REBALANCING IN ASSEMBLY LINES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

UTKU GİRİT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCİENCE
IN
INDUSTRİAL ENGİNEERİNG

AUGUST 2019

Approval of the thesis:

REBALANCING IN ASSEMBLY LINES

submitted by **UTKU GİRİT** in partial fulfillment of the requirements for the degree of **Master of Science in Industrial Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** ________________

Prof. Dr. Yasemin Serin
Head of Department, **Industrial Engineering** ________________

Prof. Dr. Meral Azizoğlu
Supervisor, **Industrial Engineering, METU** ________________

Examining Committee Members:

Prof. Dr. Ömer Kırca
Industrial Engineering, METU ________________

Prof. Dr. Meral Azizoğlu
Industrial Engineering, METU ________________

Assoc. Prof. Dr. Sakine Batun
Industrial Engineering, METU ________________

Assoc. Prof. Dr. Mustafa Kemal Tural
Industrial Engineering, METU ________________

Assoc. Prof. Dr. Banu Yüksel Özkaya
Industrial Engineering, Hacettepe University ________________

Date: 29.08.2019

ABSTRACT

REBALANCING IN ASSEMBLY LINES

Girit, Utku
Master of Science, Industrial Engineering
Supervisor: Prof. Dr. Meral Azizoğlu

August 2019, 70 pages

Assembly line balancing is an important and well recognized operations research problem. The current balancing might not stay optimal for a long time due to the changing conditions. The changing conditions, like disruptions in one or more workstations, may cause some inefficiencies, even infeasibilities, for the current balance. In this study, after the disruption, we aim to rebalance the assembly line by considering the trade-off between workload balancing (efficiency measure) and total displacement amount for the tasks assigned to the different workstations (stability measure).

We try to generate all non-dominated objective function vectors for the defined efficiency and stability measures. Two algorithms are developed: classical approach and tabu search algorithm. Our experiments have shown that the classical approach returns exact non-dominated objective vectors with up to 40 tasks and 7 workstations in one hour and the tabu search algorithm returns approximate non-dominated objective vectors that are very close to their exact counterparts and can solve large sized instances, very quickly.

Keywords: Assembly Lines, Rebalancing, Workload Balancing, Classical Approach, Tabu Search

ÖZ

MONTAJ HATLARINDA YENİDEN DENGELEME

Girit, Utku
Yüksek Lisans, Endüstri Mühendisliği
Tez Danışmanı: Prof. Dr. Meral Azizoğlu

Ağustos 2019, 70 sayfa

Montaj hattı dengeleme problemi en önemli ve iyi bilinen yöneylem araştırması problemlerinden biridir. Değişen koşullar nedeniyle mevcut hat dengesi uzun süre en verimli durumda kalamayabilir. Bir veya birden fazla iş istasyonunda yaşanabilecek bozulmalar, mevcut dengenin verimsizleşmesine hatta olursuz hale gelmesine neden olabilir. Bu çalışmada, montaj hattında yaşanan bozulmalardan sonra iş yükü dağılımını (verimlilik ölçümü) ve başlangıç ve son durumlarda aynı iş istasyonuna atanmayan işlerin toplam yer değiştirme miktarını (kararlılık ölçümü) göz önüne alarak montaj hattını yeniden dengelemeyi amaçlıyoruz.

Tanımlanan verimlilik ve kararlılık ölçütleri için tüm domine-edilmemiş objektif vektörleri, bunlara karşılık gelen verimli çözümler ile birlikte oluşturmaya çalışıyoruz. Bu amaçla iki algoritma geliştirdik: klasik yaklaşım ve tabu arama algoritması. Deneylerimiz, klasik yaklaşımın bir saatte 40'a kadar iş ve 7 iş istasyonu için tüm verimli domine-edilmemiş çözümleri bulduğunu, tabu arama algoritmasının küçük problemler için klasik yaklaşımın elde ettiği sonuçlara çok yakın sonuçlar elde ettiğini ve daha büyük problemleri çözebildiğini göstermiştir.

Anahtar Kelimeler: Montaj Hattı, Yeniden Dengeleme, İşyükü Dengeleme, Klasik Yaklaşım, Tabu Araması

To my family

ACKNOWLEDGEMENTS

First of all, I wish to express my deepest gratitude to my thesis supervisor Prof. Dr. Meral Azizoğlu for her great contribution to this thesis study. I would also like to thank to her for being such an understanding and supportive supervisor during our study.

I would like to thank my mother, father and brother for their endless encouragement. I could never complete my thesis studies without their love and support.

I would also like to thank to many other people for contributing to this study directly or indirectly. I would like to thank Gökçe Özkan, Ege Bozkurt, Orçun Koral, Yağmur Orkun and Utku Tarık Bilgiç for being the best friends one can have and always being there for me whenever I need their supports.

I would like to thank to my dear colleagues from ASELSAN and METU Industrial Engineering Department, especially to Dr. Aybeniz Yiğit, Hatice Doyduk, Umutcan Ayasandır, Sami Serkan Özarık and Erkut İşcanoğlu for being the most supportive colleagues.

Last but not least, I would like to express my love and deepest gratitude to my dear Defne Yaman for providing me with unending support and continuous encouragement through this study. This accomplishment would not have been possible without her invaluable support.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGEMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xiii

LIST OF FIGURES xiv

CHAPTERS

1. INTRODUCTION 1

2. LITERATURE REVIEW 5

  2.1. Literature on Assembly Line Balancing Problems (ALBP) 5

    2.1.1. Literature on Type-I & Type-II Problems 5

    2.1.2. Literature on Type-III Problems 6

  2.2. Literature on Assembly Line Rebalancing 8

    2.2.1. Literature on Rebalancing with Stochastic Task Times 8

    2.2.2. Literature on Rebalancing with Deterministic Task Times 9

3. PROBLEM DEFINITION AND MATHEMATICAL MODELING 11

  3.1. Problem Definition 11

  3.2. Mathematical Modeling 12

4. SOLUTION APPROACHES 17

  4.1. Non-Dominated Objective Vectors 17

  4.2. Extreme Non-Dominated Solutions 17

    4.2.1. The Non-Dominated Objective Vector with Smallest Efficiency Value . 18

4.2.2. The Non-Dominated Objective Vector with Smallest Stability Value 20

4.3. Generating All Non-Dominated Objective Vectors 26

4.3.1. Calculating Upper Bound on Cycle Time 26

4.3.2. Calculating Earliest and Latest Assignable Workstation Information 28

4.4. Modified Mathematical Model.. 29

5. MULTI OBJECTIVE TABU SEARCH APPROACH.................................. 39

6. COMPUTATIONAL EXPERIMENT 47

6.1. Data Generation 47

6.2. Performance Measures 49

6.3. Preliminary Experiments 52

6.3.1. Effect of Task Times 52

6.3.2. Effect of Earliest and Latest Possible Task Assignment Information...... 53

6.4. Main Experiments 54

6.4.1. Experiments with Classical Approach (CA) 54

6.4.2. Comparison Between Classical Approach and Tabu Search Algorithm.. 59

6.4.3. Testing Limits of Tabu Search Algorithm... 63

7. CONCLUSIONS 65

REFERENCES 67

LIST OF TABLES

TABLES

Table 4.1. Task times of the sample problem instance 33

Table 5.1. Tabu Tenure comparison results 41

Table 6.1. Configurations with K_N, K and disrupted workstations 48

Table 6.2. The performance of CA for U[1,10] and U[1,50] 52

Table 6.3. The performance of the Classical Approach with and without E_i and L_i . 54

Table 6.4. The average and maximum number of non-dominated objective vectors for K_N=4 & K=5 and K_N=4 & K=7 55

Table 6.5. The average and maximum number of non-dominated objective vectors for K_N=5 & K=6, K_N=5 & K=8, K_N=6 & K=7 and K_N=6 & K=9 55

Table 6.6. The average and maximum CPU times for K_N=4 & K=5 and K_N=4 & K=7 56

Table 6.7. The average and maximum CPU times (in seconds) for K_N=5 & K=6, K_N=5 & K=8, K_N=6 & K=7 and K_N=6 & K=9 56

Table 6.8. Comparison between Classical Approach and Tabu Search Algorithm for N=25 and N=30 60

Table 6.9. Comparison between Classical Approach and Tabu Search Algorithm for N=35 and N=40 61

Table 6.10. Average CPU times and number of non-dominated objective vectors found by Tabu Search Algorithm 63

LIST OF FIGURES

FIGURES

Figure 4.1. An initial configuration for the example instance.................................21

Figure 4.2. Sample instance precedence diagram...33

Figure 4.3. Initial configuration before the disruption...34

Figure 4.4. Efficient solutions obtained for Example 4.3 37

Figure 5.1. Illustration of short term, long term and weight loops..........................42

CHAPTER 1

INTRODUCTION

Assembly lines are manufacturing systems that reside a sequence of workstations performing repetitive tasks. These systems are flow-oriented and used for standardized high volume products. Assembly line balancing (ALB) problem is a significant problem for the management of the assembly lines. It concerns the assignment of the work elements, so called tasks, to the processing units, so called workstations, in the most efficient way. The tasks are assigned to the workstations by considering cycle time, precedence relationships and some line-specific constraints. The pioneering work on ALB problem was proposed by Salveson in 1955 and thereafter several variations of the problem are studied in the literature. The assembly lines still attract the researchers as the advances in the technology are continuously triggering many advanced line structures.

