DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

SOLVING ASSEMBLY LINE BALANCING PROBLEM WITH POSITIONAL CONSTRAINTS AND WORKER ASSIGNMENTS USING MATHEMATICAL PROGRAMMING AND HEURISTIC SOLUTION APPROACHES

by
Raziye OKYAY

March, 2018
İZMİR

SOLVING ASSEMBLY LINE BALANCING PROBLEM WITH POSITIONAL CONSTRAINTS AND WORKER ASSIGNMENTS USING MATHEMATICAL PROGRAMMING AND HEURISTIC SOLUTION APPROACHES

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

Raziye OKYAY

March, 2018

İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"SOLVING ASSEMBLY LINE BALANCING PROBLEM WITH POSITIONAL CONSTRAINTS AND WORKER ASSIGNMENTS USING MATHEMATICAL PROGRAMMING AND HEURISTIC SOLUTION APPROACHES"** completed by **RAZİYE OKYAY** under supervision of **PROF. DR. ŞEYDA TOPALOĞLU YILDIZ** 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.

Prof. Dr. Şeyda TOPALOĞLU YILDIZ

Supervisor

Doç. Dr. Özcan Kılınçcı

Jury Member

Dr. Öğr. Ü. M. Ali İLGİN

Jury Member

Prof. Dr. Kadriye ERTEKİN

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

First of all, I would like to give my profound thanks to my supervisor Prof. Dr. Şeyda TOPALOĞLU YILDIZ for her continuous support, invaluable advice and understanding throughout my M.Sc. thesis. Thanks to her guidance, profound knowledge, and motivation, I was able to overcome challenges.

I would also like to express my thanks to my friends Burcu KUBUR and Tuğçe DABANLI for motivating me in this period.

I would like to express my gratitude to my parents, Raşit OKYAY, Hatice OKYAY, and my siblings, Ebru, Damla and Uğur for their love and endless support throughout my life.

Raziye OKYAY

SOLVING ASSEMBLY LINE BALANCING PROBLEM WITH POSITIONAL CONSTRAINTS AND WORKER ASSIGNMENTS USING MATHEMATICAL PROGRAMMING AND HEURISTIC SOLUTION APPROACHES

ABSTRACT

There are many studies in the literature on assembly line balancing (ALB) problems. The ALB problems differ from each other in various aspects. In this thesis, we consider the ALB problem with hierarchical worker assignment, positional constraints, station paralleling options, and task assignment restrictions. The objective of this ALB problem is to decide on the number of parallel stations to be opened in each work stage and to assign tasks and workers to stations such that the sum of station opening costs and worker costs is minimized. To solve this problem, we initially propose an integer programming (IP) model, and then develop a simulated annealing (SA) algorithm to obtain high-quality solutions in reasonable computational times. For the SA algorithm firstly, a modified version of the Rank Positional Weight heuristic is developed to generate an initial solution. To generate new different solutions from the current solution, four kinds of neighborhood search structures are used which are single_transfer, two_transfer, swap and stage separation. In order to enhance the solution quality of SA algorithm, it is hybridized with a local search. We use a giant leap procedure to investigate an inferior or unvisited search space for the probability of finding a better solution. To find the optimal parameters of the SA algorithm, we employ the Taguchi method. Thus, the solution quality and the running time of the SA algorithm improve. A set of test problems are solved using both the proposed IP model and SA algorithm. The computational results show the effectiveness of SA algorithm.

Keywords: Assembly line balancing, hierarchical worker assignment, integer programming, simulated annealing, parallel stations, positional constraints

POZİSYONEL VE İŞÇİ ATAMA KISITLARI İLE MONTAJ HATTI DENGELEME PROBLEMİNİN MATEMATİKSEL PROGRAMLAMA VE SEZGİSEL ÇÖZÜM YAKLAŞIMLARI İLE ÇÖZÜMÜ

ÖZ

Montaj hattı dengeleme (MHD) problemleri üzerine literatürde birçok çalışma bulunmaktadır. Bu MHD problemleri çeşitli açılardan birbirlerinden farklıdır. Bu tezde, MHD problemini hiyerarşik işçi ataması, pozisyonel kısıtlar, istasyon paralelleme seçenekleri ve iş atama kısıtlamaları ile birlikte ele alınmıştır. Bu MHD probleminin amacı, istasyonların açılış maliyetleri ve işçi maliyetlerinin toplamını en aza indirilecek şekilde; her bir çalışma aşamasında açılacak paralel istasyonların sayısına ve işleri ve işçileri istasyonlara atanmasına karar vermektir. Bu sorunu çözmek için ilk olarak bir tamsayılı programlama (TP) modeli önerilmiş ve ardından makul hesaplama zamanlarında yüksek kaliteli çözümler elde etmek için bir benzetilmiş tavlama (BT) algoritması geliştirilmiştir. İlk olarak BT algoritması için, Pozisyon Ağırlığı sezgiselinin değiştirilmiş bir versiyonu başlangıç çözüm üretmek için geliştirilmiştir. Mevcut çözümden yeni farklı çözümler üretmek için, tekli_ transfer, ikili_transfer, yer değiştirme ve aşama ayırma olmak üzere dört çeşit komşu arama yapısı kullanılmıştır. Algoritmamızın çözüm kalitesini yükseltmek için BT algoritması bir yerel arama ile hibridize edilmiştir. Daha iyi bir çözüm bulabilmek için veya incelenmemiş bir arama alanını araştırmak için bir dev sıçrama prosedürü kullanıyoruz. BT algoritmasının optimum parametrelerini bulmak için Taguchi yöntemi kullanılmıştır. Böylece, BT algoritmasının çözüm kalitesi ve çalışma süresi iyileştirilmiştir. Bir dizi test problemi önerilen TP modeli ve BT algoritması kullanılarak çözülmüştür. Hesaplama sonuçları BT algoritmasının etkililiğini göstermektedir.

Anahtar kelimeler: Montaj hattı dengeleme, hiyerarşik işçi atama, matematiksel modelleme, benzetim tavlaması, paralel istasyon, pozisyonel kısıt

CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER ONE
INTRODUCTION

An assembly line is a manufacturing process in which the semi-finished assembly moves from workstation to workstation where the parts are added in sequence until the final assembly is produced. Assembly lines are flow oriented production systems which are of great importance in the industrial production of high quantity standardized products. The workpieces visit stations successively as they are moved along the line usually by some kind of transportation system, e.g., a conveyor belt. Recently, assembly lines have also gained importance in low volume production of customized products. Originally, assembly lines were developed for a cost efficient mass-production of standardized products, designed to exploit a high specialization of labour and the associated learning effects (Shtub & Dar-El, 1989; Scholl, 1999). They are common methods of assembling complex items such as automobiles and other transportation equipment, household appliances and electronic goods.

An assembly line consists of serial stages connected with a material handling system. To produce finished products, certain assembly tasks are performed repeatedly at each stage. Some restrictions such as precedence constraints, processing times of tasks, number of workstations, cycle time, incompatibility relations between tasks are taken into consideration while tasks are assigned to single or duplicated workstations. The decision problem of optimally partitioning (balancing) the assembly work among the stations with respect to some objective is known as the assembly line balancing (ALB) problem. According to the objective function, the simple assembly line balancing (SALB) problems are divided into two basic classes: the SALB-1 and SALB-2. The SALB-1 problem minimizes the number of workstations with a specified cycle time, while the SALB-2 problem minimizes the cycle time with a specified number of workstations.

The traditional assembly line usually includes features such as production of one product, fixed task duration, one sided serial line layout etc. However, the real life assembly lines may differ from the traditional assembly lines in some aspects. For

this reason, recent researches includes some features such as zoning constraints, task assignment restrictions, parallel stations, production of multi model product, different line configurations etc. in accordance with the real life assembly line. In addition to these, due to the fact that processing times of tasks differ according to which worker performs the task, the assembly line worker assignment and balancing (ALWAB) problems have arisen. The reason of this difference can be skill, effort, experience etc. In addition to these problems, a hierarchical workforce structure in which the qualification levels of the workers are ranked hierarchically has been incorporated into assembly line balancing by Sungur & Yavuz (2015). This problem is known as assembly line balancing with hierarchical worker assignment (ALBHW) problem.

1.1 Aim of the Thesis

This thesis focuses on the ALB problem with hierarchical worker assignment, positional constraints, station paralleling and task assignment restrictions. The objective of this ALB problem is to find the optimal assignment of workers and tasks to the stations, and to decide on the optimal parallel stations such that the total cost comprised of station paralleling costs and worker costs is minimized. In previous studies in the literature, the ALB problem that considers all these realistic situations of the assembly line at the same time has not been considered yet. For this reason, in this thesis, we firstly develop a novel integer programming (IP) model that encompasses all these restrictions. Then, we propose a simulated annealing (SA) algorithm in order to obtain efficient solutions for larger scale problems that cannot be solved by the IP model. Finally, the performances of the proposed IP model and the SA algorithm are compared with each other in terms of solution quality and time.

1.2 Outline of the Thesis

This thesis consists of six chapters. The remainder of the thesis is organized as follows:

In chapter two, we initially give general information about assembly lines, basic terms of assembly lines, ALB problems and the classification of ALB problems.

Then, the assembly line worker assignment and balancing problem, and the assembly line balancing problem with hierarchical worker assignment are described and the corresponding mathematical models to these problems, which are proposed by Miralles, Garcia-Sabater, Andres & Cardos (2007) and Sugur & Yavuz (2015) are given, respectively. Finally, the literature review of the ALB problems with worker assignment is presented.

In chapter three, we initially describe the studied problem in this thesis and present its mathematical programming formulation, and then give an illustrative example.

In chapter four, the proposed solution methodology is introduced. In this chapter, we firstly give information about the basic SA algorithm, and then describe the SA based solution approach in detail.

In chapter five, initially information about the generation of problem instances which are used for the computational study is given and then the parameter settings of the proposed SA algorithm, which are determined using the Taguchi method, are given. Finally, the computational results of the proposed IP model and the SA algorithm are presented and the comparative analysis of the obtained results for both approaches is given.

Finally, chapter six includes concluding remarks and several ideas for possible future research.

CHAPTER TWO
LITERATURE REVIEW

In this chapter, we first give information about assembly lines, assembly line balancing problems and then present literature review.

2.1 Assembly Lines

Assembly line is a mass production system in which while a product travels along the production line, transactions are made such as adding, combining in a series of workstations by workers, robots or machines. The finished product is obtained when all the tasks are completed in the workstations. Basic terms of assembly lines are as follows:

- *Task* is the smallest identifiable piece of the total assembly work and *processing time* is the time required to perform a task.
- *Workstation/ Station* is a location where tasks are performed on assembly line by workers, robots or machines.
- *Cycle time* is the total processing time needed to produce the finished product and besides the time which is required to complete the tasks in each workstation on the assembly line.
- *Workstation/Station time* is the sum of the processing times of tasks in a workstation.
- *Precedence relations* give the predecessor and successor tasks of each task. Due to the technological and/or organizational restrictions, tasks cannot be carried out in an arbitrary sequence. These precedence relations are generally shown with a precedence diagram. Figure 2.1 indicates a precedence relations diagram of an example with seven tasks. In the figure, nodes and node weights represent tasks and processing times of tasks, respectively.

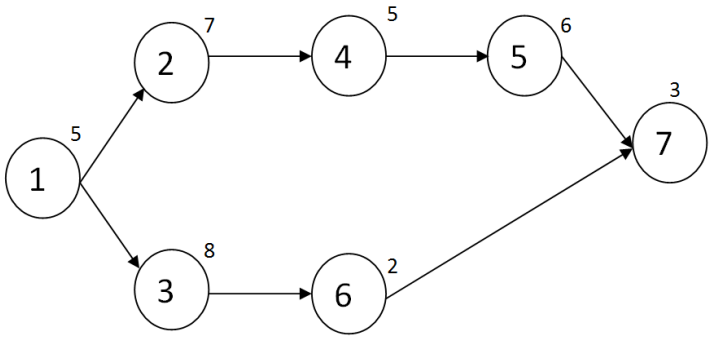


Figure 2.1 Precedence diagram

2.2 Assembly Line Balancing Problem

Assembly line balancing (ALB) problem deals with the assignment of tasks to workstations optimally under certain constraints. These constraints can be precedence relationships between the tasks, cycle time constraints and the number of workstations available.

According to Baybars (1986), deterministic ALB problems are divided into two groups. The first is simple assembly line balancing (SALB) problem and the second is general assembly line balancing (GALB) problem. The SALB problem is also divided into two types as SALB-1 and SALB-2 in terms of the objective function. Both problem types include the precedence constraints between the tasks and the objective function of SALB- 1 problem is to minimize the number of stations when cycle time is known and the objective function of SALB-2 problem is to minimize cycle time when the number of stations is known. The GALB problem is a more general version of the SALB problem. Scholl & Becker (2006) worked on the GALB problem, and in addition to previous restrictions, they dealt with a more flexible production system thanks to the conditions such as allowing opening parallel stations, indicating the mismatch between tasks (incompatibilities), modeling mixed-model production lines.

Depending on the number of models, the assembly lines are categorized into three groups: single (SMALB), mix (MMALB) and multi (MuMALB) model. These lines

are shown in Figure 2.2 (Becker & Scholl, 2006). The SMALB and MMALB problems contain production of one type of product and production of more than one type of product on the assembly line, respectively. Lastly, in MuMALB problem the production of different models is carried out in batches. When the model is changed, setup preparations are made.

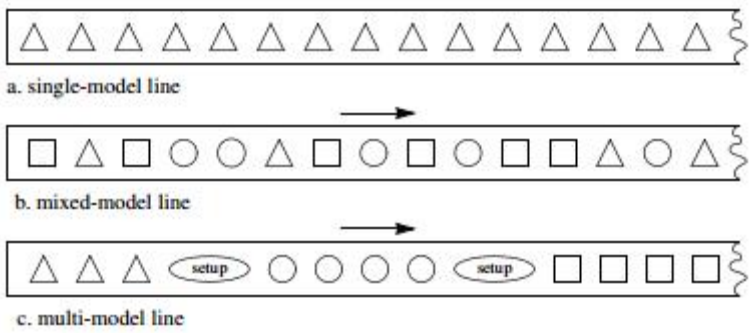


Figure 2.2 Assembly lines for single, mixed and multi model

With regard to the task duration, there are two types of ALB problems: deterministic and stochastic. In deterministic ALB problem, the processing times are known and there are very few differences between the processing times. In the other, the processing times of tasks differ by various reasons and these differences are expressed by probability distributions.

The ALB problem is also divided into seven groups: ALB-1, ALB-2, ALB-F, ALB-E, ALB-3, ALB-4 and ALB-5 in terms of the objective function (Scholl & Becker, 2006). ALB-1 and ALB-2 problems have been previously explained in this subsection. The objective function of ALB-F problem is to establish whether or not a feasible line balance exists when cycle time and number of stations are given. The objective function of ALB-E problem is to maximize line efficiency by minimizing the cycle time and the number of stations. The objective functions of ALB-3, ALB-4 and ALB-5 problems are to maximize the workload smoothness between the stations, the work relatedness and multiple objectives with ALB-3 and ALB-4, respectively. The most important classification for ALB problems is listed below in Figure 2.3.

