 614-623.
- Nagata, Y., & Bräysy, O. (2009). A powerful route minimization heuristic for the vehicle routing problem with time windows. *Operations Research Letters*, 37(5), 333-338.
- Nearchou, A. C. (2007). Balancing large assembly lines by a new heuristic based on differential evolution method. *The International Journal of Advanced Manufacturing Technology*, 34(9-10), 1016-1029.
- Nearchou, A. C. (2011). Maximizing production rate and workload smoothing in assembly lines using particle swarm optimization. *International Journal of Production Economics*, 129(2), 242-250.
- Palpant, M., Artigues, C., & Michelon, P. (2004). LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131(1-4), 237-257.
- Pastor, R., & Ferrer, L. (2009). An improved mathematical program to solve the simple assembly line balancing problem. *International Journal of Production Research*, 47(11), 2943-2959.
- Pastor, R., Ferrer, L., & García, A. (2007). Evaluating optimization models to solve SALBP. In *International Conference on Computational Science and Its Applications* (791-803). Berlin, Heidelberg: Springer.

- Perron, L. (2003). Fast restart policies and large neighborhood search. *Principles and Practice of Constraint Programming at CP*, 2833, 1-15.
- Pisinger, D., & Ropke, S. (2007). A general heuristic for vehicle routing problems. *Computers & Operations Research*, 34(8), 2403-2435.
- Polat, O., Kalayci, C. B., Mutlu, Ö., & Gupta, S. M. (2016). A two-phase variable neighbourhood search algorithm for assembly line worker assignment and balancing problem type-II: an industrial case study. *International Journal of Production Research*, 54(3), 722-741.
- Ribeiro, G. M., & Laporte, G. (2012). An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers & Operations Research*, 39(3), 728-735.
- Ropke, S., & Pisinger, D. (2006). An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4), 455-472.
- Roshani, A., Roshani, A., Roshani, A., Salehi, M., & Esfandyari, A. (2013). A simulated annealing algorithm for multi-manned assembly line balancing problem. *Journal of Manufacturing Systems*, 32(1), 238-247.
- Salveson, M. E. (1955). The assembly line balancing problem. *The Journal of Industrial Engineering*, 18-25.
- Santini, A. (2019). An adaptive large neighbourhood search algorithm for the orienteering problem. *Expert Systems with Applications*, 123, 154-167.
- Scholl, A. (1999). *Balancing and sequencing of assembly lines*. Physica-Verlag: Heidelberg.

- Scholl, A., & Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, 168(3), 666-693.
- Scholl, A., & Voß, S. (1997). Simple assembly line balancing—Heuristic approaches. *Journal of Heuristics*, 2(3), 217-244.
- Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., & Dueck, G. (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2), 139-171.
- Seyed-Alagheband, S. A., Ghomi, S. F., & Zandieh, M. (2011). A simulated annealing algorithm for balancing the assembly line type II problem with sequence-dependent setup times between tasks. *International Journal of Production Research*, 49(3), 805-825.
- Shaw, P. (1997). *A new local search algorithm providing high quality solutions to vehicle routing problems*. Technical Report, APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK.
- Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In *International Conference On Principles And Practice Of Constraint Programming* (417-431). Berlin, Heidelberg: Springer.
- Sivasankaran, P., & Shahabudeen, P. (2014). Literature review of assembly line balancing problems. *The International Journal of Advanced Manufacturing Technology*, 73(9-12), 1665-1694.
- Sze, J. F., Salhi, S., & Wassan, N. (2016). A hybridization of adaptive variable neighbourhood search and large neighbourhood search: Application to the vehicle routing problem. *Expert Systems with Applications*, 65, 383-397.

- Ticha, H. B., Absi, N., Feillet, D., & Quilliot, A. (2019). Multigraph modeling and adaptive large neighborhood search for the vehicle routing problem with time windows. *Computers & Operations Research*, 104, 113-126.
- Uğurdağ, H. F., Rachamadugu, R., & Papachristou, C. A. (1997). Designing paced assembly lines with fixed number of stations. *European Journal of Operational Research*, 102(3), 488-501.
- Watanabe, T., Hashimoto, Y., Nishikawa, I., & Tokumaru, H. (1995). Line balancing using a genetic evolution model. *Control Engineering Practice*, 3(1), 69-76.
- Yilmazlar, I. O., Jeyes, A., Fiore, A., Patel, A., Spence, C., Wentzky, C., ... & Taaffe, K. M. (2020). A Case Study in Line Balancing and Simulation. *Procedia Manufacturing*, 48, 71-81.
- Zhang, H., Yan, Q., Liu, Y., & Jiang, Z. (2016). An integer-coded differential evolution algorithm for simple assembly line balancing problem of type 2. *Assembly Automation*, 36(3), 246-261.
- Zheng, Q., Li, M., Li, Y., & Tang, Q. (2013). Station ant colony optimization for the type 2 assembly line balancing problem. *The International Journal of Advanced Manufacturing Technology*, 66(9-12), 1859-1870.