Several classifications for the ALB problem have been made in the literature. One important classification is due to the problem types where the types are defined as:

• Type-I Problem: Minimize the number of workstations for a specified cycle time value

• Type-II Problem: Minimize the cycle time for a specified number of workstations

• Type-III Problem: Minimize the workload variation between workstations

Rebalancing of the assembly lines is a newly studied problem area and despite its practical importance it is somewhat neglected. In this study, an assembly line rebalancing problem is considered where the task assignments to the workstations are already made. Most assembly lines are not balanced from scratch and but rather rebalanced considering the changes in the environment. Thus, rebalancing the line with respect to the initial configuration, but not from scratch, is more appropriate when any parameter alters, e.g., workstation shutdowns, cycle time changes, task time changes and new technological changes. These alterations are called disruptions and it may affect one or more workstations or the precedence structure so that the tasks may not be assigned to their original workstations in the new schedule.

In this study, the disruptions are assumed to occur on some specified workstations and they make the line balance infeasible. Once the disruption occurs, the disrupted workstations' tasks should be moved to other workstations. In doing so, the tasks on some other non-disrupted workstations may have to move to other workstations to improve the predetermined performance measure(s). If a task is assigned to a different workstation than its original, it is called disrupted.

After the disruption, in the rebalance, the trade-off between the stability measure and efficiency measure should be well established.

As a stability measure, we aim to reduce the total distance traveled by the disrupted tasks under the assumption that the workstations are equally spaced. Such a total distance traveled can be considered as the amount of movement made by the changes between the original schedule and reschedule. The setups are already settled for the current workstation and any movement to other workstation incurs cost that is a function of the travel length. The movements might be due to operators, machines, equipment and raw materials that are located in the original workstation.

As an efficiency measure, we aim to minimize the squared workloads of the workstations. Our aim is to achieve balanced work among all workstations by penalizing squared workloads. Higher workloads are penalized more for better smoothing. Uneven distribution of workloads would result unfair evaluation of the assembly workers that need to be avoided by the operation managers.

Our stability measure of total distance traveled and efficiency measure of total squared workload are conflicting as en route to keeping the total distance at reasonable level, one has to increase the workstation loads, thereby the total squared workload. Recognizing this fact, we find the non-dominated objective vectors for these two measures. We assume that the preference function of the decision maker is not known and he/she may make a preference according to the non-dominated objective function set.

In this study, we aim to generate the non-dominated objective function set together with an efficient solution representing each objective function value. We present two approaches: one for generating the exact non-dominated objective vector set, the other for the approximate set. The exact non-dominated objective vector set is generated via classical approach using the mixed integer linear programming models. The approximate set is generated in polynomial time by using tabu search algorithm. Our experiments show that the classical approach is helpful for small-sized problem instances and the tabu search algorithm finds satisfactory solutions for large-sized problem instances in reasonable time.

The rest of thesis is organized as follows: the related literature is reviewed in Chapter 2. In Chapter 3, we define the problem and give the mixed integer linear model. The classical approach is introduced in Chapter 4. In Chapter 5, we present our tabu search solution procedure. Chapter 6 discusses the results of our extensive experiments. Chapter 7 concludes the study and discusses potential related research areas.

CHAPTER 2

LITERATURE REVIEW

In this chapter, we review the assembly line balancing (ALB) studies and give related background for rebalancing assembly line problems with stochastic and deterministic task times.

2.1. Literature on Assembly Line Balancing Problems (ALBP)

As discussed in the introduction, there are three types of ALB problems: Type I, Type II and Type III. We base our literature review on this classification.

2.1.1. Literature on Type-I & Type-II Problems

Salveson (1955) published the first assembly line balancing problem paper. Since then, thanks to the ever increasing technological advances of the assembly lines, the related problems become very popular and challenging for the researchers and are still being studied in the literature.

Type-I problem and Type-II problem are shown as strongly NP-hard through the reduction to the well-known bin packing problem. Classical approaches are developed for small sized problem instances to find exact solutions and heuristic approaches are developed for bigger sized problem instances to find approximate solutions.

Several survey papers have been published for the assembly line balancing problems, some noteworthy of those are due to Baybars (1986), Boysen et al. (2007), Battaia and Dolgui (2013) and Sivasankaran and Shahabudeen (2014).

Baybars (1986) discussed some deterministic models for the exact solutions of Type-I problem and Type-II problem. He explained the ALB problem and its properties, gave the mathematical formulations for general Type-I problem and Type-II problem and reported a computational comparison.

Boysen et al. (2007) classified assembly line balancing problems for providing better communication between the researchers and the practitioners. They classified the assembly line literature according to the nature of the precedence graphs, workstation and line characteristics, objective functions and approaches used, i.e., exact or heuristic methods.

Battaia and Dolgui (2013) gave an extensive review, covering about 300 studies on the ALB problem. They basically focused on studies between years 2007 and 2012, and analyzed the recent research in several industrial contexts. They categorized the studies by the manufacturing environment, model size (single-model and multi-model), line layout (straight-type and U-type), task specifications (deterministic, assignment dependent, etc.), workstations and their attributes, constraints, objective functions, approaches used.

Sivasankaran and Shahabudeen (2014) also made a comprehensive review for the ALB literature. They classified the literature into eight categories by the model size, the line layout and the nature of task times. For each category and paper reviewed, they gave information about approach used (mathematical model, tabu search, genetic algorithm, etc.), problem type (Type-I or Type-II), goals (minimize cost, minimize idle time, etc.) and computational efficiency (solution time and closeness to the exact solutions).

2.1.2. Literature on Type-III Problems

Type-III problem literature is more recent and scarce compared to those of Type-I problem and Type-II problem. The existing studies have mentioned the importance of workload smoothing and proposed methods for solving the problem.

Rachamadugu and Talbot (1991) indicated that assigning nearly identical workloads to each workstation and to each person in the workstations is an important concern for many practitioners. Uneven workload assignments usually lead to a need for compensatory management actions like different payment systems. Smunt and Perkins (1985) showed that allocating the workload as even as possible, maximizes the production rate for low variance task times and long assembly lines. Groover (2013) argued that the unequal distribution of workload among the workstations increases the ergonomic risks and he also emphasized the importance of workload smoothing to reduce the ergonomic risks.

Esmaeilbeigi et al. (2015), Azizoglu and Imat (2018) and Finco et al. (2019) suggested exact solution approaches for the workload smoothing problem. Esmaeilbeigi et al. (2015) and Finco et al. (2019) used mixed integer linear programming formulations and Azizoğlu and İmat (2018) proposed a branch and bound algorithm along with some reduction mechanisms and bounds. They developed a simple heuristic rule to find an initial upper bound for their search algorithm.

Rachamadugu and Talbot (1991), Kim et al. (1998), Ponnambalam et al. (2000), Nearchou (2011) and Mozdgir et al. (2013) considered the approximate solution approaches for the workload smoothing problem. Rachamadugu and Talbot (1991) considered the total absolute deviation from target value as the objective function and proposed a heuristic procedure. Kim et al. (1998) considered the mean absolute deviation as the objective and proposed a heuristic based genetic algorithm. Nearchou (2011) suggested particle swarm optimization heuristic to minimize the cycle time and to maximize the workload smoothing of assembly lines. Mozdgir et al. (2013) considered the workload smoothness index as the objective and used an evolutionary algorithm, so called differential evolution algorithm. Ponnambalam et al. (2000) studied a multi-objective problem with different objectives like minimizing the number of workstations, maximizing the line efficiency and minimizing the deviation of the maximum workstation load from any other workstations.

We are not aware of any tabu search algorithm proposed for workload smoothing objectives.

2.2. Literature on Assembly Line Rebalancing

The research on the assembly line rebalancing is of recent origin and somewhat limited. We study the assembly line rebalancing literature in two categories considering the task time characteristics. As mentioned in Gamberini et al. (2006), rebalancing problems have multi-objective nature that trades off between the efficiency and stability objectives.

2.2.1. Literature on Rebalancing with Stochastic Task Times

Gamberini et al. (2006), Gamberini et al. (2009) and Celik et al. (2014) consider stochastic task times.

Gamberini et al. (2006) considered minimizing sum of the unit labor cost and expected incomplete unit costs and maximizing similarity between the initial and new configuration when any input parameter changes. They considered the stochastic version of the line balancing problems and proposed a heuristic solution approach that provides approximate non-dominated objective vectors set. They combined the heuristic solution approach of Kottas and Lau (1973) and a multi attribute decision making procedure of Hwang and Yoon (1981).

Gamberini et al. (2009) considered the same objectives with Gamberini et al. (2006); however, they proposed two different solution approaches to generate approximate non-dominated objective vector set: a genetic algorithm and a multiple single pass algorithm. Multiple single pass algorithm is an improved version of the one proposed in Gamberini et al. (2006). It combines four heuristic procedures, one of which is from the study of Gamberini et al. (2006). The computational results show that multiple single-pass algorithm dominates both genetic algorithm in Gamberini et al. (2009) and heuristic approach proposed in Gamberini et al. (2006).