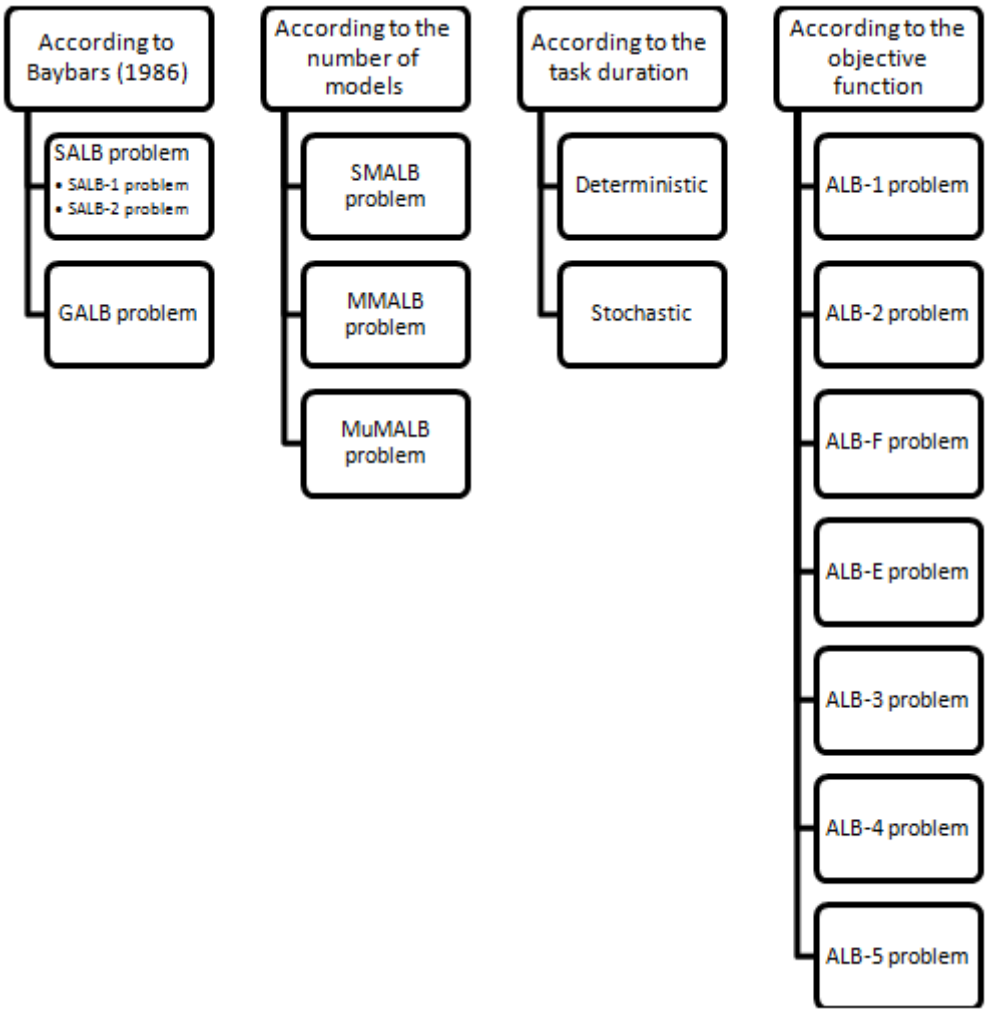


Figure 2.3 The classification of the ALB problems

In an effort to approximate ALB models to assembly lines in practice, real life constraints have been developed such as task assignment restrictions, positional constraints, parallel stations and zoning constraints. Firstly, task assignment restrictions enforce that some tasks must be assigned to the same station and some group of tasks must be performed alone in a station alone due to resource or technical restrictions (Boysen, Fliedner & Scholl, 2007). Secondly, positional constraints impose that the workpiece must be placed in a specific position in order to perform a task. According to Johnson (1983), some stations must be located on one (left) side of the line and other stations on the other (right) side of the line. Any specific task might be performed on only the left side, only the right side or possibly either side. Thirdly, parallel stations must be opened in order to increase productivity or to create

a more balanced line as well. Buxey (1974) presented the techniques of assembly line balancing with the parallel operation of identical stations for a reduction in idle time. Lastly, under the zoning constraint each task is performed in a specific zone or zones due to physical limitations of the production process (Mitchell, 1957; Tonge, 1960).

2.3 Assembly Line Worker Assignment and Balancing Problem

In traditional ALB problems, the processing times of tasks do not differ among workers. But in real life this situation is not like that. The processing times of a task performed by different workers are different due to the different skill or experience levels. Thus, a new decision as to who will perform the task should be given alongside the balancing problem. In the literature these ALB problems are referred to as assembly line worker assignment and balancing (ALWAB) problem. ALWAB problems are divided into seven groups according to objective functions, such as traditional ALB problems. These are ALWAB-1, -2,-3, -4, -5, -E and -F problems. The ALWAB problem has been first introduced by Miralles et al. (2007). The aim of the problem is to minimize the cycle time, given certain available workers in the Sheltered Work Centres for the Disabled.

Some characteristic assumptions for the ALWAB-2 problem are as stated below:

- Task processing times are deterministic and precedence relationships are known.
- Task processing time differs with respect to which one of the workers executes the task.
- The assembly line is a serial paced line with no buffers .
- A single model product is assembled on the assembly line.
- Workers can be very slow or even incapable when executing certain tasks, but very efficient when executing others.
- Each worker is assigned to only one workstation.
- Each task is assigned to only one workstation, on condition that the worker selected for that station is capable of performing the task.

According to these assumptions, the mathematical model can be expressed as follows:

Notation

i, j	task
h	worker
s	workstation
N	set of tasks
H	set of available workers
S	set of workstations
A	set of assignments a priori (i, h) task–worker
Z	set of assignments a priori (h, s) worker–station
C	cycle time
m	number of workstations
p_{hi}	processing time for task i when worker h executes it
D_j	set of tasks immediately preceding task j in the precedence network
x_{shi}	binary variable equal to 1 only if task i is assigned to worker h at station s
y_{sh}	binary variable equal to 1 only when worker h is assigned to station s

$$\min C \tag{2.1}$$

Subject to

$$\sum_{h\in H}\sum_{s\in S} x_{shi} = 1 \qquad \forall i \in N \tag{2.2}$$

$$\sum_{s\in S} y_{sh} \leq 1 \qquad \forall h \in H \tag{2.3}$$

$$\sum_{h\in H} y_{sh} \leq 1 \qquad \forall s \in S \tag{2.4}$$

$$\sum_{h\in H}\sum_{s\in S} s * x_{shi} \leq \sum_{h\in H}\sum_{s\in S} s * x_{shj} \qquad \forall i, j / i \in D_j \tag{2.5}$$

$$\sum_{i\in N} p_{hi} * x_{shi} \leq C \qquad \forall h \in H;\ \forall s \in S \tag{2.6}$$

$$\sum_{i \in N} x_{shi} \leq M * y_{sh} \qquad \forall h \in H;\ \forall s \in S \qquad (2.7)$$

$$\sum_{s \in S} x_{shi} = 1 \qquad \forall (i,h) \in A \qquad (2.8)$$

$$y_{sh} = 1 \qquad \forall (s,h) \in Z \qquad (2.9)$$

with:

$$M > \sum_{h \in H} \sum_{i \in N} p_{hi}$$

The objective function (2.1) of the model minimizes the cycle time. Constraint (2.2) provides that every task i is assigned to a single station s and worker h. Constraints (2.3) and (2.4) ensure that each worker can be assigned to at most one station, and that at every station there is at most one worker assigned. Constraint (2.5) satisfies the precedence relationships between tasks i and j, where i is predecessor of j. Constraint (2.6) provides that the total duration of tasks assigned should not surpass cycle time C when worker h is assigned to station s to perform the tasks. Here cycle time C is a variable that measures the longest total duration among the stations. Constraint (2.7) provides the relation between the variables x_{shi} and y_{sh}. If task i is assigned to worker h at station s, worker h should have been assigned to station s. Constraint (2.8) provides that the task-worker assignments must be considered a priori because of therapeutic or other specific reasons. Constraint (2.9) ensures that the station-worker assignments must be considered a priori due to some physical features of disabled workers.

2.4 Assembly Line Balancing Problem with Hierarchical Worker Assignment

Emmons & Burns (1991) for the first time developed a model that solves the hierarchical workforce scheduling problem. The hierarchical workforce consists of several worker types. The work force is classified into m types. The first worker type ($type - 1$) is the most highly qualified worker type and the last worker type ($type - m$) is the least qualified worker type. The model is developed in an attempt to find the smallest number and most economical mix of workers. In this problem, the main assumption is that the less qualified workers cost less. Studies have

continued on the hierarchical workforce scheduling problems. Hung & Emmons (1993) have dealt with the multiple-shift hierarchical workforce scheduling problem under a compressed workweek arrangement called the 3-4 workweek. Hung (1994) considered single-shift off-day scheduling of a hierarchical workforce and proposed a simple one-pass method that frequently gives the least cost labor mix. Billionnet (1999) developed an integer programming formulation to solve the problem studied by Hung (1994). An optimal algorithm for the single shift scheduling of hierarchical workforce has been introduced by Narasimhan (1997). Pastor & Corominas (2010) proposed a bi-criteria integer programming model for hierarchical optimization problem in which the first objective is to minimize the cost and the second is to maximize the suitability of task assignment.

But the hierarchical workforce on assembly line is first introduced by Sungur & Yavuz (2015). They developed a mathematical model for the assembly line balancing problem with hierarchical worker assignment (ALBHW). This mathematical model is as follows:

i, j tasks ($i, j \in T$)

h worker types ($h \in H$)

s stations ($s \in S$)

m upper bound on the number of stations

n number of tasks

l number of task/worker types

T set of tasks

S set of stations

K set of task types

H set of worker types

P_i set of immediate predecessors of task i

A set of precedence relations

G precedence graph

T_s set of tasks assigned to station s

h_s type of worker assigned to station s

k_i type of task i

c_h cost of type-h worker

t_{ih} time of task i when it is performed by type-h worker

Z cycle time

v_s 1, if station s is established; 0, otherwise ($\forall s \in S$)

x_{ihs} 1, if task i is assigned to station s equipped with type-h worker; 0, otherwise ($\forall i \in T, \forall h \in H | h \leq k_i, \forall s \in S$)

y_{hs} 1, if type-h worker is assigned to station s; 0, otherwise ($\forall h \in H, \forall s \in S$)

y_h total number of type-h workers assigned to the stations ($\forall h \in H$)

C total cost

$$\min C = \sum_h c_h * y_h \tag{2.1}$$

subject to

$$\sum_{h \leq k_i} \sum_s x_{ihs} = 1 \qquad \forall i \in T \tag{2.2}$$

$$\sum_h y_{hs} = v_s \qquad \forall s \in S \tag{2.3}$$

$$\sum_i x_{ihs} \leq n * y_{hs} \qquad \forall h \in H, \forall s \in S \tag{2.4}$$

$$\sum_s y_{hs} = y_h \qquad \forall h \in H \tag{2.5}$$

$$\sum_{h \leq k_j} \sum_s s * x_{jhs} \leq \sum_{h \leq k_i} \sum_s s * x_{ihs} \qquad \forall (j,i) \in A \tag{2.6}$$

$$\sum_i \sum_{h \leq k_i} t_{ih} * x_{ihs} \leq Z \qquad \forall s \in S \tag{2.7}$$

$$v_s \geq v_{s+1} \qquad \forall s \in S | s < m \tag{2.8}$$

$$v_s \in \{0,1\} \qquad \forall s \in S \tag{2.9}$$

$$x_{ihs} \in \{0,1\} \qquad \forall i \in T, \forall h \in H | h \leq k_i, \forall s \in S \tag{2.10}$$

$$y_{hs} \in \{0,1\} \qquad \forall h \in H, \forall s \in S \tag{2.11}$$

The objective function (2.1) minimizes the total cost. Constraint (2.2) provides that each task i is assigned to exactly one station equipped with one worker of type-h. Constraint (2.3) satisfies that a single worker is assigned to each station

established on the line. Constraint (2.4) defines the relation between x_{ihs} and y_{hs} variables: If type-h worker is not assigned to station s, i.e., $y_{hs} = 0$, then the values of all related x_{ihs} variables are forced to be zero as well. Constraint (2.5) determines the total number of type-h workers that are assigned to the stations. Constraint (2.6) ensures the precedence relations between the tasks. Constraint (2.7) prevents that the total task time of any station from exceeding the cycle time Z. Constraint (2.8) satisfies the consecutive establishing of stations. Constraints (2.9)–(2.11) define the binary restrictions on the variables.

2.5 Literature Review

Salveson (1955) published the first paper on ALB problem. After this pioneering work, the studies are continued in this area. As studies continue on the ALB problem, more flexible production systems have been developed thanks to the conditions such as allowing opening parallel stations, indicating the mismatch between tasks (incompatibilities), modeling mixed and multi model production lines.

Buxey (1974) presented the techniques of ALB for the parallel operation of identical stations for a reduction in idle time. Pinto, Dannenbring & Khumawala (1975) proposed a branch and bound algorithm with paralleling, in which a specific task is assigned to more than one station at cost of additional facilities. Bard (1989) presented an effective dynamic programming algorithm to solve the ALB problem in which both stations and tasks are paralleled. Furthermore this algorithm took into account the unproductive time during the solution. In another work, Askin & Zhou (1997) proposed a nonlinear integer programming model which allowed mixed-model production and the use of identical parallel workstations at each stage of the serial production system. A greedy heuristic approach was also developed to solve the problem.

The ALB problem for multiple objective problems is solved using simulated annealing (SA) by McMullen & Frazier (1998) when paralleling of workstations is permitted. This study showed how simulated annealing can be used to find improved solutions for balancing a production line where task times are stochastic, multiple

products are produced in a mixed-model fashion, parallel workstations are allowed, and cycle time performance as well as total labor and equipment cost are all important. Vilarinho & Simaria (2002) developed a mathematical model and a two stage procedure based on SA for the ALB problem with parallel workstations and zoning constraints. The first goal of the model is to minimize the number of workstations along the line and the second goal is to balance the workloads between and within workstations. The proposed procedure achieved good results even on large sized problems. Bukchin & Rubinovitz (2003) took into consideration the choice of equipment with parallel stations. They intended to minimize the total cost and the number of stations by a proposed integer linear programming model.

Akpınar & Bayhan (2011) developed a hybrid genetic algorithm for mixed model assembly line balancing problem of type I (MMALB-I) with parallel workstations and zoning constraints. The hybrid genetic algorithm consisted of integrating by sequentially Kilbridge & Wester Heuristic, Phase-I of Moodie & Young Method, and Ranked Positional Weight Technique with genetic algorithm. The three goals of the algorithm are respectively to minimize the number of workstations, maximize the workload smoothness between workstations, and maximize the workload smoothness within workstations. Çakır, Altiparmak & Dengiz (2011) proposed a new solution approach based on a SA algorithm, called m_SAA which performs a multinomial probability mass function approach, tabu list, repair algorithms and a diversification strategy for multi-objective optimization of a single-model stochastic ALB problem with parallel stations.

There can be some constraints related to the positioning of the product in the assembly line depending on the products. Essafi, Delorme, Dolgui & Guschinskaya (2010) introduced the machining line balancing problem with positional constraints for the flexible transfer lines composed of identical CNCs. The constraint is related to the positioning of the part. Lapierre & Ruiz (2004) developed a priority-based heuristic for ALB problem with two sides and two different heights in a case study. In the heuristic, some tasks must be performed only on a specific side of the conveyor because of product sizes. Tuncel & Topaloğlu (2013) formulated an integer

linear programming model for an electronics company, which assigns tasks to the stations taking into account the positional constraints of the workpiece when tasks are performed on the workpiece.

In addition to previous restrictions, ALWAB problem has emerged which is an extension of the ALB problem in a case study (Miralles et al., 2007). The aim of the problem is to minimize the cycle time, given certain available workers in the Sheltered Work Centres for Disabled. Chaves, Miralles & Lorena (2007) proposed a clustering search approach for solving the ALWAB-2 problem and tested on four problem families: Roszieg, Heskia, Wee-Mag and Tonge. Miralles, García-Sabater, Andrés & Cardós (2008) presented a basic branch and bound approach with three different search strategies for the ALWAB problem in a real case. These search strategies are depth first search with complete node development, best first search and minimal lower bound. Then a branch and bound-based heuristic was developed for large-sized problems. The results of the computational study revealed that the quality of the solution was at a reasonable level.

To solve the ALWAB-2 problem, Chaves, Lorena & Miralles (2009) developed a hybrid clustering search algorithm in that the iterated local search was integrated into the clustering search algorithm. Considering the solution quality and robustness of the method, the results of this algortihm were better than the clustering search approach developed by Chaves et al. (2007). Moreira & Costa (2009) developed a tabu search algorithm that tries to incorporate simplicity, flexibility, accuracy and speed for solving the ALWAB-2 problem. Blum & Miralles (2011) presented an iterative algorithm based on beam search to solve the ALWAB-2 problem and this study showed that this algorithm included better results than previous studies. Moreira, Ritt, Costa & Chaves (2012) developed a constructive heuristic approach based on task and worker assignment priority rules. This approach was hybridized into a biased random-key genetic algorithm. The results of this approach were found sufficient when other existing heuristics were taken into consideration. Araujo, Costa & Miralles (2012) introduced two new extensions of the ALWAB problem that allow parallelization and collaboration between different workers. Two mathematical

formulations and a constructive heuristic have been developed for the ALWAB problem.

An iterative genetic algorithm has been proposed with modified bisection search, genetic algorithm and iterated local search by Mutlu, Polat & Supçiller (2013) to solve ALWAB-2 problem. The results of this algorithm found reasonable in terms of robustness and efficiency of the method. Borba & Ritt (2014) developed an interval probabilistic beam search algorithm as well as a task-oriented branch-and-bound procedure. The results of the interval probabilistic beam search algorithm achieved better than a hybrid genetic algorithm and an iterated beam search (except the instance group of Wee-Mag). Vila & Pereira (2014) proposed a branch, bound and remember algorithm which is reported for two different time limits (60 seconds and 600 seconds) and no time limit. This implemented algorithm has shown better solution quality than other studies.

Moreira, Miralles & Costa (2015) introduced the Assembly Line Worker Integration and Balancing (ALWIB) Problem. This problem have arisen since the conventional and disabled workers both perform the tasks in same line. The aim of the problem is to minimize the number of additional workstations while the given disabled workers are integrated to the line. This problem is named ALWIB-1 problem. The authors developed a mathematical model and a constructive insertion heuristic for the ALWIB-1 problem. Thanks to this study, the losses of productivity are much lower than expected.

In ALWAB problem, processing times vary depending on the worker type but costs are not taken in consideration. Thus, Sungur & Yavuz (2015) introduced for the first time a formal definition and a mathematical model for the ALBHW problem. The authors developed an integer mathematical model. The aim of this model is finding the optimal assignment of workers and tasks to the stations such that total cost is minimized.

Ramezanian & Ezzatpanah (2015) developed an evolutionary algorithm called imperialist competitive algorithm (ICA) to solve mixed-model ALWAB problem. There are two objectives in the proposed model. The objectives are to minimize the total cycle time and the operating costs related to workers, respectively. Zacharia & Nearchou (2016) proposed a multi-objective evolutionary algorithm for the bi-objective ALWAB problem. The objectives of the problem are to minimize the cycle time and the smoothness index of the workload of the line.

To solve the ALWAB-2 problem, Polat, Kalayci, Mutlu & Gupta (2016) developed a two-phase variable neighbourhood search algorithm. A real case study from a consumer electronics company has been solved using the developed algorithm. Ritt, Costa & Miralles (2016) proposed an extension of the ALWAB problem for minimizing the expected cycle time under uncertain worker availability. A two-stage mixed integer model and local search heuristics have been proposed to solve the ALWAB problem with stochastic worker availability. As a result, the line's efficiency can be improved by mean of stochastic modeling. Akyol & Baykasoğlu (2016) developed a new type of ALWAB problem, called ErgoALWAB problem, which takes into account ergonomic risks. To solve the ErgoALWAB problem, a multiple-rule based constructive randomized search algorithm is proposed.

Moreira, Pastor, Costa & Miralles (2017) presented the multi-objective ALWIB problem of type-2. The problem aims to minimize the cycle time and maximize heterogeneous workers integration while maintaining productivity levels. The authors formulated the model and then proposed constructive heuristics. Pereira (2018) presented an interval data version of the ALWAB problem. The task times are known that lower and upper bounds, and the aim of the problem is to minimize the absolute maximum regret among all of the possible scenarios. The problem was solved with exact and heuristic solution methods.

The overview of approaches on ALWAB problem in the literature are presented in Table 2.1.

Table 2.1 An overview of approaches on ALWAB problem in the literature

Paper	Problem type	Solution method	Special restriction			
			Parallel stations	Assignment restriction	Positional constraint	Hierarchical workforce
Miralles et al. (2007)	ALWAB-2	Integer programming model				
Chaves et al. (2007)	ALWAB-2	Clustering search				
Miralles et al. (2008)	ALWAB-2	Branch and bound+ Branch and bound based heuristic				
Chaves et al. (2009)	ALWAB-2	Hybrid clustering search				
Moreira & Costa (2009)	ALWAB-2	Tabu search				
Blum & Miralles (2011)	ALWAB-2	Iterative beam search				
Moreira et al. (2012)	ALWAB-2	Hybrid genetic algorithm				
Araujo et al. (2012)	ALWAB-2	Mathematical model+ Constructive heuristic	√			
Mutlu et al. (2013)	ALWAB-2	Iterative genetic algorithm				
Borba & Ritt(2014)	ALWAB-2	Interval probabilistic beam search				
Vila & Pereira (2014)	ALWAB-2	Branch and bound with remember algorithm				
Moreira et al. (2015)	ALWIB-1	Mathematical model+ Constructive heuristic				
Sungur & Yavuz (2015)	ALBHW	Mathematical model				√
Ramezanian & Ezzatpanah (2015)	Mixed model ALWAB	Imperialist competitive algorithm				
Zacharia & Nearchou (2016)	Bi objective ALWAB	Multi-objective evolutionary algorithm				
Polat, et al. (2016)	ALWAB-2	Two-phase variable neighbourhood search algorithm				
Ritt et al. (2016)	ALWAB problem with stochastic worker availability	Two-stage mixed integer model + Local search heuristics				
Akyol & Baykasoğlu (2016)	ErgoALWAB	Multiple-rule based constructive randomized search algorithm				
Moreira et al. (2017)	ALWIB-2	Mathematical model+ Constructive heuristics				
Pereira (2018)	Interval data version of the ALWAB	Exact + Heuristic solution methods				
This thesis	ALBHW	Mathematical model + Simulated annealing algorithm	√	√	√	√

While positional constraints, parallel stations and task assignment restrictions are considered individually in previous studies in the literature, we have developed a mathematical model and a SA-based solution algorithm to solve the ALBHW problem that incorporates all these constraints.

CHAPTER THREE

PROBLEM DEFINITION

In this chapter, we present the problem definition, proposed mathematical formulation and an illustrative example.

3.1 Proposed Mathematical Formulation

In this section, the problem definition and the proposed mathematical formulation are presented for the ALBHW problem with positional constraints, task assignment restrictions and parallel stations.

The assumptions of the model are listed as follows:

- A single model product is produced on the assembly line.
- The precedence relations between the tasks are known.
- Task times are deterministic and vary according to the types of workers.
- The workpiece must be placed in a specific position in order to perform a task. (e.g. front and back tasks)
- Some groups of tasks must be assigned to the same station.
- Some of the tasks must be assigned to a station alone.
- Parallel stations are allowed in each stage of the production line.

In this problem, tasks require different levels of worker qualification and there is a hierarchical workforce structure in which a lower qualified worker can be substituted by higher qualified ones with a higher cost. The first worker type (type-1) is the most qualified worker type, whereas the last worker type (type-H, where H is the number of worker types) is the least qualified one. Accordingly, as the skill levels of workers increase, the processing times of tasks decrease. Parallel stations are allowed when the processing times of tasks exceed the cycle time and for better fit of task assignment. In addition to these, in real life assembly lines the workpieces must be in specific positions when tasks are being performed on the assembly line due to the technical, resource related etc. reasons. For example, during the assembly of a

refrigerator some tasks such as attaching a crisper are performed on the front of the workpiece, and some tasks such as attaching a condenser are performed on the back of the workpiece. These positional constraints enforce that only tasks which are performed in the same position can be assigned to the same station. The objective of this ALB problem is to find the optimal assignment of workers and tasks to stages, and to decide on the optimal number of parallel stations in each stage such that the total cost comprised of station paralleling costs and worker costs is minimized.

In the formulation, the following notation is used:

N	number of tasks, $i, j = 1, \dots, N$
M	upper bound on the number of stages that can be opened, $s = 1, \dots, M$
K	maximum number of parallel stations in each stage, $k = 1, \dots, K$
L	number of part positions required for performing the tasks, $l = 1, \dots, L$
H	number of worker types, $h = 1, \dots, H$
PT_l	set of tasks that can be performed when the part is situated in position l
TT_r	set of tasks that should be assigned together in a stage excluding all other tasks (it also includes sets with one task that should be assigned alone in a stage), $r = 1, \dots, R$
G	precedence graph
k_i	the least qualified worker type for task i
$Wcost_h$	cost of type-h worker
W_k	cost factor for paralleling k identical stations
T_{ih}	processing time of task i when it is performed by type-h worker
C	cycle time
UB	upper bound for the number of parallel stations in a stage
LB	lower bound for the number of parallel stations in a stage

Decision variables

x_{ihs}	1, if task *i* is assigned to stage *s* equipped with type-*h* worker; 0, otherwise
z_{skl}	1, if stage *s* is opened with *k* stations in parallel for the part position *l*; 0, otherwise
y_{hs}	1, if type-*h* worker is assigned to stage *s*; 0, otherwise
a_s	number of parallel stations in stage *s*
b_{hs}	total number of type-*h* workers in stage *s*

The proposed integer programming (IP) model can be formulated as follows:

Objective function

$$min \sum_{s}\sum_{h} Wcost_h . b_{hs} + \sum_{s}\sum_{k}\sum_{l} W_k . z_{skl} \tag{3.1}$$

Subject to

$$\sum_{h\le k_i}\sum_{s} x_{ihs} = 1 \qquad \forall i \tag{3.2}$$

$$\sum_{i} x_{ihs} \le N . y_{hs} \qquad \forall h, s \tag{3.3}$$

$$\sum_{k}\sum_{l} z_{skl} \le 1 \qquad \forall s \tag{3.4}$$

$$\sum_{h} y_{hs} = \sum_{k}\sum_{l} z_{skl} \qquad \forall s \tag{3.5}$$

$$\sum_{h\le k_j}\sum_{s} s . x_{jhs} \le \sum_{h\le k_i}\sum_{s} s . x_{ihs} \qquad \forall (j,i) \in G \tag{3.6}$$

$$\sum_{i}\sum_{h\le k_i} t_{ih} . x_{ihs} \le C . \sum_{k}\sum_{l} k . z_{skl} \qquad \forall s \tag{3.7}$$

$$\sum_{i\in PT_l}\sum_{h} x_{ihs} \le |PT_l| . (1 - \sum_{k}\sum_{t|t\ne l} z_{skt}) \qquad \forall s, l \tag{3.8}$$

$$\sum_{i \notin TT_r} \sum_{h} x_{ihs} \leq N.(|TT_r| - \sum_{i \in TT_r} \sum_{h} x_{ihs}) \quad \forall s, r \quad (3.9)$$

$$x_{ihs} = x_{jhs} = \cdots = x_{mhs} \quad \forall (i, j, \ldots, m) \in TT_r, \forall r, h, s \quad (3.10)$$

$$\sum_{k} \sum_{l} z_{skl} \geq \sum_{k} \sum_{l} z_{(s+1)kl} \quad \forall s \,|s < M \quad (3.11)$$

$$\sum_{k} \sum_{l} z_{skl}.k = a_s \quad \forall s \quad (3.12)$$

$$b_{hs} = y_{hs}.a_s \quad \forall s, h \quad (3.13)$$

$$b_{hs} \leq UB.y_{hs} \quad \forall s, h \quad (3.14)$$

$$b_{hs} \geq LB.y_{hs} \quad \forall s, h \quad (3.15)$$

$$b_{hs} \leq a_s - LB.(1 - y_{hs}) \quad \forall s, h \quad (3.16)$$

$$b_{hs} \geq a_s - UB.(1 - y_{hs}) \quad \forall s, h \quad (3.17)$$

$$x_{ihs} \in \{0,1\} \quad \forall i, s, h | h \leq k_i \quad (3.18)$$

$$z_{skl} \in \{0,1\} \quad \forall s, k, l \quad (3.19)$$

$$y_{hs} \in \{0,1\} \quad \forall h, s \quad (3.20)$$

The objective function (3.1) of the proposed model minimizes the labor and equipment cost. Constraint (3.2) ensures that each task is assigned to exactly one stage equipped with one worker of type-*h* qualified for the task. Constraint (3.3) provides the relation between x_{ihs} and y_{hs} variables; that is, if type-*h* worker is not assigned to stage *s*, then the values of all related x_{ihs} variables are forced to be equal to zero as well. Constraint (3.4) provides that if a station is opened in stage *s*, the number of parallel stations in this stage and the part positions allowed for these stations are unique. Constraint (3.5) ensures that a single worker type is assigned to each opened stage on the line. Constraint (3.6) guarantees the fulfillment of the precedence relationships between the tasks; that is, no task is assigned to an earlier station than its predecessors. Constraint (3.7) satisfies that the total workload time in a stage does not exceed the total capacity time in that stage, which corresponds to the number of parallel stations in this stage multiplied by the objective cycle time of the

line. Constraint (3.8) enables that only tasks performed in the same position of the workpiece can be assigned to the same stage. Constraints (3.9) and (3.10) deal with the assignment of tasks that must be performed together in a stage or in different stages, respectively. Constraint (3.11) ensures that stages are opened in ascending order. Constraint (3.12) calculates the number of parallel stations in stage *s*. Constraint (3.13) determines the total number of type-*h* workers in stage *s*. However, this constraint is in the quadratic form and it is linearized by Constraints (3.14)-(3.17). The linearization should consider two cases. The first case is when $y_{hs} = 0$, that is, type-*h* worker is not assigned to stage s. Constraints (3.14) and (3.15) guarantee that b_{hs} is forced to be zero if $y_{hs} = 0$. Constraints (3.16) and (3.17) also satisfy this relation. The second case is when $y_{hs} = 1$, and that b_{hs} should be equal to a_s. Constraints (3.16) and (3.17) imply that $a_s \leq b_{hs} \leq a_s$, forcing $b_{hs} = a_s$. Constraints (3.14) and (3.15) also satisfy this relation. Constraints (3.18)-(3.20) define the binary variables.

3.2 An Illustrative Example

An illustrative example of Jackson (1956) consisting of eleven tasks is presented and solved in this section. The precedence relationships diagram is given in Figure 3.1(a). Figure 3.1(b) shows the structure of the data for the tasks in the precedence relationships diagram. As seen in Figure 3.1 (a), the number of available worker types and the number of part positions required are three and two, respectively, i.e. $H = 3$ and $L = 2$. Accordingly, when task 3 is considered, $k_3 = 2$ means that task 3 can be performed by type-1 and type-2 workers. The processing times of task 3 when performed by type-1 and type-2 workers are five and six, respectively. The processing times taken from the original problem are assumed to be the processing times when performed by type-1 worker. The processing times when performed by type-2 and type-3 workers are arranged by Sungur & Yavuz (2015).