Celik et al. (2014) proposed a heuristic (ant colony optimization algorithm) to rebalance U-type assembly lines with stochastic task times. Their objective is minimizing the total rebalancing cost that consists of task reallocation, workstation opening and closing costs.

2.2.2. Literature on Rebalancing with Deterministic Task Times

Grangeon et al. (2011), Yang et al. (2013), Faccio et al. (2015), Sanci and Azizoglu (2017) and Belassiria et al. (2018) studied the assembly line rebalancing problems with deterministic task times.

Grangeon et al. (2011) considered rebalancing problem with mixed models for an automotive firm. The line is balanced for each month and changes in production demand or line structure arise a need for rebalancing before new month starts. Rebalancing has three steps: obtaining a feasible solution, decreasing the number of workstations and smoothing the workload between workstations for the minimum number of workstations. They proposed three heuristic approaches one for each step. They improved the procedure in Boutevin (2003) by reducing the number of task assignment changes while smoothing the workstation loads.

Yang et al. (2013) studied a three-objective rebalancing problem with mixed-models. Their efficiency measures are minimizing the number of workstations and maintaining even workload distribution and the stability measure is minimizing the total processing time of reassigning tasks. They proposed a genetic algorithm and enhanced its efficiency by a local search.

Faccio et al. (2015) considered minimizing number of workstations as efficiency measure and minimizing the number of displaced tasks as stability measure. They developed an exact solution method that could solve the problem instances up to 34 tasks in less than 2 seconds. For bigger problem instances, they proposed decomposition approaches.

Sanci and Azizoglu (2017) considered a rebalancing problem with workstation breakdowns or shutdowns. Once the disruption occurs, they find all non-dominated objective vectors with respect to minimum cycle time and minimum number of tasks assigned to different workstations. They suggested two exact solution methods: a branch and bound algorithm and a mathematical model based procedure. Found that their branch and bound algorithm runs much faster than the mathematical model based procedure.

Belassiria et al. (2018) studied a real-life rebalancing problem for an automotive cable manufacturer. They proposed a genetic algorithm that is hybridized with a priority rule. They assumed that the task transposition costs are negligible, thereby considering no stability measure. However, they introduced assignment restrictions. By this way, some tasks with heavy machine requirement are assigned to the same workstation on the initial configuration. Their efficiency measures are maximizing the line efficiency expressed as cycle time and workload smoothing. They gave higher importance to minimize cycle time and they smoothed the workload for the minimum cycle time value.

The study by Sanci and Azizoglu (2017) is the most closely related one to ours. The similarities are due to the problem environment, like workstation breakdowns are considered as causes of disruption and classical (mathematical model based) approach to find the non-dominated objective set. We use different efficiency and stability measures and we propose different solution approach from Sanci and Azizoglu (2017).

CHAPTER 3

PROBLEM DEFINITION AND MATHEMATICAL MODELING

In this chapter, we give the problem definition and provide the mixed integer programming model.

3.1. Problem Definition

There are several assumptions that define the boundaries of our problem and these assumptions are listed below:

- Initial workload distribution is optimal according to our efficiency measure.
- There is a single product type.
- Paced transfers are considered.
- The system is static and deterministic; i.e., all parameters are certain and do not change.

In original configuration, N tasks are already assigned to K workstations. A disruption, that affects some specified workstations (K_D), occurs. K_N is the set of non-disrupted, i.e. non-affected, workstations. Our aim is to rebalance the line after the disruption considering the following measures.

i. Efficiency Measure: Sum of squared workloads (WL)

ii. Stability Measure: Total replacement cost (TR)

We assume the initial line balance optimizes our efficiency measure of minimizing the sum of squared workloads.

3.2. Mathematical Modeling

Using this problem structure, we defined our mathematical model with the following parameters:

- K = set of workstations in the initial balance
- K_D = closed workstation set
- K_N = set of non-disrupted workstations in the new balance
- t_i= processing time of task i for $i = 1,2, \dots . N$
- IP_i = immediate predecessor set of task i for $i = 1,2, \dots . N$

IP$_i$ values together give the precedence structure.

We explain our decision variable which explain the new balance as follows:

- $x_{ik} = \begin{cases} 1, \; if \; task \; i \; is \; in \; WS \; k \; in \; new \; balance \\ 0, \; otherwise \end{cases}$

for $i = 1, \; \dots, \; N$ and $k \in K_N$

The constraints are as stated below.

Each task must be assigned to one and only one workstation.

$$\sum_{k \in K_N} x_{ik} = 1 \qquad \text{for } i = 1,2, \dots . N$$

The precedence relations should be satisfied. Accordingly, if j ϵ IP$_i$, it should not be placed later than task i.

$$\sum_{k \in K_N} k * x_{jk} \leq \sum_{k \in K_N} k * x_{ik} \qquad \text{for } i = 1,2, \dots . N \; and \; j \in IP_i$$

The binary constraint is as stated below:

$x_{ik} = 0\ or\ 1$ for $i = 1,2, \ldots . N$ and $k \in K_N$

We next explain our performance measures

i. Efficiency Measure – Total Squared Workload

We let W_k denote the workload of workstation k.

Accordingly,

$W_k = \sum_{i=1}^{N} t_i * x_{ik}$ for $k \in K_N$

The sum of squared workloads is used instead of the sum of workloads at each workstation once we use sum of workloads at each workstation ($\sum_{k=1}^{K} W_k$), our objective considers cycle time only. However, our main purpose is assigning workload to the stations as uniform as possible. For example, between alternatives (10, 6, 4) and (10, 7, 3) which have equal cycle time, we want to select alternative 1. Therefore, the sum of square of workloads at workstations is used as it leads to a smooth distribution to the workstations.

Sum of square of workloads, WL is

$WL = \sum_{k \in K_N} {W_k}^2 = \sum_{k \in K_N} [\sum_{i=1}^{N} t_i * x_{ik}]^2$

$= \sum_{k \in K_N} [\sum_{i=1}^{N} {t_i}^2 * {x_{ik}}^2 + \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} t_i * t_j * x_{ik} * x_{jk}]$

$= \sum_{k \in K_N} [\sum_{i=1}^{N} {t_i}^2 * {x_{ik}}^2] + \sum_{k \in K_N} [\sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} t_i * t_j * x_{ik} * x_{jk}]$

$\sum_{k \in K_N} [\sum_{i=1}^{N} {t_i}^2 * {x_{ik}}^2]$ is equal to $\sum_{i=1}^{N} {t_i}^2$ since

$\sum_{k \in K_N} {x_{ik}}^2 = \sum_{k \in K_N} x_{ik} = 1$ as x_{ik}'s are binary and each task should be assigned.

This follows

$$WL = \sum_{i=1}^{N} t_i^2 + \sum_{k \in K_N} \left[\sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} t_i * t_j * x_{ik} * x_{jk} \right]$$

Minimizing WL is therefore equivalent to

Minimizing $\sum_{k \in K_N} \left[\sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} t_i * t_j * x_{ik} * x_{jk} \right]$ as $\sum_{i=1}^{N} t_i^2$ is constant.

Hence our efficiency measure WL is expressed as follows:

$$WL = \sum_{k \in K_N} \left[\sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} t_i * t_j * x_{ik} * x_{jk} \right]$$

Note that WL is non-linear and can be linearized through the use of new decision variable and constraint sets.

Therefore, we define y_{ijk} as follows:

$$y_{ijk} = \begin{cases} 1, \ if \ x_{ik} = x_{jk} = 1 \\ 0, \ otherwise \end{cases}$$

Accordingly,

$$WL = \sum_{k \in K_N} \left[\sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} t_i * t_j * y_{ijk} \right]$$

$y_{ijk} \geq x_{ik} + x_{jk} - 1$ for $i = 1,2,..N, \ j = 1,2,..N \ and \ k \in K_N$ (β)

$y_{ijk} \geq 0$ for $i = 1,2,..N, \ j = 1,2,..N \ and \ k \in K_N$

Constraint set (β) guarantees that $y_{ijk} \geq 1$ only when $x_{ik} = x_{jk} = 1$. $y_{ijk} \geq 1$ will be satisfied as strict equality due to the positive coefficient of y_{ijk}, which is (t_i * t_j) in the efficiency measure WL expression.

ii. Stability Measure – Total Displacement Cost

We define TR_i as the total displacement cost of task i.

Accordingly,

$TR_i = \sum_{k \in K_N} |k - k_i| * x_{ik}$ where k_i is the workstation of task i, in the initial configuration.