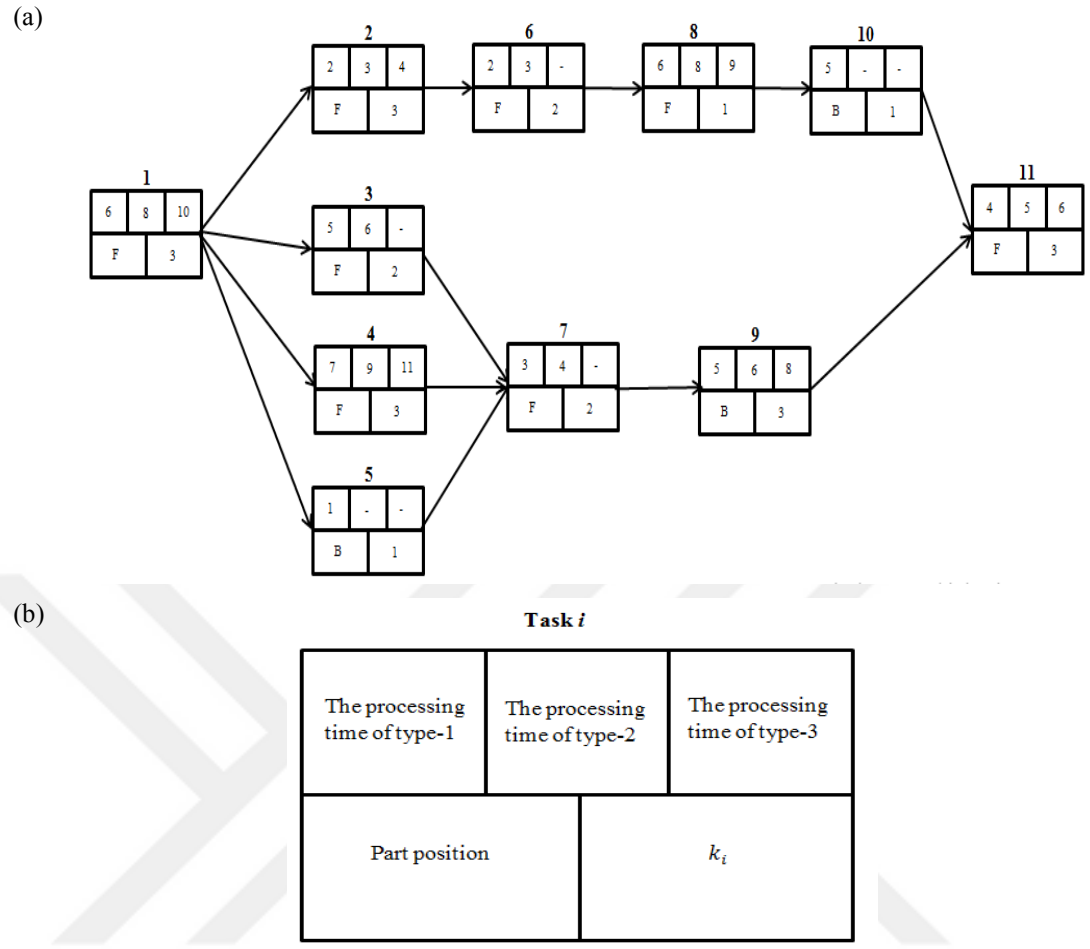


Figure 3.1 Precedence diagram and structure of tasks in the precedence diagram

The tasks indicated as F in Figure 3.1 (a) are performed when the workpiece is turned to the front side, whereas the tasks indicated as B are performed when the workpiece is turned to the backside. All the tasks assigned to a stage should consist of either all front or all back tasks.

The processing times of some tasks exceed the cycle time, which is six units. When this is the case, a parallel station should be opened. Also, parallel stations are allowed in order to provide better fit of task assignment,.

There are two task assignment restrictions in this illustrative example. The first of these is that tasks 6 and 7 must be assigned to the same station. The second one is that task 11 must be assigned to a station alone. The maximum number of parallel

stations in each stage is equal to three and the upper bound on the number of stages is equal to the total number of tasks, respectively, i.e. $K = 3$ and $M = N = 11$.

The station opening cost and worker cost are given in Table 3.1 and Table 3.2, respectively for this illustrative example. For example, if a stage contains two parallel stations with type-3 workers, the station and worker costs are 22 and 98 $(49 \times 2 = 98)$, respectively. The total cost is 120 $(22 + 98 = 120)$.

Table 3.1 Station opening cost

# of parallel station	Station cost (W_k)
1	10
2	22
3	35

Table 3.2 Worker cost

Worker type	Worker cost ($Wcost_h$)
1	100
2	70
3	49

The optimal task assignments obtained from the IP model are illustrated in Table 3.3. As seen from the table, the total cost is equal to 931. For example, tasks 2 and 4 are assigned to stage 3 with type-2 workers. This stage consists of two parallel stations.

Table 3.3 Optimal task assignment for the illustrative example

Stage	Parallel stations	Worker type	Tasks	Stage Time
1	1	1	1	6
2	1	1	5	1
3	2	2	2, 4	12
4	1	2	3	6
5	1	1	6, 7	5
6	1	1	8	6
7	1	1	10	5
8	1	2	9	6
9	1	3	11	6
Total Cost:	931			

CHAPTER FOUR
PROPOSED SOLUTION METHODOLOGY

In this chapter, the proposed solution methodology is introduced. Initially, we give information about simulated annealing (SA) algorithm. Subsequently, we describe the proposed solution approach based on the SA algorithm in detail. We give information about initial solution generation, objective function generation and neighborhood search structures (NSSs).

4.1 Simulated Annealing Algorithm

The SA is a well-known metaheuristic for solving the combinational optimization problems. Firstly, Metropolis, Rosenbluth, Rosenbluth, Teller & Teller (1953) proposed this algorithm which is similar to the annealing process. Then, Kirkpatrick, Gelatt & Veechi (1983) and Cerny (1985) applied the SA algorithm to solve NP-hard combinatorial optimization problems.

The acceptance mechanism of the SA algorithm is originated from the annealing process to avoid getting trapped in local optima. The SA algorithm starts with an initial solution. The initial solution becomes the current solution ($s_{current}$) and the best solution (s_{best}). The objective function value of the initial solution is set to the objective function value of the current solution ($f(s_{current})$) and the best solution ($f(s_{best})$). A control parameter called T is adjusted to an initial temperature (T_0). A neighboring solution (s) is created from $s_{current}$ by a specific NSS. If the objective function value of , $f(s)$, is better than $f(s_{current})$, $f(s_{current})$ and $s_{current}$ are updated to $f(s)$, and s, respectively, and this current solution is checked with s_{best}. If the objective function value of the current solution ($f(s_{current})$) is better than $f(s_{best})$, $f(s_{best})$ and s_{best} are updated to $f(s_{current})$, and $s_{current}$, respectively. On the other hand, if $f(s)$ is worse than $f(s_{current})$, then s might still change $s_{current}$ with the probability of Metropolis criterion, $EXP(-\Delta / T)$ (where $\Delta = f(s) - f(s_{current})$). T decreases by a cooling schedule rate until the stopping criteria is satisfied ($T > T_{final}$), where T_{final} is the final temperature. The number of

solutions searched in each temperature level is set to I. The SA algorithm is given in Figure 4.1.

Procedure $Simulated\_annealing$
Input: $T_0,\ T_{final},\ I, NSS,\ cooling\ schedule$
Output: $s_{best}, f(s_{best})$
$s_{current} = initialization$
$s_{best} = s_{current}$
$f(s_{best}) = f(s_{current})$
$T = T_0$
while $T > T_{final}$
  for $iter = 1: I$
    s =neighbor of $s_{current}$ created by a specific NSS
    if $f(s) < f(s_{current})$
      $f(s_{current}) = f(s)$
      $s_{current} = s$
      if $f(s_{current}) < f(s_{best})$ //check with the best solution
        $f(s_{best}) = f(s_{current})$
        $s_{best} = s_{current}$
      end if
    else
      if $random < (1/(exp(abs(f(s) - f(s_{current}))/T)))$
        $f(s_{current}) = f(s)$
        $s_{current} = s$
      end if
    end if
  end for
  T =temperature reduction by a $cooling\ schedule$
end while

Figure 4.1 Pseudocode of the SA algorithm

When the past literature that uses the SA algorithm for the solution of the ALB problem is concerned, some examples of these studies can be given as follows. Suresh & Sahu (1994) used the SA algorithm for solving the ALB problem with stochastic processing times. McMullen & Frazier (1998) solved the ALB problem for multiple objectives and when parallel workstations are permitted. Erel, Sabuncuoğlu & Aksu (2001) solved the ALB problem with U-type configuration. Vilarinho & Simiria (2002) proposed a SA approach for the SALB problem with parallel workstations in which the aim is to minimize the number of workstations and to smoothe the workload among the workstations. Baykasoğlu (2006) developed a multiple objective SA algorithm for solving the simple and U-type ALB problems. The goal of these problems is to maximize smoothness index and line performance. Çakır et al. (2011) proposed a solution approach based on the SA algorithm which

employs a multinomial probability mass function approach, tabu list, repair algorithms and a diversification strategy for multi-objective optimization of the single-model stochastic ALB problem with parallel stations. Roshani & Giglio (2016) developed a mixed-integer programming model and two SA approaches to solve the multi-manned ALB problem. The goal of the problem is to minimize the cycle time. Kellegöz (2016) developed a new mathematical formulation and Gantt based SA method for the ALB problems with multi-manned stations for minimizing the number of workers used in the line and the number of stations opened in the line.

4.2 Proposed Simulated Annealing Algorithm

In the proposed SA algorithm which is shown in Figure 4.9, the original SA algorithm shown in Figure 4.1 is improved by giant leap and local search mechanisms for the related problem. The details of these parts will be given in the next sections. The terms in the proposed SA algorithm are given in Table 4.1.

Table 4.1 The terms in the proposed SA algorithm

T_0	initial temperature	$nbtasks$	number of tasks
T_{final}	final temperature	$wcost_h$	cost of type- h worker
T	current temperature	$pcost_k$	cost of paralleling k identical stations
I	the number of neighborhood searches in each temperature	$Precedence$	precedence relations between tasks
s_{best}	the best solution during the search	wp_i	part position of task i
$f(s_{best})$	objective function value of s_{best}	$processingtime_{i,h}$	processing time of task i with type-h worker
$s_{current}$	current solution	C	cycle time
$f(s_{current})$	objective function value of $s_{current}$	$togethertasks$	sets of tasks that should be assigned together in a stage
nss_list	NSS queue	$alonetask$	set of tasks that should be assigned alone in a stage
i	i controls the change of NSS	pw_i	positional weight of task i
nss_i	NSS i in the nss_list	$task_i$	task in the i^{th} rank when all tasks are sorted in the descending order of their positional weights
nss_count	number of consecutive temperatures passed with no improvements	nb_jump_task	number of skipped tasks in the positional weight ordering list before $task_i$, which could not be assigned in the earlier attempts
$struct\_current$	current NSS		

4.2.1 Solution Representation

A solution representation is given in Figure 4.2 for an illustrative example based on Jackson (1956) problem with eleven tasks and positional constraints, which is presented in Section 3.2. A solution is represented as a matrix with dimension $nbtasksx5$, where $nbtasks$ is the number of tasks. The solution matrix consists of five columns. The first column indicates the tasks, the second column the stages, the third column the number of parallel stations, the fourth column the worker type assigned, and the last column the position of the workpiece (e.g. if the position of the workpieces when assembly tasks are performed on their front sides, the value in the fifth column is one. Otherwise, this value is two). For example, tasks 2 and 4 are

assigned to stage 3 with type-2 worker, and this stage consists of two parallel stations. The full explanation of this solution is shown in Table 4.2.

1	1	1	1	1
5	2	1	1	2
2	3	2	2	1
4	3	2	2	1
3	4	1	2	1
6	5	1	1	1
7	5	1	1	1
8	6	1	1	1
10	7	1	1	2
9	8	1	2	2
11	9	1	3	1

Figure 4.2 Solution representation

Table 4.2 Explanation of solution representation

Stage	Parallel stations	Worker type	Tasks	Position of the workpiece
1	1	1	1	Front side
2	1	1	5	Back side
3	2	2	2, 4	Front side
4	1	2	3	Front side
5	1	1	6, 7	Front side
6	1	1	8	Front side
7	1	1	10	Back side
8	1	2	9	Back side
9	1	3	11	Front side

4.2.2 Initial Solution Generation

The initial solution significantly affects the result of the algorithm. Therefore, we use a modified version of the Rank Positional Weight heuristic (Helgeson & Birnie, 1961) to generate an initial solution in the proposed SA algorithm. Firstly, the positional weight of a task is determined by summing its processing time plus the processing times of tasks connected with itself and its successors and the tasks are listed by the descending order of their positional weight. These processing times belong to the type-1 worker.

- The task with the largest positional weight remaining tasks among the remaining tasks waiting to be assigned ($task_i$) is selected first.

- If the selected task belongs to one of the $togethertasks$ sets, the predecessors of all the tasks belonging to this set are checked whether they have been assigned. If these predecessors are already assigned to a stage, the tasks in this set are assigned to a new stage all together.
- If the selected task belongs to $alonetask$ set, the predecessors of the task are checked whether they have been assigned. If these predecessors are already assigned to a stage, this task is assigned to a new stage by itself.
- If the selected task does not belong to one of the $togethertasks$ sets nor it belongs to $alonetask$ set, the predecessors of the task are checked whether they have been assigned. If these predecessors are already assigned to a stage, the this task is tried to be assigned to one of the opened stages starting from the first one.
    - If the part position of the selected task (wp_{task_i}) is the same as the part position of the tasks on the recent stage and if the tasks on this stage do not belong to one of the $togethertasks$ sets and $alone\ task$ set, the predecessors of the selected task are checked whether they have been assigned to earlier stages.
    - If precedence relations is not feasible, it is passed to the next stage. Otherwise, the sum of the processing times of all the tasks in the stage with the type-1 workers ($total\ processing\ time$) is calculated.
    - If $total\ processing\ time$ considering type-1 worker exceeds the cycle time multiplied by K, where K is upper bound for the number of parallel stations in a stage, it is passed to the next stage. Otherwise, the selected task is assigned to the stage this.
    - If the current task cannot be assigned to any previously opened stage, then it is assigned to a new stage.

If the current task ($task_i$) is assigned to a stage and there is the skipped task in the positional weight ordering list before current task, which could not be assigned in the earlier attempts ($nb\_jump\_task \neq 0$), the next selected task to be assigned is the first skipped task in the positional weight ordering list before current task. Otherwise,

the selected task to be assigned is the next task in the list. The assignments continue in this way. A detailed pseudocode of the initial solution generation is given in Figure 4.3.

Procedure *initial_solution_generation*
Input: $processingtime_{i,h}, task_i, Precedence, \mathrm{C}, wp_i, nbtasks, alonetask, togethertasks$
Output: *initial solution*
$i = 1$ // i provides the row control of the $task_i$
while $i <= nbtasks$
  if $task_i \in togethertasks$
    check the assignment status for the predecessors of $task_i$
    if the predecessors have been assigned
      assign all the tasks in this $togethertasks$ set to new stage
    end if
  elseif $task_i \in alonetask$
    check the assignment status for the predecessors of $task_i$
    if the predecessors have been assigned
      assign $task_i$ to new stage
    end if
  else
    check the assignment status for the predecessors of $task_i$
    if the predecessors have been assigned
      for j = 1 to Y //Y is the number of opened stages
        if $(wp_{task_i} = wp_{\text{any task on stage } j})$ ***and*** (any task on stage j) $\notin togethertasks$ ***and*** (any task on stage j) $\notin alonetask$
          check the precedence relation status when $task_i$ is assigned to the stage on the j^{th} row
            if the precedence relation is not feasible
              continue
            else
              calculate the sum of the processing times of all the tasks in the stage with the type-1 workers // calculate the $total\ processing\ time$
                if $total\ processing\ time > (C \times K)$
                  continue
                else
                  assign $task_i$ with other tasks to stage j
                  break
                end if
            end if
        end if
      end for
      if $task_i$ has not been assigned to previously opened stages
        assign $task_i$ to new stage
      end if
    end if
  end if
calculate the number of skipped tasks **(**nb_jump_task)
  if $task_i$ is assigned to a stage ***and*** $nb\_jump\_task \neq 0$
    $i = d$ // d is location of the first skipped task in the $task_i$
  else
    $i = i + 1$
  end if
end while

Figure 4.3 Pseudocode of the initial solution generation

4.2.3 Parallel Stations and Worker Type Assignments

When a task is being assigned to a new stage or an existing stage, or a task is removed from an existing stage, the decision on the number of parallel stations opened and the worker type assigned to the stage is determined as follows.

Firstly, all possible total cost values, consisting of station paralleling and worker costs, are calculated considering all combinations of the number of parallel stations and worker type assigned. For example, let us assume that the number of worker types and the maximum number of parallel stations allowed in each stage are both two, i.e. $H = 2$ and $K = 2$, and station paralleling cost and worker cost for each type are given as in Tables 4.3 and 4.4, respectively. Accordingly, total cost values are calculated for all combinations as illustrated in Table 4.5. Then, these costs are sorted in the ascending order. If the task is assigned to a new stage, starting from the lowest total cost configuration, the associated number of parallel stations and type of worker are tried for this stage. If the processing time of the assigned task when performed by the corresponding worker type does not exceed the cycle time multiplied by the number of parallel stations, this configuration is accepted. Otherwise, the next lowest total cost configuration is tried. This process continues until a feasible configuration is found. If the task is assigned to an already existing stage provided that this assignment is feasible, the sum of processing times of all existing tasks in this stage and the assigned task should be controlled whether it exceeds the cycle time multiplied by the number of parallel stations.

It is also required to find the least total cost configuration when a task is removed from a stage. The possible configurations are tried in the same order and the first feasible one satisfying the time limit is identified as the lowest cost choice.

Table 4.3 Station opening costs

# of parallel stations (k)	Station cost (W_k)
1	10
2	22

Table 4.4 Worker costs

Worker type (h)	Worker cost ($\mathrm{Wcost_h}$)
1	100
2	70

Table 4.5 Total cost

# of parallel station (k)	Worker type (h)	Total cost ($totalcost_{k,h}$)
1	1	$10 + 1 * 100 = 110$
2	1	$22 + 2 * 100 = 222$
1	2	$10 + 1 * 70 = 80$
2	2	$22 + 2 * 70 = 162$

4.2.4 Objective Function Calculation

The objective function of the proposed model is to minimize total cost which consists of the worker and station costs. The station opening cost depends on the number of parallel stations in each stage, whereas the worker cost depends on the worker type and the number of workers assigned for each stage considering the number of parallel stations opened. The pseudocode of this calculation is shown Figure 4.4.

Procedure Obj_cal
Input: $nbtasks, parallelcost, wcost$
Output: obj
$obj = 0$ // obj = objective function value
for $m = 1: nbstages$ // $nbstages$ = number of stages
find the number of parallel workstations *(k)* and the worker type *(h)* for the m^{th} stage
$obj = obj + (pcost(k) + k * wcost(h))$
end for

Figure 4.4 Pseudocode for the objective function calculation

This calculation is used after the initial solution generation, the neighborhood search structures, the local search and the giant leap.

4.2.5 Neighborhood Search Structures

To produce new different solutions from the current solution, four kinds of NSSs are used: transfer, two_transfer, swap and stage separation.

4.2.5.1 Single_Transfer

In the first NSS, *single_transfer*, a randomly selected task is transferred to a randomly selected stage. In this structure, the points to consider are as follows:

- The randomly selected task cannot belong to the sets of $togethertasks$ or $alonetask$.
- The randomly selected task cannot be assigned to the stages of tasks belonging to $togethertasks$ or $alonetask$ sets.
- The part position of the selected task should be the same as the part position of the tasks in the transferred stage.
- The precedence relations and cycle time constraints should be satisfied.

An example of single_transfer NSS is shown in Table 4.6. Let us assume that the randomly selected task and stage are 4 and 2, respectively. Thus, task 4 is transferred to stage 2 considering that the above requirements are satisfied. After the transfer, the lowest total cost configuration is identified for stage 1 with tasks 1 and 2, and for stage 2 with tasks 3 and 4, in their renewed form as described in 4.2.3. Before the transfer, the worker type and the number of parallel stations were respectively one and three for stage 1 . After the single_transfer NSS, both the worker type and the number of parallel stations became two. On the other hand, for stage 2 the worker type and the number of parallel stations became one and two, respectively.

Table 4.6 The *single_transfer* neighborhood search structure

Stage 1				**Stage1**		
tasks	# of parallel stations	worker type		tasks	# of parallel stations	worker type
1-2-**4**	3	1	**Single_Transfer**	1-2	2	2
Stage 2				**Stage 2**		
tasks	# of parallel stations	worker type		tasks	# of parallel stations	worker type
3	1	2		3-**4**	2	1

4.2.5.2 Two_Transfer

In the second NSS, *two_transfer*, two randomly selected tasks are transferred to randomly selected stages. In this structure, the points to consider are as follows:

- The randomly selected tasks cannot belong to the sets of $togethertasks$ or $alonetask$.
- The randomly selected tasks cannot be assigned to the stages of tasks belonging to $togethertasks$ or $alonetask$ sets.
- The part positions of the selected tasks should be the same as the part positions of the tasks in the transferred stages.
- The precedence relations and cycle time constraints should be satisfied.

An example of two_transfer NSS is shown in Table 4.7. Let us assume that the randomly selected tasks are 2 and 4, respectively, The randomly selected stages for the selected tasks by chance are the same, and these stages are 2. Thus, tasks 2 and 4 are transferred to stage 2 considering that the above requirements are satisfied. After the transfer, the lowest total cost configuration is identified for stage 1 with task 1, and for stage 2 with tasks 2, 3 and 4 in their renewed form, as described in 4.2.3. Before the transfer, the worker type and the number of parallel stations were respectively one and three for stage 1. After the two_transfer NSS, both the worker type and the number of parallel stations became one. On the other hand, for stage 2 the worker type and the number of parallel stations became two and three, respectively.

Table 4.7 The *two_transfer* neighborhood search structure

Stage 1				**Stage1**		
tasks	# of parallel stations	worker type		tasks	# of parallel stations	worker type
1-**2**-**4**	3	1	**Two_Transfer**	1	1	1
Stage 2				**Stage 2**		
tasks	# of parallel stations	worker type		tasks	# of parallel stations	worker type
3	1	2		**2**-3-**4**	3	2

4.2.5.3 Swap

In the third NSS, *swap*, the stages of two randomly selected tasks are exchanged. In this movement, the points to consider are as follows:

- The randomly selected tasks cannot belong to the sets of $togethertasks$ and $alonetask$.
- The part positions of the selected tasks should be the same.
- The precedence relations and cycle time constraints should be satisfied.

An example of swap NSS is shown in Table 4.8. Let us assume that the randomly selected tasks are 3 and 4, respectively, and task 3 is exchanged with task 4. Thus, tasks 3 and 4 are transferred respectively to stage 1 and stage 2 considering that the above requirements are satisfied. After the transfer, the lowest total cost configurations are identified for both the stages as in the previous NSSs.

Table 4.8 The *swap* neighborhood search structure

Stage 1				**Stage1**		
tasks	# of parallel stations	worker type		tasks	# of parallel stations	worker type
1-2-**4**	3	1	**Swap**	1-2-**3**	3	2
Stage 2				**Stage 2**		
tasks	# of parallel stations	worker type		tasks	# of parallel stations	worker type
3	1	2		**4**	2	3

4.2.5.4 Stage_Separation

In the fourth NSS, *stage_separation*, the randomly selected stage (a) is divided to two stages, and the decision of how existing tasks are allocated to these stages is determined by a randomly selected $divider$ which takes value between 1 and $k-1$ (where k is the number of tasks in the selected stage). Accordingly, the first divided stage contains $divider$ number of tasks, and the second stage contains the remaining number of tasks. In this NSS, the points to consider are as follows:

- The number of tasks in the selected stage should be greater than one ($k > 1$) and the tasks in the selected stage cannot belong to the sets of $togethertasks$ or $alonetask$.
- The precedence relations must be satisfied.

Two examples of stage_separation NSS are given in Figure 4.5. In Figure 4.5 (a), the randomly selected stage is 6, and it contains only two tasks. For this reason, the $divider$ can only take value one, and thus it splits stage 6 into two stages 6 and 7 containing task 9 and task 10, respectively. In Figure 4.5 (b), the randomly selected stage 6 contains three tasks. Thus, the value of the $divider$ can be either one or two. When the $divider$ is selected as two, tasks 9 and 10 are assigned to stage 6, whereas task 11 is assigned to newly opened stage 7. In addition, to increase the variability of solutions that can be obtained, tasks can be randomly allocated between the split stages.

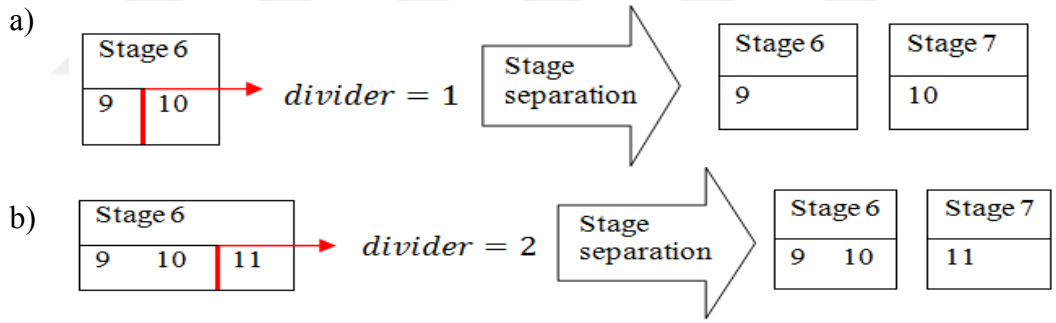


Figure 4.5 Examples of the stage_separation neighborhood search structure

When the stage_seperation NSS is applied to stage 6 in Figure 4.5 (a), the lowest total cost configurations for the split stages are shown in Table 4.9.

Table 4.9 The stage separation neighborhood search structure

Stage 6		
tasks	# of parallel stations	worker type
9-10	2	1

Stage separation

Stage 6		
tasks	# of parallel stations	worker type
9	1	2
Stage 7		
tasks	# of parallel stations	worker type
10	1	1

4.2.6 Neighborhood Search Structure Change

Variable neighborhood search (VNS), proposed by Mladenović & Hansen (1997), is a metaheuristic that systematically changes neighborhoods during the search to get rid of the local optimum. In order to escape from the risk of encountering a premature convergence to local optima and to navigate unvisited solution spaces to get higher-quality solutions, we used the concept of VNS within our proposed SA algorithm as discussed in Seyed-Alagheband, Ghomi & Zandieh (2011), and in this concept if one neighborhood search structure cannot provide better solutions after some algorithm iterations, another one is used.

In the proposed SA approach, if the best solution is not improved (i.e. $impv = 0$) after five consecutive temperatures using the current NSS ($struct\_current$),or the current NSS ($struct\_current$) cannot produce a new solution due to there are too many constraints in our model, we refer to this situation as a blockage ($\text{block} = 1$), and when it happens, the next NSS in the NSS queue (nss_list) is used. For instance, '$transfer - swap - stage\_separator$' is a NSS queue. The proposed algorithm starts from the transfer NSS and searches the neighborhood of the current solution until it is unable to continue the above situations are provided. Then, the next NSS in the queue, in this example swap, is used. Through this process, if the algorithm reaches the end of the queue, it will restart from the first NSS ($\text{nss}_1 = \text{transfer}$) in the queue. It should be also noted that if best solution is not improved after five consecutive temperatures, the temperature is adjusted to the temperature which was observed five consecutive temperatures before. In addition to this case, if there is a blockage in the current NSS ($struct\_current$), the temperature is adjusted

to the previous temperature before blockage. If best solution is improved i.e. $impv = 1$, the $struct\_current$ is not changed. The proposed c_neigh procedure is presented in Figure 4.6.

Procedure c_neigh
Input: $T, nss_{count}, i, nss_{list}, struct_{current}, impv, block$
Output: $T, struct\_current$
if $impv = 0$ (i.e. best solution is not improved) ***and*** $block = 0$ (i.e. there is not a blockage in $struct\_current$)
  $nss\_count = nss\_count + 1$ // number of consecutive temperatures passed with no improvements
  if $nss\_count = 5$ // best solution is not improved after five consecutive temperatures
    go back to the temperature that was observed five consecutive temperatures before
    $i = i + 1$ //i controls the change of NSS
    $struct\_current = nss_i$ //skip to next NSS
    $nsscount = 0$
  end if
elseif $impv = 1$ i.e. best solution is improved
  $nss\_count = 0$
elseif $block = 1$ //there is a blockage in $struct\_current$
  go back to previous temperature before blockage
  $i = i + 1$
  $struct\_current = nss_i$ //skip to next NSS
  $nss\_count = 0$
end if
if all of NSSs are used
  $i = 0$
end if

Figure 4.6 Pseudocode for NSS change

4.2.7 Local Search

The proposed SA is also integrated with a local search and the local search procedure works as follows: Initially, the task with the lowest index in the first stage, which cannot belong to the sets of $togethertasks$ and $alonetask$, in the current solution ($s_{current}$) representation is picked to be transferred to a stage randomly. If it cannot be assigned to the randomly selected stage because of the constraints, a feasible stage is searched by randomly. If it cannot be assigned to any stage, then the next task is picked. If the new solution x results in a better objective function value, the current solution is updated to the new solution x. This procedure repeats at most for all the succeeding tasks in the current solution. This local search is terminated at the first improvement. Finally, the objective function of this new solution x is checked with the objective function of the best solution at hand. The best solution is replaced with the new solution if it has a lower objective function value. The pseudocode of the local search is given in Figure 4.7.

```
Procedure local_search
Input: f(s_current), s_current, f(s_best), s_best, nbtasks, alonetask, togethertasks
Output: f(s_current), s_current, f(s_best), s_best
k == 1
for k = 1:nbtasks
  if k ∈ togethertasks or k ∈ alonetask
    continue //continue with the next iteration in the loop
  else
  generate new solution x by transferring task k to a randomly selected stage which is satisfied
the constraints, in the s_current
    if task k cannot be assigned to any stage because of the constraints
      continue
    else
      Z = obj_calc(x)
      if Z < f(s_current) // check with the current solution
        s_current = x
        f(s_current) = Z
        if Z < f(s_best)// check with the best solution
          s_best = x
          f(s_best) = Z
        end if
        break
      end if
    end if
  end if
end for
```

Figure 4.7 Pseudocode for the local search of the proposed algorithm

4.2.8 Giant Leap

Despite the proposed NSSs, there is still a considerable chance that the solution may get trapped in local optima. A novel mechanism, called migration mechanism, and a new operator, called giant leap (GL), to avoid getting trapped in local optima is first proposed by Naderi, Zandieh & Roshanaei (2009). Then, GL is used by Seyed-Alagheband et al. (2011) on the assembly line type II problem with sequence-dependent setup times between tasks to escape from the local optima. GL is able to leave the current search space and head towards an inferior or unvisited search space for the probability of finding a better solution. We use GL immediately after the local search in the proposed SA algorithm. The procedure is as follows: the GL operator produces 80 neighbor solutions from the current solution ($s_{current}$) around the solution space by using *two_transfer* NSS. The neighbor solutions are generated in pairs successively, for example, 80 neighbor solutions are produced in forty pairs, in this case the total number of pairs (k) and the total number of neighbor solution ($k \times 2$) are respectively 40 and 80. Every pair produced is compared to each other within

itself and the best neighbor solution in the pair is kept. Then the best neighbor solution (c) is chosen from these kept neighbor solutions (b_i). The current solution ($s_{current}$) is updated to the solution which is the best of all neighbor solutions. Finally, this new current solution is checked with the best solution (s_{best}). If it is better than the best solution, the best solution is updated. The proposed GL procedure is presented in Figure 4.8.