Total displacement cost, hereafter is defined as

TR = $\sum_{i=1}^{N} TR_i = \sum_{i=1}^{N} \sum_{k \in K_N} (|k - k_i| * x_{ik})$

For the sake of clarity, we recapitulate our model as follows:

Min WL = $\sum_{k \in K_N} \left[\sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} t_i * t_j * y_{ijk} \right]$ (O1)

Min TR = $\sum_{i=1}^{N} \sum_{k \in K_N} (|k - k_i| * x_{ik})$ (O2)

$\sum_{k \in K_N} x_{ik} = 1$ for $i = 1, \ldots, N$ (1)

$\sum_{k \in K_N} k * x_{jk} \leq \sum_{k \in K_N} k * x_{ik}$ for $i = 1,2,\ldots,N$ and $j \in IP_i$ (2)

$y_{ijk} \geq x_{ik} + x_{jk} - 1$ for $i, j = 1,2,..N$ and $k \in K_N$ (3)

$x_{ik} = 0$ or 1 for $i = 1, \ldots, N$ and $k \in K_N$ (4)

$y_{ijk} \geq 0$ for $i, j = 1,2,..N$ and $k \in K_N$ (5)

We refer the constraint sets (1) through (5) as $x \in X$, and our problem is expressed as

Min WL

Min TR

$x \in X$

CHAPTER 4

SOLUTION APPROACHES

This chapter first defines the non-dominated objective vectors and then discusses the extreme non-dominated objective vectors.

4.1. Non-Dominated Objective Vectors

For minimizing objectives WL and TR, a solution s in x ϵ X is called efficient if there is no other solution t in x ϵ X with $WL^t \leq WL^s$ and $TR^t \leq TR^s$ where strict inequality holds at least once. In other words, objective function values of solution s and t should be $(WL^s, TR^s) \leq (WL^t, TR^t)$: $WL^s \leq WL^t$ and $TR^s < TR^t$ or $WL^s < WL^t$ and $TR^s \leq TR^t$. (WL^s, TR^s) which is the resulting vector of solution s is said to be non-dominated. We can say that solution t is dominated by solution s and the objective vector of (WL^t, TR^t) is dominated by the objective vector of (WL^s, TR^s).

4.2. Extreme Non-Dominated Solutions

Once an efficient solution has the smallest objective function value for one objective, it is called extreme efficient solution. The objective function vectors corresponding to the extreme efficient solutions are referred to as extreme non-dominated objective vectors. We now discuss the generation of two extreme non-dominated objective vectors and corresponding extreme efficient solutions.

4.2.1. The Non-Dominated Objective Vector with Smallest Efficiency Value

Assume the following problem

(M1) Min $\sum_{k=1}^{K} {W_k}^2$ = WL

Subject to $x \in X$

WL^*, the optimal objective function value, is a lower bound of all efficient solutions with respect to the total squared workloads. On the other hand, any solution having a total load of WL^* may not be efficient since another optimal solution with smaller stability value may exist.

Among the optimal assignments the one with the smallest stability value can be found through the solution of the following problem:

(M2) Min TR

Subject to $x \in X$

$WL = WL^*$

Therefore, two stages are needed in order to find the efficient solution with WL value of WL*

1. Solve Minimize WL subject to $x \in X$

   Let WL^* be the optimal objective function value.

2. Solve Minimize TR subject to $x \in X$ and $WL = WL^*$

   Let TR^* be the optimal objective function value.

The objective function is modified as WL + ϵ_{TR} * TR for a sufficiently small value of ϵ_{TR}, instead of solving the problem in two stages. In other words, the following problem can be solved to get (WL^*, TR^*) solution.

Min $\quad$ WL + ϵ_{TR} * TR

s.t. $\quad$ $x \in X$

where $\quad$ $0 < \epsilon_{TR} << 1$

ϵ_{TR} should be sufficiently small so that the total squared workload does not increase even one unit for the highest reduction of the TR value. According to that,

$WL^* + \epsilon_{TR} * TR_{max} \leq WL^* + 1 + \epsilon_{TR} * TR_{min}$

where

TR_{min} = minimum possible value of TR

= total number of tasks in the disrupted workstations

TR_{min} is the minimum movement that would occur when all tasks of the disrupted workstations are moved to the previous or next workstation.

TR_{max} = maximum possible value of TR

$= \sum_{i=1}^{N} \max\{k_i - 1\,,\ K - k_i\}$

where k_i is the original workstation of task i

max $\{k_i - 1\,,\ K - k_i\}$ is the maximum movement due to task i either to the first workstation or last workstation, whichever is further.

Therefore;

$WL^* + \epsilon_{TR} * TR_{max} \leq WL^* + 1 + \epsilon_{TR} * TR_{min}$

$\epsilon_{TR} * (TR_{max} - TR_{min}) \leq 1$

$\epsilon_{TR} \leq \frac{1}{TR_{max} - TR_{min}}$

In our experiments, we set

$\epsilon_{TR} = \frac{1}{TR_{max} - TR_{min}+1}$.

We also use earliest and latest possible workstation information while calculating TR_{max} thereby ϵ_{TR}.

We set $TR_{max} = \sum_{i=1}^{N} \max \{k_i - E_i , L_i - k_i\}$

where E_i is earliest assignable workstation for task i and L_i is latest assignable workstation for task i. E_i and L_i computations are discussed in Section 4.3.2.

4.2.2. The Non-Dominated Objective Vector with Smallest Stability Value

Consider the following problem

Min	$\sum_{k=1}^{K} W_k{}^2$
Subject to	$x \in X$
	$TR = TR^*$

where TR^* is the minimum displacement amount of tasks in the efficient set.

TR^* serves as a lower bound for the minimum displacement amount of tasks of the all efficient solutions.

TR* can be found by considering only the disrupted workstations. The solution is moving the tasks on the disrupted workstations to one of their closest workstations. For each task, there are at most two closest workstations, one to the left and one to the right.

As the minimum total disruption value is only related with the disrupted tasks, the allocation will be among the disrupted tasks, hence the problem size will be reduced. Moreover, we reduce the number of workstations and consider only the workstations that are closest to the disrupted workstations.

The following example illustrates the minimum displacement of tasks.

Example 4.1: We have 21 tasks, 4 workstations and initial configuration is given as follows:

WS 1	**WS 2**	**WS 3**	**WS 4**
1,3,4,21	5,6,7,8,9,11,12	10,13,14,15,16,17	2,18,19,20

Figure 4.1. An initial configuration for the example instance

Suppose workstation 3 is disrupted.

To find minimum total squared load with minimum disrupted amount, we consider disrupted tasks 10,13,14,15,16 and 17. These tasks should be assigned to either left (WS2) or right (WS4) workstations by considering total squared load.

Workstation assignment of tasks in WS1, WS2 and WS4 will not change after the disruption since we want to minimize total disrupted amount.

Mathematical model for minimizing smallest stability value is given below:

Parameters

N' = set of tasks on disrupted workstations

K' = set of workstations that are closest to the disrupted workstations

L_k = Workload of the workstation k in initial configuration for $k \in K'$

t_i = completion time of task i for $i \in N'$

IP_i = immediate predecessor cluster of task i for $i \in N'$

K_j = workstation index of non-disrupted task j, s.t. $j \in IP_i$ or $i \in IP_j$ for any disrupted task i

In our example, N'= {10, 13, 14, 15, 16, 17} and K'= {WS2, WS4}.

Decision Variable

$$x_{ik} = \begin{cases} 1, & \text{if task } i \text{ is assigned to WS } k \text{ in new configuration} \\ 0, & \text{otherwise} \end{cases}$$

for $i \in N'$ and $k \in K'$

Objective Function

Minimize $\sum_{k \in K'} {W_k}^2 = \sum_{k \in K'} (L_k + \sum_{i \in N'} t_i * x_{ik})^2$

$= \sum_{k \in K'} {L_k}^2 + 2 * \sum_{k \in K'} (L_k * \sum_{i \in N'} t_i * x_{ik}) + \sum_{k \in K'} (\sum_{i \in N'} t_i * x_{ik})^2$

$\sum_{k \in K'} {L_k}^2$ is constant and can be removed from the objective function.

$\sum_{k \in K'} (\sum_{i \in N'} t_i * x_{ik})^2$ is non-linear and can be linearized through the use of new decision variable and constraint sets using the linearization method used in Chapter 3.

Therefore, our minimum stability value model becomes:

Minimize $2 * \sum_{k\in K'}(L_k * \sum_{i\in N'} t_i * x_{ik}) + \sum_{k\in K'}\sum_{i\in N'}\sum_{j\in N', j\neq i} t_i * t_j * y_{ijk}$

subject to $\sum_{k\in K'} x_{ik} = 1$ for $\forall i \in N'$

$\sum_{k\in K'} k * x_{ik} \leq \sum_{k\in K'} k * x_{jk}$ for $\forall i,j \in N'$ and $\forall i \in IP_j$

$y_{ijk} \geq x_{ik} + x_{jk} - 1$ for $\forall i,j \in N'$ and $\forall k \in K'$

$\sum_{k\in K'} k * x_{ik} \leq K_j$ for $\forall i \in IP_j$ and $\forall j \notin N'$

$\sum_{k\in K'} k * x_{ik} \geq K_j$ for $\forall j \in IP_i$ and $\forall j \notin N'$

$x_{ik} = 0\ or\ 1$ for $\forall i \in N'$ and $\forall k \in K'$

$y_{ijk} \geq 0$ for $\forall i,j \in N'\ and\ \forall k \in K'$

When there is a single workstation disrupted, the problem size is further reduced and a more efficient formulation is provided. For a single disruption, two cases exist.

Case I. There is a single closest workstation, say u. (Disruption occurs in first or last workstations)

In the single closest workstation problem, all tasks of the disrupted workstation are moved to workstation u.