Procedure $giant\_leap$
Input: $f(s_{current}), s_{current}$, $f(s_{best}), s_{best}$
Output: $f(s_{current}), s_{current}$, $f(s_{best}), s_{best}$
for $i = 1{:}\,k$ // k is number of pairs, and $k \times 2$ is equal to number of giant leap operator
  for $j = 1{:}\,2$
    $a_j = two\_transfer(s_{current})$
  end for
  $b_i =$ the best a_j // the best neighbor solution in the i^{th} pair
end for
$c =$ the best b_i// c is the best neighbor at the end of giant leap
update the $s_{current}$ and $f(s_{current})$ according to c
if $f(s_{current}) < f(s_{best})$// check with the best solution
  $f(s_{best}) = f(s_{current})$
  $s_{best} = s_{current}$
end if

Figure 4.8 Pseudocode for GL

4.2.9 Cooling Schedule

The cooling schedule controls the behavior of SAs thanks to temperature grade and cooling rate. To avoid local minimum, worse solution can be accepted depending on the temperature. The temperature is gradually reduced under the cooling schedule. Two types of cooling schedule, linear and exponential, are used in the proposed SA. These cooling schedule rates are given in Table 4.10 where T_0, T_f and N are initial temperature, final temperature, and the desired number of temperatures between T_0 and T_f, respectively. These parameters are tuned in the next subsection. The final temperature is fixed at 1. The pseudocode of the proposed SA algorithm is shown in Figure 4.9.

Table 4.10 Cooling schedule rates

Cooling schedule	Formula		
Linear	$T_i = T_0 - i * \frac{T_0 - T_f}{N}$		$i = 1, \dots, N$
Exponential	$T_i = \frac{A}{i+1} + B$	$A = \frac{(T_0 - T_f)(N+1)}{N}$ $B = T_0 - A$	$i = 1, \dots, N$

Procedure $Simulated\_annealing$
Input: T_0, T_{final}, I, nss_list
Output: $s_{best}, f(s_{best})$
$s_{current} = initial\ solution$ in $initial\_solution\_generation$
$s_{best} = s_{current}$
$f(s_{best}) = f(s_{current})$
$T = T_0$
$nss\_count = 0$
$nss\_list = [single\_transfer, swap, stage\_separation]$ //NSS queue
$i = 1$
$struct\_current = nss_i$
while $T > T_{final}$
  $imprv = 0$
    for $iter = 1{:}I$
      s =neighbor of $s_{current}$ created by $struct\_current$
      if $block = 1$
        break
      end if
      if $f(s) < f(s_{current})$
        $f(s_{current}) = f(s)$
        $s_{current} = s$
          if $f(s_{current}) < f(s_{best})$ //check with the best solution
            $f(s_{best}) = f(s_{current})$
            $s_{best} = s_{current}$
            $impv = 1$
          end if
      else
        if $random < (1/(exp(abs(f(s) - f(s_{current}))/T)))$
          $f(s_{current}) = f(s)$
          $s_{current} = s$
        end if
      end if
    end for
    T =temperature reduction by a $cooling\ schedule$
    check the update NSS situation with c_neigh
     update $f(s_{current})$ and $f(s_{best})$ with $local\_search$
    update $f(s_{current})$ and $f(s_{best})$with $giant\_leap$
end while

Figure 4.9 Pseudocode of the proposed SA algorithm

CHAPTER FIVE
COMPUTATIONAL STUDY

In this chapter, we firstly give information about the generation of problem instances and SA parameters tuning. After that, the computational results of the proposed IP model and SA algorithm are presented and compared with each other.

5.1 Problem Instances Generation

The proposed IP model and the SA algorithm have been applied to solve 60 test problems which are the SALB-1 problem instances proposed by Otto, Otto & Scholl, 2013. The instances have been modified to deal with the specific situations of our problem such as varying processing times of tasks according to the qualifications of workers and the part positions while the tasks are performed. Their data set is characterized by the following parameters: number of tasks (small, medium, large, and very large), order strength (0.20, 0.60, and 0.90), distribution of task times (PB: peak at the bottom, PM: peak in the middle, and BM: bimodal), and type of the precedence graph (bottleneck, chain, and mixed). The specifications of our test problems are accordingly arranged as follows:

The small (*S*), medium (*M*) and large (*L*) data sets which include 20, 50 and 100 tasks, respectively, are used. The order strength (OS) of a data set is measured according to the number of precedence relations. Mastor (1970) defines the OS as the total number of ordering relations divided by $P = n(n-1)/2$; where $P =$ possible number of ordering relations, and $n =$ number of tasks. We have used two OS levels which are low and high, i.e., low OS=0.20, and high OS=0.9. The low and high order strengths have been selected to more clearly observe the effect of the order strengths on the results.

Otto et al. (2013) generate task times according to three types of probability distributions: normal with peak at small tasks (peak at the bottom, or PB), bimodal (or BM), which is a combination of two normal distributions with a higher peak at

small tasks, and normal with a peak at 0.5 x Cycle Time (peak in the middle, or PM). These distributions are asserted to be closed to those typically found in manufacturing.

Precedence graphs are grouped into three categories according to the presence of chain and bottleneck tasks in them (Otto et al., 2013). Instances of the first category, CH, contains at least 40% chain tasks. The second category, BN, consists of instances containing bottleneck tasks with the least degree of eight except for small instances with a minimum degree of four. Finally, the category MIXED contains graphs which do not control for the share of chain or bottleneck tasks. We use those instances that belong to the MIXED category.

The hardness of a problem instance is measured by the trickiness measure (*Tr)*. It indicates the difficulty of reaching an optimal solution, and it is defined as a statistically estimated share of non-optimal solutions in the solution space. *Tr* is divided into five categories as follows:

- extremely tricky (*ET*) ($Tr \geq 0.995$)
- very tricky (*VT*) ($Tr \in [0.95, 0.995)$
- tricky (*TR*) ($Tr \in [0.5, 0.95)$)
- less tricky (*LT*) ($Tr \in (0, 0.5)$)
- trivial, i.e., all the randomly generated solutions are optimal

In this problem, tasks require different levels of worker qualification. The first worker type (type-1) is the most qualified worker type, whereas the last worker type (type-3) is the least qualified one. Accordingly, the data set is further adapted to our problem by considering the percentages of type-1, type-2 and type-3 tasks as 20%, 30%, 50%, respectively as in Sungur & Yavuz (2015).

The processing times of the original problem instances are assumed to belong to the type-1 worker. In Corominas, Pastor & Plans (2008), there are two types of workers which are skilled permanent worker and unskilled temporary worker. If a

task is assigned to a skilled permanent worker, the task time for this worker is assumed to be the standard time. On the other hand, when the task is assigned to an unskilled permanent worker, the task time for this worker is the standard time multiplied by a time factor (w_1) which is greater than one. The processing times of tasks for type-2 and type-3 workers are determined as $t_{i,h+1} = w_1 * t_{i,h}$, by rounding the resulting values to nearest integers, where w_1 equals to 1.1. The cost of workers is determined as $Wcost_{h+1} = w_2 * Wcost_h$, where w_2 is the cost factor and equals to 0.7, and the cost of type-1 workers is set to $Wcost_1$=100. The similar settings are used by Sungur & Yavuz (2015) as well. We use the same formulas to arrange the processing times and worker costs.

We consider that the number of part positions and the number of parallel stations are two and three, respectively, i.e. $L = 2$ and $K = 3$.

The cost for paralleling k identical stations is determined as $W_k = \frac{k*W_{k-1}}{k-1} + \varepsilon$, where the cost of opening only one station is equal to 10 and $\varepsilon = 2$ (Bukchin & Rubinovitz, 2003).

The percentage of part positions corresponding to front and back tasks are taken as 70% and 30% of the total number of tasks, respectively, and the positions of the tasks are determined randomly. As additional operational constraints, Task 1 should be performed alone at a station, whereas Tasks 19 and 20 should be assigned to the same station.

The selected parameter combination corresponding to a selected problem instance is given in Table 5.1. Here, the first column *Problem no* indicates the size of the problem data as to small, medium, and large with S (small), M (medium), L (large) letters, and the number of the problem instance as referenced in Otto et al. (2013). The *Category* column shows the order strength, task time distribution and trickiness degree for a problem instance.

Table 5.1 A selected problem instance

Problem no	Order strength	Task time distribution	Trickiness category	Category
S291	0.2	Peak at the bottom	Less tricky	0.2/PB/LT

5.2 SA Parameters Tuning

The SA algorithm contains a large number of parameters, and the values of these parameters directly affect the solution quality of the algorithm. There are various experimental designs to decide on these values. Essentially, traditional experimental design procedures are too complicated and are not easy to use since a large number of experimental work has to be carried out when the number of process parameters increases such as the full factorial design in which the effects of these factors are investigated simultaneously, and all possible combinations of the values of the parameters are tried (Cochran & Cox, 1992). For this reason, in order to calibrate the proposed SA, the Taguchi method is used in this thesis to statistically design the experimental investigation. The Taguchi method uses a special design of orthogonal arrays to study the entire parameter space with only a small number of experiments. It divides the factors into two basic clusters: controllable and noise factors (uncontrollable). The noise factors are those over which we have no direct control. Since the elimination of the noise factors is impractical and often impossible, the Taguchi method seeks to minimize the effect of noise and to determine the optimal levels of important controllable factors based on the concept of robustness (Tsai, Ho, Liu & Chou, 2007). Besides determining the optimal levels, Taguchi establishes the relative significance of individual factors in terms of their main effects on the objective function. The variation of the output results is measured by means of Signal-to-Noise (S/N) ratio where the terms “signal” and “noise” indicate the mean response variable and the standard deviation, respectively.

Orthogonal arrays of Taguchi, the signal-to-noise (S/N) ratio, and the analysis of variance (ANOVA) are employed to find the optimum levels of factors and their effects on the objective function. The S/N ratio indicates the amount of variation

present in the response variable. The aim is to maximize the S/N ratio. Taguchi method categorizes objective functions into three groups: the smaller-the-better type, the larger-the-better type, and the nominal-is-best type. For our problem type, the smaller-the-better, the S/N ratio is formulated as in Formula 5.1 (Phadke, 1989):

$$\text{S/N ratio} = -10 \times \log_{10}(objective\ function)^2 \tag{5.1}$$

The aim of the Taguchi method is to perform a small number of experiments for solving the problem with orthogonal array that contains the parameter space. According to this method, factors are categorized into two groups: controllable and noise factors (uncontrollable). Thanks to the signal to noise (*S/N*) ratio and the analysis of variance (ANOVA), optimum levels of parameters and their effects on the objective function are determined.

For the Taguchi method, three problems are randomly chosen from different problem size. S325, M348 and L326 are selected to adjust parameters of the small, medium and large data sets, respectively.

Factors and their levels are presented in Table 5.2.

Table 5.2 Factors and their levels

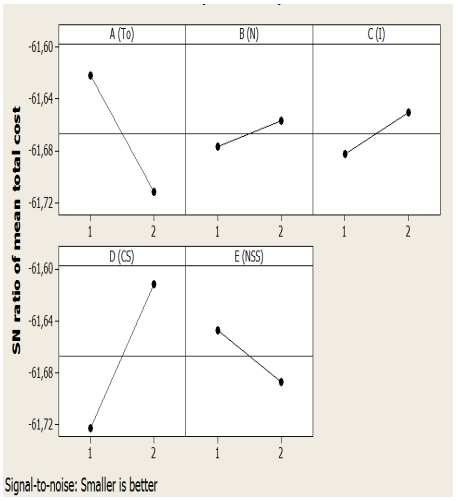
Factor	Small data set	Medium data set	Large data set
A (T_0)- Initial temperature	A1 (10) A2 (20)	A1 (10) A2 (15)	A1 (10) A2 (15)
B (N)- Number of temperatures between T_0 and T_f	B1 (30) B2 (50)	B1 (60) B2 (80)	B1 (80) B2 (100)
C (I)- Number of neighborhood searches in each temperature	C1 (30) C2 (60)	C1 (40) C2 (60)	C1 (30) C2 (50)
D (CS)- Cooling schedule type	D1 (Linear) D2 (exponential)	D1 (Linear) D2 (exponential)	D1 (Linear) D2 (exponential)
E (NSS)- Neighborhood search structure	E1 (Single_transfer - Swap-Stage separation) E2 (Single_transfer - Swap- Single_transfer -Stage separation)	E1 (Single_transfer - Swap-Stage separation) E2 (Single_transfer - Swap- Single_transfer -Stage separation)	E1 (Single_transfer - Swap-Stage separation) E2 (Single_transfer - Swap- Single_transfer -Stage separation)

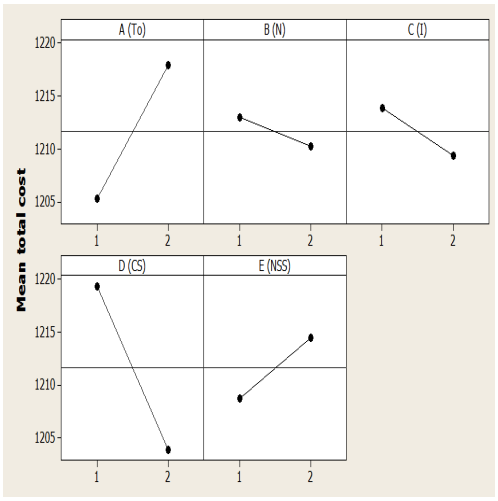
As seen from Table 5.2, each group has five factors, each with levels, and accordingly the proper orthogonal array fulfilling all our requirements is selected as L8 orthogonal array which is denoted in Table 5.3. We implemented the proposed SA algorithm five times for each selected problem and used the mean total cost value to compare their performances. After the Taguchi experimental design, the *S/N* ratio and mean total cost for each level of the factors from each group are given in Figure 5.1.

Table 5.3 The orthogonal array L_8

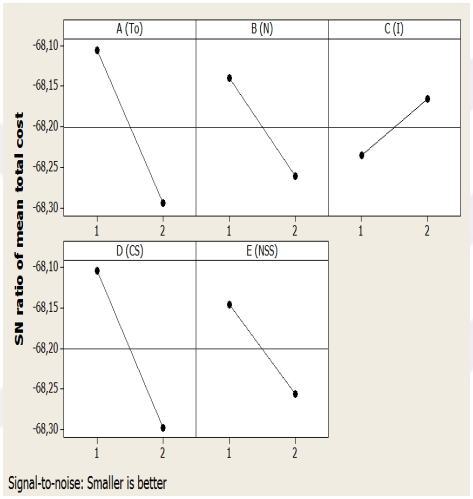
Trial	A (To)	B (N)	C (I)	D (CS)	E (NSS)
1	1	1	1	1	1
2	1	1	1	2	2
3	1	2	2	1	1
4	1	2	2	2	2
5	2	1	2	1	2
6	2	1	2	2	1
7	2	2	1	1	2
8	2	2	1	2	1

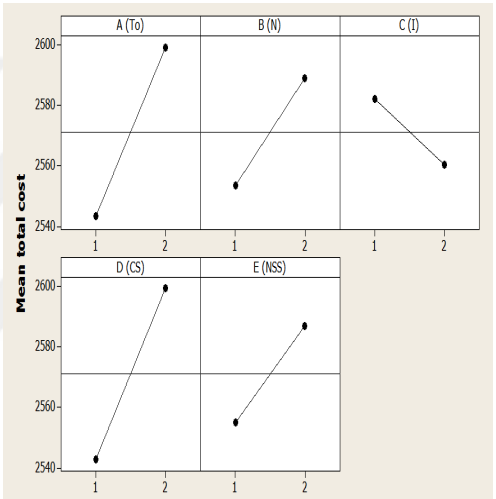
(a)



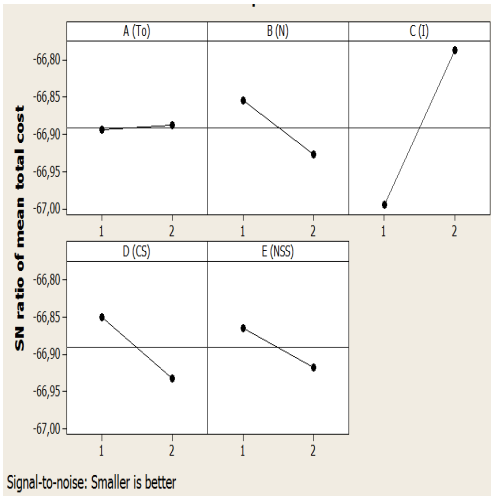


(b)





(c)



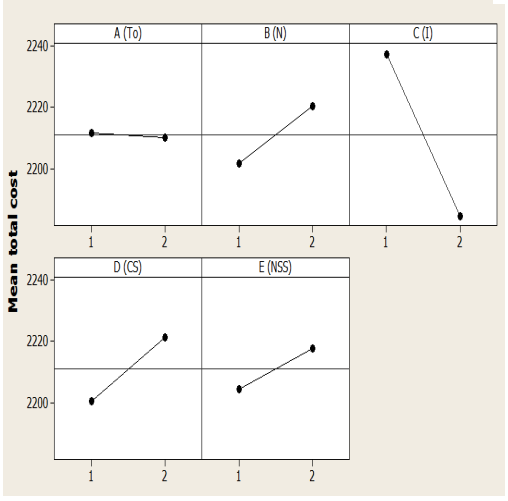


Figure 5.1 The mean S/N ratio and the mean total cost plots for each level of factors

(d)

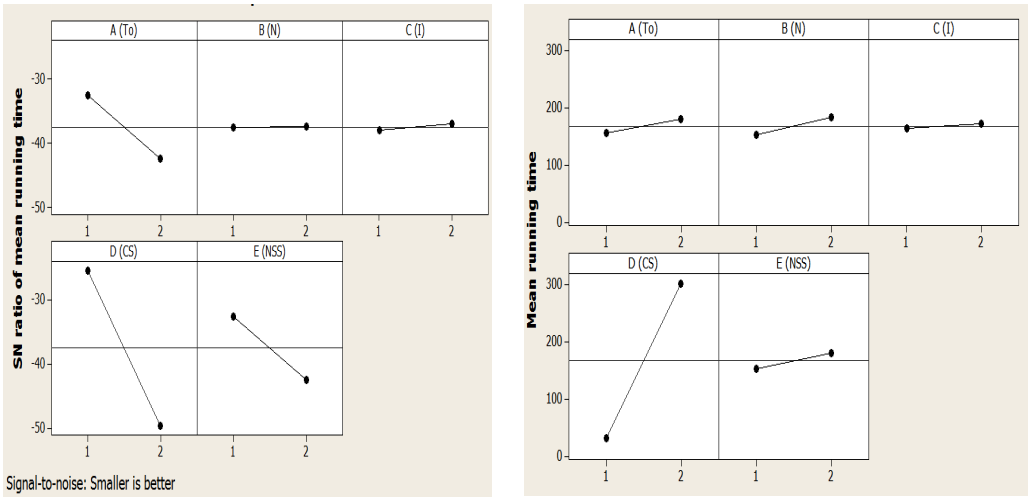


Figure 5.1 continues

Analysis of variance for the SN ratios in Table 5.4 reveals that it is found that parameter selection in the 95% confidence interval is not significant when p values are examined. However, the parameters are determined according to Figure 5.1. As seen from Figure 5.1 (a), A1, B2, C2, D2 and E1 are the preferred levels of the factors A, B, C, D, E, which belong to small size problems, respectively, i.e., T_0*=10, N=50, I=60, CS=exponential* and *NSS= transfer-swap-stage seperation*. As seen from Figure 5.1 (b), A1, B1, C2, D1 and E1 are the preferred levels of the factors A, B, C, D, E, which belong to medium size problems, respectively, i.e., T_0*=10, N=60, I=60, CS=linear* and *NSS= transfer-swap-stage seperation*. As seen from Figure 5.1 (c), B1, C2, D1 and E1 are the preferred levels of the factors B, C, D, E, which belong to large size problems, respectively, i.e., *N=80, I=50, CS=linear* and *NSS= transfer-swap-stage seperation*. According to this Taguchi experiment, there is little difference between A1 and A2, thus more investigation is needed to determine the suitable level of factor A. As it is seen in the Figure 5.1 (d), it is determined as A1, i.e., T_0 *=10*, according to the mean running time which is another performance criterion.

The effects of each factor on the objective function are determined by the ANOVA for each problem size separately. Table 5.4 reports the results of the analysis. In the Table 5.4, *DF*, *SS*, *MS* and *F* refer to degree of freedom, sum of squares, mean of squares and distribution, respectively. *Error* is a factor which

includes the impacts of all unrecognized factors. As the *p-value* diminishes, the level of the importance of the factor increases. For small size problems, the results of the variance analysis show that the most important factor is *CS* with a share of 0.118. T_0, *NSS*, *I* and *N* follow this factor sequentially. In the *Rank* column, the order of influence levels of factors is shown. For medium size problems, the most important factor is *CS* with a share of 0.263. T_0, *N* and *NSS* follow this factor, respectively, and *I* has the least influence. And finally, the most important factor is *I* with a share of 0.052 for large size problems. *CS*, *N*, *NSS* and T_0 follow this factor sequentially.

Table 5.4 Analysis of variance for SN ratios

Small data set						
Source	DF	SS	MS	F	p-value	Rank
A (To)	1	0.016166	0.016166	4.62	0.165	2
B (N)	1	0.000791	0.000791	0.23	0.681	5
C (I)	1	0.002067	0.002067	0.59	0.522	4
D (CS)	1	0.024637	0.024637	7.04	0.118	1
E (NSS)	1	0.003226	0.003226	0.92	0.438	3
Error	2	0.006999	0.003499			
Total	7	0.053887				
Medium data set						
Source	DF	SS	MS	F	p-value	Rank
A (To)	1	0.071365	0.071365	2.23	0.274	2
B (N)	1	0.029546	0.029546	0.92	0.438	3
C (I)	1	0.009861	0.009861	0.31	0.634	5
D (CS)	1	0.075789	0.075789	2.37	0.263	1
E (NSS)	1	0.024161	0.024161	0.76	0.476	4
Error	2	0.063913	0.031956			
Total	7	0.274634				
Large data set						
Source	DF	SS	MS	F	p-value	Rank
A (To)	1	0.000093	0.000093	0.02	0.902	5
B (N)	1	0.010836	0.010836	2.26	0.272	3
C (I)	1	0.086073	0.086073	17.92	0.052	1
D (CS)	1	0.013416	0.013416	2.79	0.237	2
E (NSS)	1	0.005396	0.005396	1.12	0.400	4
Error	2	0.009605	0.004803			
Total	7	0.125419				

5.3 Computational Results

The proposed IP model and the SA algorithm are used to solve the test problems using CPLEX 12.6.1 and Matlab 7.12.0, respectively. The computer system used is Intel® Core™ i7-5500U CPU with 12.0 GB RAM and 2.40 GHz speed. We limited the computation time to 1800 seconds for the IP model. The proposed SA algorithm was run twenty times for each problem instance.

Twenty problems were used for each problem size. Table 5.5 to 5.7 show the computational results of small size, medium and large problems respectively. The first *Problem no* indicates the size of the problem data as to small, medium, and large with S (small), M (medium), L (large) letters, and the number of the problem instance as referenced in Otto et al. (2013). The second column with heading *Category* contains the order strength, task time distribution and trickiness degree for a problem instance. For the IP model, the third and fourth columns indicate the number of constraints and variables required respectively. The fifth and sixth columns show the total cost value obtained from the IP model and the solution time of the model for each problem instance. The seventh column contains the best total cost found among 20 runs of the SA algorithm. The eighth column provides the average total cost value of 20 runs. Moreover, the ninth column contains the standard deviation of total cost over 20 runs for each problem instance. The tenth column provides the average computational times. Finally, the eleventh and twelfth columns the relative percentage deviation (RPD) of minimum total cost and computational time obtained by the SA algorithm from that of obtained by the IP model solution, respectively. The *RPD* is calculated by the following formula 5.2.

$$RPD = \frac{Alg - Min}{Min} * 100 \tag{5.2}$$

where Min is the solution of the mathematical model, and Alg is the minimum total cost value of the proposed algorithm. Clearly, lower values are preferred.

Table 5.5 The computational results of small sized test problems

Small size problems		IP model				SA algorithm				Total cost	Time
No	Category	# of const.	# of var.	Total cost	CPU time (sec)	Total cost			Time (sec)		
						Best	Avrg	Stdev	Avrg	RPD %	RPD %
S291	0.2/PB/LT	616	1461	**549**	10	**549**	549	0	56.9	0	469
S303	0.2/PB/TR	613	1461	**469**	10	**469**	469	0	83	0	730
S313	0.2/PB/VT	616	1461	**528**	16	**528**	528	0	98.8	0	517.5
S325	0.2/PM/LT	615	1461	**1190**	1800	**1187**	1202.3	8.7	35	-0.25	-98.05
S321	0.2/PM/TR	615	1461	**1093**	1800	**1095**	1108.2	3.1	34.5	0.18	-98.08
S319	0.2/PM/VT	615	1461	**1159**	1800	**1161**	1200.3	17.1	74.6	0.17	-95.8
S329	0.2/PM/ET	615	1461	**1009**	1800	**1009**	1026.5	26.9	60	0	-96.6
S324	0.2/PM/ET	615	1461	**952**	1800	**954**	971.2	15.9	36.8	0.21	-97.9
S346	0.2/BM/LT	615	1461	**667**	33	**667**	681.1	15.8	84.7	0	156.6
S348	0.2/BM/TR	615	1461	**661**	41	**661**	668.5	13.3	174.6	0	325.8
S443	0.9/PB/LT	634	1461	**806**	20	**806**	806	0	237.9	0	1089.5
S441	0.9/PB/TR	630	1461	**764**	21	**764**	764	0	120.5	0	473.8
S453	0.9/PB/VT	633	1461	**646**	15	**646**	646	0	532.1	0	3447.3
S469	0.9/PM/LT	628	1461	**1151**	1800	**1153**	1164.9	6.1	218.2	0.17	-87.8
S466	0.9/PM/TR	630	1461	**1171**	584	**1173**	1203	7.1	162.8	0.17	-72.1
S476	0.9/PM/VT	637	1461	**1047**	884	**1059**	1070	7	148.2	1.14	-83.2
S492	0.9/BM/LT	632	1461	**806**	36	**806**	813.4	12.9	228.9	0	535.8
S494	0.9/BM/TR	625	1461	**699**	55	**699**	699.2	0.6	692.5	0	1159.09
S491	0.9/BM/VT	631	1461	**806**	50	**806**	814	11.2	350	0	600
S497	0.9/BM/ET	629	1461	**705**	23	**705**	706.8	0.6	146.6	0	537.3
Average				843.9	629.9	844.8	854.5	7.3	178.8	0.09	465.6

Back tasks are 2, 7, 10, 12, 13, 16 for small data sets (S)

As seen from Table 5.5, for the small sized problems the SA algorithm can find the same best 13 solutions obtained by the IP model. The model could only verify the optimality of 14 problems out of 20. For problem S325, the algorithm achieves a better solution compared to the model solution found within the time limit. For the remaining instances, the algorithm achieves very approximate solutions with only 0.34% deviation from the model solutions in average. When the average deviation value considering all the problems is taken into account, it is as small as 0.09%. The SA algorithm has obtained the optimal solution values for six problems with zero standard deviation. The total cost values are obtained with 7.3 average standard deviation value by the algorithm.

Table 5.6 The computational results of medium sized test problems

Medium size problems		**IP model**				**SA algorithm**				Total cost	Time
No	Category	# of const.	# of var.	Total cost	CPU time (sec)	Total cost			Time (sec)		
						Best	Avrg	Stdev	Avrg	RPD %	RPD %
M304	0.2/PB/LT	1556	8151	**857**	1594	**901**	918.6	26.9	4.4	5.1	-99.7
M325	0.2/PB/TR	1553	8151	**937**	1800	**960**	1009.4	31.5	2.8	2.4	-99.8
M309	0.2/PB/ET	1560	8151	**827**	1800	**924**	951	9.2	6.5	11.7	-99.6
M348	0.2/PM/TR	1553	8151	**2670**	1800	**2516**	2587.4	44.8	13.9	-5.76	-99.2
M346	0.2/PM/ET	1556	8151	**2473**	1800	**2439**	2529.4	53.8	7.3	-1.37	-99.5
M374	0.2/BM/LT	1555	8151	**1218**	1800	**1281**	1292.5	15.5	6	5.17	-99.6
M360	0.2/BM/TR	1555	8151	**1332**	1800	**1511**	1539.5	20.2	4.4	13.4	-99.7
M366	0.2/BM/VT	1553	8151	**1334**	1800	**1535**	1535	0	14.5	15.0	-99.1
M375	0.2/BM/ET	1555	8151	**1383**	1800	**1471**	1508.7	30.4	3.9	6.36	-99.7
M452	0.9/PB/LT	1606	8151	**1376**	1800	**1408**	1408	0	449	2.32	-75
M451	0.9/PB/TR	1601	8151	**1467**	1800	**1437**	1437	0	134.2	-2.04	-92.5
M458	0.9/PB/VT	1608	8151	**1485**	1800	**1464**	1464	0	98.2	-1.41	-94.5
M475	0.9/PB/ET	1601	8151	**3407**	1800	**1464**	1464	0	49.5	-57	-97.2
M495	0.9/PM/ET	1604	8151	**3709**	1800	**2892**	2920.7	26.9	82.4	-22	-95.4
M525	0.9/BM/LT	1616	8151	**1540**	1800	**1449**	1449	0	74.6	-5.9	-95.8
M522	0.9/BM/TR	1614	8151	**1623**	1800	**1587**	1587	0	174.4	-2.21	-90.3
M502	0.9/BM/VT	1601	8151	**3219**	1800	**1469**	1469	0	105.1	-54.3	-94.1
M504	0.9/BM/ET	1598	8151	**1893**	1800	**1590**	1590.4	0.5	42.2	-16	-97.6
M505	0.9/BM/ET	1599	8151	**2615**	1800	**1791**	1791	0	77.6	-31.5	-95.6
M524	0.9/BM/ET	1616	8151	**2459**	1800	**1869**	1871.1	6.6	95.6	-23.9	-94.6
Average				1891	1790	1597.9	1616.1	13.3	72.3	-8.09	-95.9

Back tasks are 2, 5, 6, 9, 14, 16, 18, 22, 24, 30, 37, 41, 44, 46, 49 for medium data sets (M)

As seen from Table 5.6, for the medium sized problems the IP model has only solved the first problem instance optimally. All other solutions are the best solutions obtained within the time limit. The proposed SA algorithm achieves better results than the IP model for 12 problems. For the remaining eight instances, the algorithm achieves approximate solution values with an average of 7.7% deviation in relatively shorter computation times compared to the IP model. When the average deviation value considering all the problems is taken into account, it is -8.09%, which indicates that the SA algorithm outperforms the IP model with 8.09% better solution values in average. As far as the computation time is considered, the SA algorithm outperforms the IP model with -95.9% RPD.