$WL_u^* = L_u + \sum_{i\in N'} t_i$

$WL_k^* = L_k$ for all $k \neq u$

TR* = # of tasks in N’

Case II. There are two closest workstations, say u and v. (Disruption occurs in one of the workstation 2,3,…,K-1)

The associated problem is formulated as:

Min $\quad WL_u^2 + WL_v^2$

Subject to $\quad WL_u = L_u + \sum_{i \in d} t_i * x_{iu}$

$WL_v = L_v + \sum_{i \in d} t_i * (1 - x_{iu})$

$x_{iu} \geq x_{ju}$ $\quad$ for $(i,j) \in d$ and $i \in IP_j$

$x_{iu} = \{0,1\}$ $\quad$ for $\forall i \in d$

$u<d<v$ where d is disrupted workstation and u is previous and v is following workstation of d.

Note that there are |D| binary variables.

Moreover, we do not need to linearize the objective function as it reduces to the even load distribution, C_{max}, problem.

We state this result formally in theorem below.

$Z_1 = WL_u^2 + WL_v^2$

$Z_2 = \text{Max}\ \{WL_u, WL_v\}$

Theorem: Minimizing Z_1 is equivalent to minimizing Z_2.

Proof: Assume $Z_2^* = WL_u = \text{Max}\ \{WL_u, WL_v\}$

We will show that any increase above Z_2^* value increases Z_1^* value.

Assume Z_2^* is increased by k units such that $WL_u = WL_u^* + k$ and $WL_v = WL_v^* - k$.

Accordingly,

Z_l will be affected by

$Z_l = (WL_u^* + k)^2 + (WL_v^* - k)^2$

$\quad = (WL_u^*)^2 + (WL_v^*)^2 + 2 * k^2 + 2 * k * (WL_u^* - WL_v^*)$

$Z_{new} = Z_{old}^* + 2 * k^2 + 2 * k * (WL_u^* - WL_v^*)$

$k > 0$ and $WL_u^* \geq WL_v^*$

Therefore, $Z_{new} > Z_{old}$.

This follows that an increase of Z_1^* value increases the Z_2^* value.

Hence, minimizing Z_1 is equivalent to minimizing Z_2, i.e., we can use minimizing cycle time as objective function without using any linearization method.

Suppose workstation r is disrupted. Updated mathematical model is given below.

Parameters

r = disrupted workstation

N' = set of tasks on r

L_k = load of workstation k in the initial configuration

A lower bound on C_{max} is:

$LB(C_{max}) = \frac{L_{r-1} + L_r + L_{r+1}}{2}$ (distribution minimizes the maximum load)

Decision Variables

$$x_i = \begin{cases} 1, \ if\ task\ i\ is\ assigned\ to\ r-1 \\ 0, \ if\ task\ i\ is\ assigned\ to\ r+1 \end{cases}$$

C_{max} = Cycle time

Mathematical Model

Minimize C_{max}

$C_{max} \geq L_{r-1} + \sum_{i \in N'} t_i * x_i$

$C_{max} \geq L_{r+1} + \sum_{i \in N'} t_i * (1 - x_i)$

$x_i \geq x_j$ for $i \in IP_j$ and $\forall i, j \in N'$

$x_i = \{0,1\}$ for $\forall i \in N'$

$C_{max} \geq LB(C_{max})$

4.3. Generating All Non-Dominated Objective Vectors

In this chapter, we derive an upper bound on the cycle time and discuss its use for the workstation indices of the tasks. The ranges on the workstation indices of each task are used in our mathematical models.

4.3.1. Calculating Upper Bound on Cycle Time

Recall that the efficient solution with smallest stability value provides an upper bound on the WL values of the efficient solutions.

We let this WL value as UB(WL) and use it to obtain an upper bound on the maximum load of any efficient solution.

$WL^* = CT^2 + \sum_{k \neq k_b} W_k{}^2$

where k_b is the index of the bottleneck workstation, i.e., the workstation that defines CT value.

One lower bound on the $\sum_{k \neq k_b} {W_k}^2$ value, LB(K-1) distributes the total workload ($\sum_{i=1}^{N} t_i - CT$) among K-1 workstations, leading to the following expression:

$$LB(K-1) = (K-1) * \left(\frac{\sum_{i\varepsilon I} t_i - CT}{K-1}\right)^2$$

$$= \frac{(\sum_{i\varepsilon I} t_i - CT)^2}{K-1}$$

This follows

$UB(WL) \geq CT^2 + LB(K-1)$

$UB(WL) \geq CT^2 + \frac{(\sum_{i\varepsilon I} t_i - CT)^2}{K-1}$

$CT^2 * (K-1) + (\sum_{i\varepsilon\, I} t_i - CT)^2 - UB(WL) * (K-1) \leq 0$

$K * CT^2 - 2 * \sum_{i\varepsilon\, I} t_i * CT + (\sum_{i\varepsilon\, I} t_i)^2 - (K-1) * UB(WL) \leq 0$

This is a second order polynomial function of CT ($aX^2 + bX + c$) where

$a = K > 0$

$b = -2 * \sum_{i\varepsilon\, I} t_i < 0$

$c = (\sum_{i\varepsilon\, I} t_i)^2 - (K-1) * UB(WL)$

From calculus, we know that the upper root of polynomial function above is:

$$CT_2 = \frac{-b + \sqrt{b^2 - 4*a*c}}{2*a}$$

$$= \frac{2 * \sum_{i\varepsilon I} t_i + \sqrt{4 * (\sum_{i\varepsilon I} t_i)^2 - 4*K*[(\sum_{i\varepsilon I} t_i)^2 - (K-1) * UB(WL)]}}{2*K}$$

$$= \frac{\sum_{i\varepsilon I} t_i + \sqrt{(\sum_{i\varepsilon I} t_i)^2 - K * (\sum_{i\varepsilon I} t_i)^2 - K * (K-1) * UB(WL)}}{K}$$

This follows CT_2 is an upper bound on the CT values of the efficient solutions.

Example 4.2: Calculating an upper bound on cycle time

Let $\sum_{i\varepsilon\, I} t_i$ = 324, K=10 and UB(WL)=10642.

Then,

a = K =10

b = -2 * $\sum_{i\varepsilon\, I} t_i$ = -648

c = $(\sum_{i\varepsilon\, I} t_i)^2$ – (K-1) * UB(WL) = 9198

CT_2 = UB(CT) = $\frac{324+\sqrt{324^2-10 * 324^2+9*10*10624}}{10} \approx 44.$

CT_2 value is used in Earliest (E_i) and latest (L_i) computations. E_i and L_i are not only used for finding TR_{max} and ϵ_{TR} but also used in reducing the number of decision variables and constraints of mathematical model.

4.3.2. Calculating Earliest and Latest Assignable Workstation Information

The cycle time upper bound value can be used to calculate the earliest (E_i) and latest (L_i) assignable workstation information for each task i.

E_i= the earliest assignable workstation for task i

$= \left\lceil \frac{t_i+\sum_{j\in P_i} t_j}{CT} \right\rceil$ where P_i is the predecessors set of task i

L_i= the latest assignable workstation for task i

$= K - \left\lceil \frac{t_i+\sum_{j\in S_i} t_j}{CT} \right\rceil + 1$ where S_i is the successors set of task i

where

$$CT = \begin{cases} CTnew, \ if\ cycle\ time\ of\ new\ situation\ is\ known\ or\ decided \\ UB_{CT},\ otherwise \end{cases}$$

While finding E_i and L_i, if the cycle time of the assembly line in new situation (CT_{new}) is known or decided, we can use this value as CT. Otherwise, upper bound for cycle time (UB_{CT}) need to be calculated since cycle time of the initial configuration is not an upper bound for new configuration.

E_i and L_i values are also used for TR_{max} and ϵ_{TR} by using TR_{max}.

Recall that, we have found TR_{max} using the following expression as follows:

$TR_{max} = \sum_{i=1}^{N} \max\{k_i - 1\ ,\ K - k_i\}$

We strengthen TR_{max} using the earliest and latest workstation information as follows.

$TR_{max} = \sum_{i=1}^{N} \max\{k_i - E_i\ ,\ L_i - k_i\}$.

4.4. Modified Mathematical Model

Recall that our mathematical model was stated as follows:

Min WL

Min TR

Subject to $x \in X$

Constraint set (1) through (5) shown as $x \in X$.

In Chapter 4.2, we showed that two objective functions can be converted into a one objective function via ε_{TR} value.

Min WL + $\varepsilon_{TR} TR$

Subject to x ϵ X

where $\varepsilon_{TR} = \frac{1}{TR_{max} - TR_{min} + 1}$

In this section, we incorporate the earliest and latest workstation information of the tasks to the mathematical model en route to reducing the size of our mathematical model. We also add upper bound on cycle time to our model.

Our modified mathematical model that uses E_i and L_i information as bounds on the workstation indices and upper bound on cycle time is given below.