Table 5.7 The computational results of large sized test problems

Large size problems		IP model				SA algorithm				Total cost	Time
No	Category	# of const.	# of var.	Total cost	CPU time (sec)	Total cost Best	Avrg	Stdev	Time (sec) Avrg	RPD %	RPD %
L329	0.2/PB/LT	3124	31301	**2996**	1800	**1576**	1719.5	71.5	3.2	-47.3	-99.8
L332	0.2/PB/LT	3119	31301	**1621**	1800	**1700**	1802	49.7	4.8	4.8	-99.7
L326	0.2/PB/TR	3121	31301	**1759**	1800	**1455**	1593.1	65.8	3.4	-17.2	-99.8
L333	0.2/PB/TR	3116	31301	**1730**	1800	**1812**	1916.2	45.7	3.4	4.7	-99.8
L327	0.2/PB/VT	3118	31301	**2700**	1800	**1676**	1759.7	43.3	4.7	-37.9	-99.7
L331	0.2/PB/VT	3127	31301	**2840**	1800	**1767**	1877.4	54.1	4.8	-37.7	-99.7
L342	0.2/PB/ET	3123	31301	**3642**	1800	**1641**	1739.9	50.1	3.9	-54.9	-99.7
L352	0.2/PM/ET	3117	31301	**5597**	1800	**5710**	5800.5	55	114.8	2	-93.6
L311	0.2/BM/TR	3119	31301	**2866**	1800	**2197**	2291	55.2	5.5	-23.3	-99.6
L312	0.2/BM/TR	3136	31301	**3367**	1800	**2441**	2522.7	44.8	12.4	-27.5	-99.3
L305	0.2/BM/VT	3113	31301	**4240**	1800	**2377**	2452	44.3	5.3	-43.9	-99.7
L301	0.2/BM/ET	3119	31301	**3088**	1800	**2417**	2500.2	82.6	3.8	-21.7	-99.7
L325	0.2/BM/ET	3126	31301	**4309**	1800	**2714**	2821.4	57.2	4.9	-37	-99.7
L479	0.9/PB/LT	3255	31301	-*	1800	**2323**	2415.7	68.8	78.3	-	-95.6
L478	0.9/PB/TR	3262	31301	-*	1800	**2135**	2161.7	13.4	223.4	-	-87.5
L499	0.9/PB/VT	3253	31301	**6908**	1800	**1835**	1869.9	28.9	72.9	-73.4	-95.9
L496	0.9/PB/ET	3251	31301	**4594**	1800	**2363**	2407.6	20.9	97.1	-48.5	-94.6
L467	0.9/BM/TR	3225	31301	-*	1800	**2408**	2445.3	17	91.4	-	-94.9
L453	0.9/BM/VT	3223	31301	-*	1800	**2516**	2580.6	66.7	77.6	-	-95.6
L473	0.9/BM/ET	3247	31301	-*	1800	**3327**	3383.7	19.9	553.2	-	-69.2
Average				2759.5	1800	2319.5	2403	47.7	68.44	-30.6	-96.1

Back tasks are 4, 12, 14, 15, 17, 19, 20, 26, 28, 32, 35, 39, 44, 45, 49, 50, 51, 55, 59, 65, 66, 68, 70, 71, 76, 77, 80, 85, 90, 96 for large data sets (L).

*No feasible solution

As seen from Table 5.7, for the large sized problems the IP model has found feasible solutions for 15 problems within the time limit, whereas no feasible solution has been found for the remaining problems. The proposed SA algorithm achieves better results for 12 problems than those found by the IP model. When the average deviation value considering all the problems is taken into account, it is -30.6%, which indicates that the SA algorithm outperforms the IP model with 30.6% better solution values in average. As far as the computation time is considered, the SA algorithm outperforms the IP model with -96.1% RPD. It can been seen from these

computational results that the SA algorithm can find much better results in less computational times than the IP model as the problem size is increased.

CHAPTER SIX
CONCLUSION

In this thesis, we consider the assembly line balancing problem with hierarchical worker assignment (ALBHW). In addition to precedence and cycle time constraints, we take into account positional constraints, station paralleling and task assignment restrictions. Also, tasks require different levels of worker qualification and there is a hierarchical workforce structure in which a lower qualified worker can be substituted by higher qualified ones with a higher cost. The objective of this problem is to find the optimal assignment of workers and tasks to the stations, and to decide on the optimal parallel stations such that total cost is minimized for a given cycle time.

In order to solve this problem, we have initially developed a new integer programming (IP) model. The IP model guarantees the optimum solution. However, as the problem size increases, the model takes a long time to reach a feasible solution and in some cases it cannot even find a feasible solution. For this reason, we have proposed a solution approach based on the simulated annealing (SA) algorithm in order to find high quality solutions in reasonable computational times. In the development of the SA algorithm, firstly, a modified version of the Ranked Positional Weight heuristic has been applied for the initial solution generation. The algorithm employs four kinds of neighborhood search structures, which are single_transfer, two_transfer, swap and stage separation, respectively, in creating neighbor solutions from the current solution. In order to promote the solution quality of the SA algorithm, it has been hybridized with a local search procedure. Additionally, a giant leap procedure has been used to investigate an inferior or unvisited search space for the probability of finding a better solution. To find the optimal parameter levels of the SA algorithm, the Taguchi method has been employed. Finally, a set of test problems have been solved using both the proposed IP model and SA algorithm. The computational results show that as the problem size increases, the proposed SA algorithm obtains better solutions in shorter computation times than the IP model.

As a future study, other metaheuristic algorithms such as tabu search algorithm, iterated local search, variable neighborhood search, guided local search or population based metaheuristic algorithms such as genetic algorithm, particle swarm optimization can be used to solve this problem. Also the ALBHW can be extended for different line configurations such as U-shaped assembly lines, with ergonomic constraints, or for multi-model product or mixed-model assembly lines instead of single model product assembly lines.

REFERENCES

Akpınar, S., & Bayhan, G. M. (2011). A hybrid genetic algorithm for mixed model assembly line balancing problem with parallel workstations and zoning constraints. *Engineering Applications of Artificial Intelligence*, *24*(3), 449-457.

Akyol, S. D., & Baykasoğlu, A. (2016). ErgoALWABP: a multiple-rule based constructive randomized search algorithm for solving assembly line worker assignment and balancing problem under ergonomic risk factors. *Journal of Intelligent Manufacturing*, 1-12.

Araújo, F. F., Costa, A. M., & Miralles, C. (2012). Two extensions for the ALWABP: Parallel stations and collaborative approach. *International Journal of Production Economics*, *140*(1), 483-495.

Askin, R. G., & Zhou, M. (1997). A parallel station heuristic for the mixed-model production line balancing problem. *International Journal of Production Research*, *35*(11), 3095-3106.

Bard, J. F. (1989). Assembly line balancing with parallel workstations and dead time. *The International Journal Of Production Research*, *27*(6), 1005-1018.

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.

Billionnet, A. (1999). Integer programming to schedule a hierarchical workforce with variable demands. *European Journal of Operational Research*, *114*(1), 105-114.

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.

Borba, L., & Ritt, M. (2014). A heuristic and a branch-and-bound algorithm for the Assembly Line Worker Assignment and Balancing Problem. *Computers & Operations Research*, *45*, 87-96.

Boysen, N., Fliedner, M., & Scholl, A. (2007). A classification of assembly line balancing problems. *European journal of operational research*, *183*(2), 674-693.

Bukchin, J., & Rubinovitz, J. (2003). A weighted approach for assembly line design with station paralleling and equipment selection. *IIE transactions*, *35*(1), 73-85.

Buxey, G. M. (1974). Assembly line balancing with multiple stations. *Management science*, 20(6), 1010-1021.

Cakir, B., Altiparmak, F., & Dengiz, B. (2011). Multi-objective optimization of a stochastic assembly line balancing: A hybrid simulated annealing algorithm. *Computers & Industrial Engineering*, *60*(3), 376-384.

Černý, V. (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, *45*(1), 41-51.

Chaves, A. A., Miralles, C., & Lorena, L. A. N. (2007). Clustering search approach for the assembly line worker assignment and balancing problem. *In Proceedings of the 37th International Conference on Computers and Industrial engineering*, Alexandria, Egypt (pp. 1469-1478).

Chaves, A. A., Lorena, L. A. N., & Miralles, C. (2009, October). Hybrid metaheuristic for the assembly line worker assignment and balancing problem. *In International Workshop on Hybrid Metaheuristics* (pp. 1-14). Springer Berlin Heidelberg.

Cochran, W.G., & Cox, G.M. (1992). *Experimental designs*. 2nd ed. New York: Wiley.

Corominas, A., Pastor, R., & Plans, J. (2008). Balancing assembly line with skilled and unskilled workers. *Omega*, *36*(6), 1126-1132.

Emmons, H., & Burns, R. N. (1991). Off-day scheduling with hierarchical worker categories. *Operations Research*, *39*(3), 484-495.

Erel, E., Sabuncuoglu, I., & Aksu, B. A. (2001). Balancing of U-type assembly systems using simulated annealing. *International Journal of Production Research*, *39*(13), 3003-3015.

Essafi, M., Delorme, X., Dolgui, A., & Guschinskaya, O. (2010). A MIP approach for balancing transfer line with complex industrial constraints. *Computers & Industrial Engineering*, *58*(3), 393-400.

Helgeson, W. B., & Birnie, D. P. (1961). Assembly line balancing using the ranked positional weight technique. *Journal of Industrial Engineering*, *12*(6), 394-398.

Hung, R., & Emmons, H. (1993). Multiple-shift workforce scheduling under the 3-4 compressed workweek with a hierarchical workforce. *IIE transactions*, *25*(5), 82-89.

Hung, R. (1994). Single-shift off-day scheduling of a hierarchical workforce with variable demands. *European Journal of Operational Research*, *78*(1), 49-57.

Jackson, J. R. (1956). A computing procedure for a line balancing problem. *Management Science*, *2*(3), 261-271.

Johnson, R. V. (1983). A branch and bound algorithm for assembly line balancing problems with formulation irregularities. *Management Science*, *29*(11), 1309-1324.

Kellegöz, T. (2016). Assembly line balancing problems with multi-manned stations: a new mathematical formulation and Gantt based heuristic method. *Annals of Operations Research*, 1-28.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. science, *220*(4598), 671-680.

Lapierre, S. D., & Ruiz, A. B. (2004). Balancing assembly lines: an industrial case study. *Journal of the Operational Research Society*, *55*(6), 589-597.

Mastor, A. A. (1970). An experimental investigation and comparative evaluation of production line balancing techniques. *Management Science*, *16*(11), 728-746.

McMullen, P. R., & Frazier, G. V. (1998). Using simulated annealing to solve a multiobjective assembly line balancing problem with parallel workstations. *International Journal of Production Research*, *36*(10), 2717-2741.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, *21*(6), 1087-1092.

Miralles, C., Garcia-Sabater, J. P., Andres, C., & Cardos, M. (2007). Advantages of assembly lines in sheltered work centres for disabled. A case study. *International Journal of Production Economics*, *110*(1), 187-197.

Miralles, C., García-Sabater, J. P., Andrés, C., & Cardós, M. (2008). Branch and bound procedures for solving the assembly line worker assignment and balancing problem: Application to sheltered work centres for disabled. *Discrete Applied Mathematics*, *156*(3), 352-367.

Mitchell, J. (1957). A computational procedure for balancing zoned assembly lines. In *Research report No. 6-94801-1R3*. Westinghouse Research Lab Pittsburgh, Penn.

Mladenović, N., & Hansen, P. (1997). Variable neighborhood search. *Computers & operations research*, *24*(11), 1097-1100.

Moreira, M. C. D. O., & Costa, A. M. (2009). A minimalist yet efficient tabu search algorithm for balancing assembly lines with disabled workers. *Anais do XLI Simpósio Brasileiro de Pesquisa Operacional. Porto Seguro, Brasil*, 660-671.

Moreira, M. C. O., Ritt, M., Costa, A. M., & Chaves, A. A. (2012). Simple heuristics for the assembly line worker assignment and balancing problem. *Journal of Heuristics*, *18*(3), 505-524.

Moreira, M. C. O., Miralles, C., & Costa, A. M. (2015). Model and heuristics for the assembly line worker integration and balancing problem. *Computers & Operations Research*, *54*, 64-73.

Moreira, M. C. O., Pastor, R., Costa, A. M., & Miralles, C. (2017). The multi-objective assembly line worker integration and balancing problem of type-2. *Computers & Operations Research*, *82*, 114-125.

Mutlu, Ö., Polat, O., & Supciller, A. A. (2013). An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-II. *Computers & Operations Research*, *40*(1), 418-426.

Naderi, B., Zandieh, M., & Roshanaei, V. (2009). Scheduling hybrid flowshops with sequence dependent setup times to minimize makespan and maximum tardiness. *The International Journal of Advanced Manufacturing Technology*, *41*(11-12), 1186-1198.

Narasimhan, R. (1997). An algorithm for single shift scheduling of hierarchical workforce. *European Journal of Operational Research*, *96*(1), 113-121.

Otto, A., Otto, C., & Scholl, A. (2013). Systematic data generation and test design for solution algorithms on the example of SALBPGen for assembly line balancing. *European Journal of Operational Research*, *228*(1), 33-45.

Pastor, R., & Corominas, A. (2010). A bicriteria integer programming model for the hierarchical workforce scheduling problem. *Journal of Modelling in Management*, *5*(1), 54-62.

Pereira, J. (2018). The Robust (minmax regret) assembly line worker assignment and balancing problem. *Computers & Operations Research.*

Phadke, M. S. (1989). Quality engineering using robust design. *Englewood Cliffs, New Jersey: PTR Prentice-Hall Inc.*

Pinto, P., Dannenbring, D. G., & Khumawala, B. M. (1975). A branch and bound algorithm for assembly line balancing with paralleling. *The International Journal of Production Research, 13*(2), 183-196.

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.

Ramezanian, R., & Ezzatpanah, A. (2015). Modeling and solving multi-objective mixed-model assembly line balancing and worker assignment problem. *Computers & Industrial Engineering*, *87*, 74-80.

Ritt, M., Costa, A. M., & Miralles, C. (2016). The assembly line worker assignment and balancing problem with stochastic worker availability. *International Journal of Production Research*, *54*(3), 907-922.

Roshani, A., & Giglio, D. (2016). Simulated annealing algorithms for the multi-manned assembly line balancing problem: minimising cycle time. *International Journal of Production Research*, 1-21.

Salveson, M. E. (1955). The assembly line balancing problem. *Journal of Industrial Engineering*, *6*(3), 18-25.

Scholl, A. (1999). Balancing and sequencing of assembly lines (No. 10881). Darmstadt Technical University, Department of Business Administration, Economics and Law, Institute for Business Studies (BWL).

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.

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.

Shtub, A., & Dar-El, E. M. (1989). A methodology for the selection of assembly systems. *The International Journal Of Production Research*, *27*(1), 175-186.

Sungur, B., & Yavuz, Y. (2015). Assembly line balancing with hierarchical worker assignment. *Journal of Manufacturing Systems*, *37*, 290-298.

Suresh, G., & Sahu, S. (1994). Stochastic assembly line balancing using simulated annealing. *The International Journal of Production Research, 32*(8), 1801-1810.

Tonge, F. M. (1960). A heuristic program for assembly line balancing.

Tsai, J. T., Ho, W. H., Liu, T. K., & Chou, J. H. (2007). Improved immune algorithm for global numerical optimization and job-shop scheduling problems. *Applied Mathematics and Computation*, *194*(2), 406-424.

Tuncel, G., & Topaloglu, S. (2013). Assembly line balancing with positional constraints, task assignment restrictions and station paralleling: A case in an electronics company. *Computers & Industrial Engineering*, *64*(2), 602-609.

Vilà, M., & Pereira, J. (2014). A branch-and-bound algorithm for assembly line worker assignment and balancing problems. *Computers & Operations Research*, *44*, 105-114.

Vilarinho, P. M., & Simaria, A. S. (2002). A two-stage heuristic method for balancing mixed-model assembly lines with parallel workstations. *International Journal of Production Research*, *40*(6), 1405-1420.

Zacharia, P. T., & Nearchou, A. C. (2016). A population-based algorithm for the bi-objective assembly line worker assignment and balancing problem. *Engineering Applications of Artificial Intelligence*, *49*, 1-9.