Minimize

$$\sum_{k=E_i}^{L_i} \left[\sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} t_i * t_j * x_{ik} * x_{jk} \right] + \varepsilon_{TR} * \sum_{i=1}^{N} \sum_{k \in K_N} (|k - k_i| * x_{ik})$$

Subject to

$$\sum_{k=E_i}^{L_i} x_{ik} = 1 \qquad \text{for } i = 1,2,\dots,N \qquad (1)$$

$$\sum_{k=E_i}^{L_i} k * x_{jk} \leq \sum_{k=E_i}^{L_i} k * x_{ik} \qquad \text{for } i,j = 1,2,\dots,N \text{ and } j \in IP_i \qquad (2)$$

$$x_{ik} + x_{jk} - y_{ijk} \leq 1 \qquad \text{for } i,j = 1,2,\dots,N \text{ and } k = E_i,\dots,L_i \qquad (3)$$

$$x_{ik} = 0 \text{ or } 1 \qquad \text{for } i = 1,2,\dots,N \text{ and } k = E_i,\dots,L_i \qquad (4)$$

$$y_{ijk} \geq 0 \qquad \text{for } i,j = 1,2,\dots,N \text{ and } k = E_i,\dots,L_i \qquad (5)$$

$$\sum_{i=1}^{N} t_i * x_{ik} \leq CT \qquad \text{for } k \in K_N \qquad (6)$$

After that point, we refer the constraint sets (1) through (6) as $x \in X'$, and our problem is expressed as

Min WL + $\varepsilon_{TR} TR$

$x \in X'$

E_i and L_i have significant roles in decreasing the number of decision variables and constraints. For example; for assembly line dataset which have 32 tasks and 7 workstations 18% improvement in the solution time (average of 10 different run) and 35% reduction in the number of decision variables (average of 10 different run) are provided. Therefore, we decide to use E_i and L_i to our model. More experiments about E_i and L_i will be shown in Chapter 6.

Haimes et al. (1971) show that the following constrained problem gives an efficient solution.

Min WL + $\varepsilon_{TR} * TR$

Subject to $x \in X'$

TR $\leq t$

where t is between TR_{min} and TR_{max} and $\varepsilon_{TR} = \frac{1}{TR_{max} - TR_{min} + 1}$.

Using this result, Procedure 1 finds all non-dominated objective vectors by varying the TR value systematically according to the mathematical model above.

Procedure 4.1. Finding the Non-Dominated Objective Vectors

<u>*Step 0:*</u>

Calculate TR_{min} and UB_{CT}

Use UB_{CT} in order to calculate E_i and L_i

Use E_i and L_i in order to calculate TR_{max}

Use TR_{min} and TR_{max} in order to calculate $\varepsilon_{TR} = \frac{1}{TR_{max} - TR_{min} + 1}$

<u>*Step 1:*</u> Solve the problem below:

Min $WL + \varepsilon_{TR} TR$

Subject to $x \in X'$,

$TR \leq TR_{max}$

Let the optimal solution be (WL^*, TR^*)

<u>*Step 2:*</u>

If $TR^* = TR_{min}$, STOP.

If $TR^* > TR_{min}$, Update $TR_{max} = TR^* - 1$ and $\varepsilon_{TR} = \frac{1}{TR_{max} - TR_{min} + 1}$

Go to Step 1.

Each iteration of Procedure 1 returns a non-dominated objective vector. When Procedure 4.1 is completed, all non-dominated objective vectors are generated.

Example 4.3: Using Procedure 4.1 on an assembly line sample

Consider a precedence diagram from Rosenberg and Ziegler (1992). Precedence diagram, task times and initial configuration are given below in Figure 4.2, Table 4.1 and Figure 4.3, respectively.

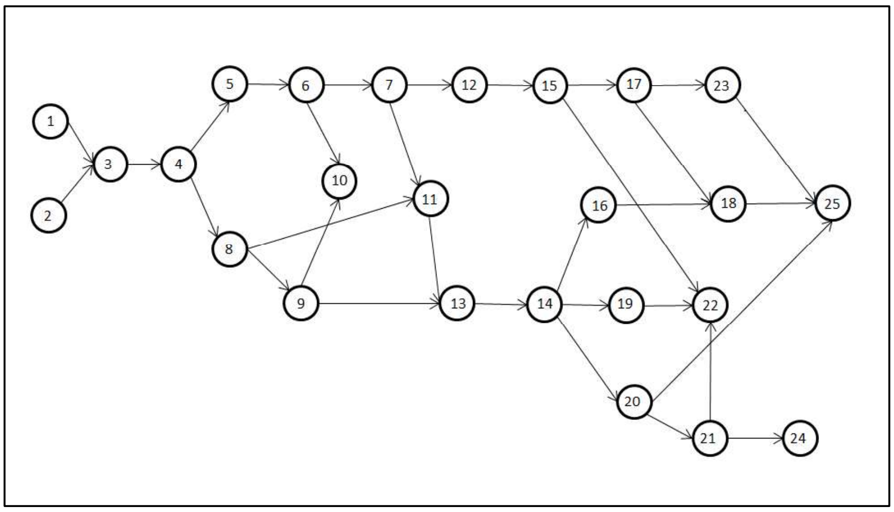

Figure 4.2. Sample instance precedence diagram

Table 4.1. *Task times of the sample problem instance*

Task	**t_i**	**Task**	**t_i**	**Task**	**t_i**	**Task**	**t_i**	**Task**	**t_i**
1	34	**6**	11	**11**	16	**16**	21	**21**	5
2	28	**7**	15	**12**	42	**17**	47	**22**	21
3	36	**8**	3	**13**	8	**18**	24	**23**	47
4	42	**9**	5	**14**	11	**19**	9	**24**	5
5	29	**10**	46	**15**	17	**20**	50	**25**	20

WS 1	**WS 2**	**WS 3**	**WS 4**	**WS 5**
1,2,3,4	5,6,7,8	9,11,12,13, 14,15,20	16,17,21	10,18,19,22, 23,24,25

Figure 4.3. Initial configuration before the disruption

Suppose WS3 is disrupted. Procedure 4.1 is implemented as follows:

<u>*Step 0:*</u> Initialization

Calculate TR_{min} and UB_{CT}

TR_{min} = # of tasks in disrupted WS = 7.

UB_{CT} = 100 (according to Section 4.3.1)

According to the formula given in Section 4.3.2, E_i and L_i values are calculated and added to the decision variables and constraints.

TR_{max} is calculated as 56 by using E_i and L_i values of all tasks according to formula given in Section 4.3.2.

$\varepsilon_{TR} = \frac{1}{56-7+1} = 0.02.$

<u>*Step 1:*</u> The problem is solved as

Min $\quad WL + 0.02 * TR$

Subject to $\quad x \in X'$,

$\quad TR \leq 56$

Corresponding optimal solution is (WL*, TR*) = (82,746, 12)

Step 2: Updating TR_{max} and ε_{TR}.

TR* $> TR_{min}$ (12 > 7).

TR_{max} is updated as 12 – 1 = 11 and $\varepsilon_{TR} = \frac{1}{11 - 7 + 1} = 0.2$.

Go to Step 1.

Step 1: The problem is solved as

Min $\quad WL + 0.2 * TR$

Subject to $\quad x \in X'$,

$\quad$ TR ≤ 11

Corresponding optimal solution is (WL*, TR*) = (82,950, 10)

Step 2: Updating TR_{max} and ε_{TR}.

TR* $> TR_{min}$ (10 > 7).

TR_{max} is updated as 10 – 1 = 9 and $\varepsilon_{TR} = \frac{1}{9 - 7 + 1} = 0.33$.

Go to Step 1.

Step 1: The problem is solved as

Min $WL + 0.33 * TR$

Subject to $x \in X'$,

TR$\leq$ 9

Corresponding optimal solution is (WL*, TR*) = (83,444, 8)

Step 2: Updating TR_{max} and ε_{TR}.

$TR^* > TR_{min}$ (8 > 7).

TR_{max} is updated as 8 – 1 = 7 and $\varepsilon_{TR} = \frac{1}{7-7+1} = 1$.

Go to Step 1.

Step 1: The problem is solved as

Min $WL + TR$

Subject to $x \in X'$,

TR$\leq$ 7

Corresponding optimal solution is (WL*, TR*) = (85,634, 7).

Step 2: Updating TR_{max} and ε_{TR}.

$TR^* = TR_{min}$. Therefore, procedure is finished.

In four iterations, our procedure gives all non-dominated objective vectors that are plotted in Figure 4.4.

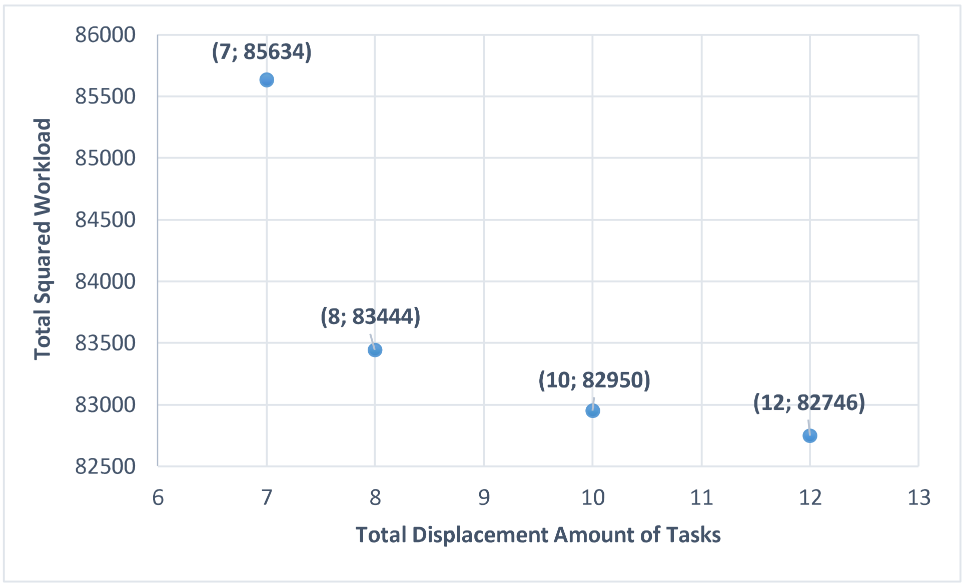

Figure 4.4. Efficient solutions obtained for Example 4.3

With considering the second objective: displacement amount, alternative solutions provided to the decision maker and DM can select any solution to implement according to importance of the objectives. For instance, if DM thinks that other workstations will be closed or opened in the short run, transporting more machines to the other workstation makes no sense, DM can implement the alternative 1 (TR=7, WL=85,634). Also, DM might select alternative 2 (TR=8, WL=83,444) since increasing displacement amount more does not provide sufficient decrease in workload distribution. Finally, if new balance will continue for a long time, DM can select the alternative 4 (TR=12, WL=82,746) which provides best workload smoothing with minimum possible displacement amount of tasks.

CHAPTER 5

MULTI OBJECTIVE TABU SEARCH APPROACH

Our preliminary computational experiment has revealed that classical approach handles the instances with up to 6 workstations when there are 35 tasks and 4 workstations when there are 40 tasks.

To find solutions to large sized real life instances we propose a multi-objective tabu search (MOTS) algorithm. Our motivation to use MOTS algorithm came from the study by Chiang (1988) who reported well performance of Tabu Search for single objective ALB problems. As far as we know, we propose the first tabu search algorithm to the multi-objective assembly line rebalancing problems.

As stated in Glover and Laguna (1998), both attributive and explicit memories are used in Tabu Search. Information related with solution attributes that change while moving from one solution to other, i.e., basic moves are recorded in attribute memory. On the other hand, elite solutions visited which include complete solutions are kept in explicit memory. Elite solutions will be discussed later.

Our algorithm starts with a feasible solution. This feasible solution is found by assigning tasks in the disrupted workstations to the closest workstations considering feasibility due to precedence relationships.

We use insertion method as a solution attribute, where one task is taken from a workstation and is assigned to another if the resulting improvement is the best for our objective function (aspiration criteria) and the insertion is not tabu active. Also, we do not allow infeasible insertions throughout the search process since their repairs may be hard and time consuming.

Glover and Laguna (1998) state that, in Tabu Search, short term memory is called as recency-based memory and it keeps track of solution attributes that have changed during the recent past. We use recency-based memory in order to track recently made insertions, i.e., tabu active solution attributes.

In our algorithm, recently used insertions (solution attributes) are recorded as tabu active (added to the short term memory) for tabu tenure period in order to prevent cycling. For tabu tenure period, tabu active insertions cannot be used in order to reach a new solution. When an insertion is no longer tabu-active, it is deleted from short term memory and could be used in future moves. The duration is usually measured by the number of iterations.

We set the tabu tenure to N/4, N/2, N and 2N tasks and make a small experiment with N=25, 30, 35 and 40. We use the following two measures for evaluation and the results are reported in Table 5.1.

M1. Number of times the exact non-dominated objective vectors are found

M2. Number of times the best objective vectors – compared to other tabu tenure values – are found

Table 5.1. *Tabu Tenure comparison results*

		Tabu Tenure Alternatives			
		N/4	**N/2**	**N**	**2N**
N=25 K_N=5 K=6	M1	7	7	8	4
	M2	7	7	8	4
N=30 K_N=6 K=9	M1	5	5	7	1
	M2	8	6	8	1
N=35 K_N=6 K=7	M1	4	6	6	1
	M2	4	6	9	1
N=40 K_N=4 K=7	M1	4	5	5	1
	M2	4	5	7	1

Note from the above table that, the performance is the best when tabu tenure is set N (number of tasks) and worst when it is 2N. Based on those results, N is set as tabu tenure.

The set of known and not-yet-dominated objective vectors are kept in an elite solution set (explicit memory). In elite solution set, we record assigned workstation assignment information for each task, total squared workload and total displacement amount for each solution and objective function value. The new solution is added to the elite set if that solution is not dominated by any other solution in elite solution set. Also, if this solution dominates any other solution from the elite set, the dominated solution is deleted. In our implementation, we set the range of elite solution set as N.

Our solution approach divides the solution space to smaller search areas and tries to find non-dominated solutions there. In our algorithm, we use three steps: short term loop, long term loop and weight loop.

The following figure illustrates our short term, long term and weight loops.

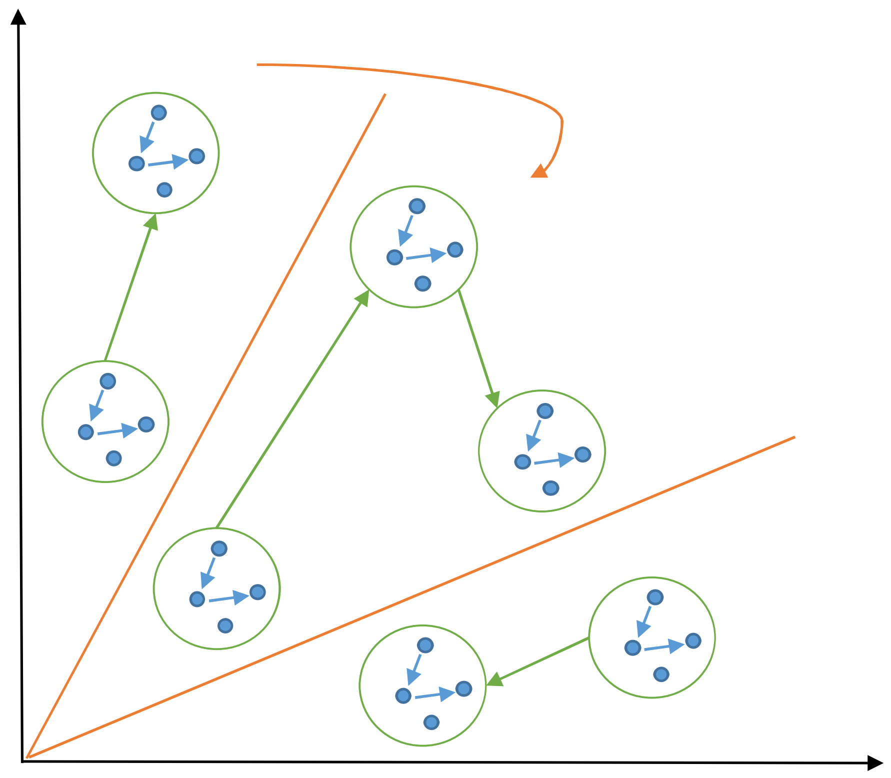

Figure 5.1. Illustration of short term, long term and weight loops

In Figure 5.1, blue arrow represents a short term iteration, green arrow represents a long term iteration and orange arrow represents weight change.

Short Term Loop

Short term loop is detailed search to find best solutions in a smaller (specific) area. In order to find best (non-dominated) solutions, we use intensification in short term iterations. Glover and Laguna (1998) state that intensification and diversification strategies are two very important parts of tabu search. In our algorithm, intensification is done in the short term loop while diversification is done in the long term loop. We use two intensification strategies in the short term loop: choosing best improving moves in each short term iterations and restart with historically found good assignment if an improvement in objective function value is not occur for stopping counter period.

In our problem, an insertion which provides best improvement in our objective function is called as good attribute. In each iteration, firstly precedence relationship feasibility check is done. We only consider insertions that are feasible according to precedence relationships. Between feasible insertions, an insertion which provides maximum decrease in our objective function is selected. If that insertion is tabu-active, tabu status of second best is checked. This procedure is continued until there is a not tabu-active insertion is found and that insertion is made in that iteration. After insertion, if new solution is not dominated by any other solution from elite solution set, it is added to elite solutions and if new solution dominates any other recorded solution, that dominated solution is deleted from elite solution set. Search is continued until it reaches to short term iteration limit or stopping counter limit.

Intensification strategies encourage historically found good basic moves. For this purpose, frequency (long term) based memory is used which demonstrates how often an insertion used during the search. If an improvement in objective function value does not occur for stopping counter period, we restart detailed search in that specific area with a new starting point includes historically found good insertions. For defining a new start point, each task is assigned to the most frequently assigned workstation during the previous search. If that assignment is not feasible for any task, second most used assignment for that task is checked. This procedure is continued until a feasible assignment for all tasks is found. With new starting point, the detailed search is repeated until it reaches to short term iteration limit or stopping counter limit.

In our solution approach, we select 10*N as stopping counter limit and 30*N as short term iteration limit according to our observations and preliminary experiments.

<u>Long Term Loop</u>

When a detailed search in a small area is complete, long term loop is used in order to visit new areas to see more solution alternatives. For this purpose, frequency (long term) based memory is used which demonstrates how often was a solution element considered and used during the search. By using frequency based memory, we try to escape from getting stuck in local optima solutions. We record the frequency ratios of insertions in frequency ratio matrix (Number of tasks, Number of workstations) and that information is used for regional differentiation i.e., exploration and for finding a new start point.

In order to perform diversification, we use the solutions which are evaluated but not yet visited. In each long term loop, we store the number of times an attribute is visited, i.e., attribute used in that short loop iteration to find a new solution, the number of times each attribute is evaluated. By using this information, we obtain frequency ratio (the number of times each attribute visited/evaluated). Low frequency ratio indicates that the attribute is seldom visited although it is frequently evaluated. Hence using this attribute in our solution, we expect to diversify the search area.

For finding a new starting point, the tasks are assigned to the workstations according to lowest frequency ratio values and precedence feasibility. After that, a detailed search for new region is started, i.e., methods in short term loop search is repeated.

In our solution approach, we select 2*N as stopping counter limit and 3*N as long term iteration limit according to our observations and preliminary experiments. If an improvement in objective function value is not occur for stopping counter period, we finish that long term loop and algorithm will continue with new weight loop.

Weight Loop

Our problem has two objectives: balancing workload between workstations and minimizing number of task displacement. In the classical approach, we use weighted sum of two objectives, where the weight of first objective is set to one and the weight of second objective is set to very small positive number ε. In tabu search algorithm, we decide to use w1 and w2 as the normalized weights of the objectives.

Objective function: w1 * WL + w2 * TR

Our objectives have different scales, i.e., WL values are significantly higher than TR values. Therefore, to see the effect of second objective better and find more non-dominated objective solutions, we use normalized weights. Our algorithm starts with (w1, w2) = (1, 0). We change the weights used in objective function to reach different areas in solution space according to the importance of the objectives. To do that, for each weight loop, the weight of the first objective is progressively decreases while the second objective's weight progressively increases.

In our solution approach, we use 5 weight loops. Therefore, we reach all solution space from (1 * WL + 0 * TR) to (0 * WL + 1 * TR).

The pseudo code for the Modified Tabu Search algorithm is provided below.

Procedure 5.1 Pseudo Code of Proposed Tabu Search Algorithm

```
Find an initial solution $x_0$ set $x_{now}=x_{best}=x_0$
Initialize frequency base memories, recency based memories and elite solutions
Repeat (Weight loop)
Decrease the normalized weight of first objective, increase the normalized weight of second objective
    Repeat (Long term loop)
    Find a feasible solution consisting the solution elements evaluated but not visited in previous iteration using frequency based memory
        Repeat (Short term loop)
        Choose feasible $x_{next}$ with minimum total aspiration value from the neighborhood of $x_{now}$
        Move from $x_{now}$ to $x_{next}$, set $x_{now}=x_{next}$
        If $x_{now}$ is better than $x_{best}$, update $x_{best}$
        Update recency based memory, frequency based memory and elite solutions
        If $x_{best}$ does not change for a given number of iterations, update $x_{now}$ according to intensification method
        Until given number of iterations is reached
    Until $x_{best}$ does not change for a given number of iterations or a given number of iterations is reached
Until a given number of iterations is reached
```

CHAPTER 6

COMPUTATIONAL EXPERIMENT

A computational experiment is designed to evaluate the performance of our solution approaches. First the data generation method and performance measures are introduced and then the preliminary and main experiments' results are reported.

6.1. Data Generation

Assembly line balancing literature has many data sets that are used in many studies. We use one of those data sets, the one proposed by Scholl and Klein (1997). The data set resides the precedence network and the task times. We take their precedence diagrams and generate new task times using two distributions: U[1,10] and U[1,50]. The U[1,10] resembles lower processing times and lower variance compared to those of U[1,50].

To generate the initial line balance, firstly, we solve the minimizing total squared workload problem, i.e., our efficiency measure.

The precedence networks that we select include the following well-known networks:

1. Roszieg, N=25 tasks

2. Sawyer, N=30 tasks

3. Gunther, N=35 tasks

4. Modified Kilbrid, N=40 tasks

5. Kilbrid, N=45 tasks

6. Hahn, N=53 tasks

7. Warnecke, N=58 tasks

8. Tonge70, N=70 tasks

9. Wee-Mag, N=75 tasks

10. Arc83, N=83 tasks

11. Lutz2, N=89 tasks

12. Mukherje, N=94 tasks

We observe that our classical approach returns exact solutions to the instances of data sets 1, 2, 3 and 4 within predefined termination limit. The tabu search algorithm could handle the instances in all data sets in two hours.

To see the effect of number of workstations in the new and initial configurations, i.e., K_N and K, we use 6 configurations that are listed in Table 6.1. The table also resides the disrupted workstations. We select the disrupted workstations as the ones that have the highest task assignment alternatives, thereby leading to harder instances.

Table 6.1. *Configurations with K_N, K and disrupted workstations*

Configuration	**K_N**	**K**	**Disrupted Workstations**
1	4	5	WS3
2	4	7	WS2, WS4 and WS6
3	5	6	WS3 or WS4
4	5	8	(WS2, WS4 and WS6) or (WS3, WS5 and WS7)
5	6	7	WS4
6	6	9	WS3, WS5 and WS7

For each N and (K_N, K) combination, we generate 10 problem instances.

We set a termination limit of two hours (elapsed time) to our classical approach and tabu search algorithm.

We observe that the classical approach could solve the instances up to 35 tasks for all 6 configurations. When N=40 tasks, only the first two configurations could be solved in two hours. Therefore, a total of (3 * 6 + 2) = 20 combinations and 20 * 10 = 200 problem instances are solved exactly by the classical approach.

To test the limit of the tabu search algorithm, we continue with the larger sized, i.e., data sets 4 through 12 for (K_N, K) = (6, 7). We try 5 problem instance for each of these 9, N values. Hence, for tabu search algorithm, we solve additional 9 * 5 = 45 problem instances.

All in all, we have 200 + 45 = 245 problem instances.

The classical approach is coded with C++ using Microsoft Visual Studio 2013 and solved by IBM ILOG CPLEX 12.6.2. Moreover, MATLAB 2016a is used for coding the tabu search algorithm. Both classical approach and tabu search algorithm are run on a computer which has properties of Intel Core i7-6700HQ CPU with 3.5 GHz, 16 GB DDR4 Ram and Windows 10.

6.2. Performance Measures

Firstly, in order to see the effect of variability of task times and the effect of using earliest and latest possible task assignment information, two performance measures (Average and Maximum Central Processing Unit Time (CPU)) are used in preliminary experiments. These performance measures are also used in experiments with classical approach in main experiment. Average and maximum number of non-dominated solutions are also reported for all classical approach experiments we made.

Then, in order to compare classical approach and tabu search algorithm, we let ES indicate the exact set of non-dominated objective vectors and AS is the set of non-dominated objective vectors returned by the tabu search algorithm. Note that AS resides the objective vectors that are not proved as exact. Using sets ES and AS, we introduce the following performance measures for the tabu search algorithm. We also report ES values in comparison tables.

PM1: Cardinality of AS

PM2: Percentage of non-dominated objective vectors returned by the tabu search algorithm

$= |ES \cap AS| \,/\, |ES|$

Recall that PM1 and PM2 consider the number of non-dominated objective vectors; however, ignore their closeness to the exact non-dominated objective vectors. Recognizing this fact, we use the following distance measures proposed in Czyzak and Jaszkiewicz (1998).

First, range of our efficiency and stability objectives are calculated.

$\text{Range}_{WL} = WL_{max} - WL_{min}$

WL_{max} and WL_{min} are the maximum and minimum total squared workload values of our exact non-dominated objective vectors.

$\text{Range}_{TR} = TR_{max} - TR_{min}$

TR_{max} and TR_{min} are the maximum and minimum total displacement amount of our exact non-dominated objective vectors.

Let (WL^E, TR^E) is a solution from set ES and (WL^A, TR^A) is a solution from set AS. In this study, we consider the maximum relative distance between (WL^E, WL^A) and (TR^E, TR^A). To clarify,

Distance between solutions (WL^E, TR^E) and (WL^A, TR^A)

$$= \text{Distance}_{\text{WL,TR}} = max\{0, \frac{WL^A - WL^E}{Range_{WL}}, \frac{TR^A - TR^E}{Range_{TR}}\}$$

PM3 is introduced to show average distance between the solutions in set ES and set AS. For all solutions in set ES, closest solution from set AS is found and then average distance value for that problem instance is calculated.

$$PM3 = \frac{1}{|ES|} * \sum_{\forall(\text{WL}_E,\text{TR}_E)\in\text{ES}} min_{\forall(\text{WL}_A,\text{TR}_A)\in\text{AS}} \{Distance_{WL,TR}\}$$

For each solution (WL^E, TR^E) from set ES, closest solution (WL^A, TR^A) from set AS is found. Between these pairs, pair which has maximum distance between (WL^E, TR^E) and (WL^A, TR^A) gives the maximum distance between the solutions in set ES and set AS, i.e., our performance measure, PM4.

$$PM4 = max_{\forall(\text{WL}_E,\text{TR}_E)\in\text{ES}} [min_{\forall(\text{WL}_A,\text{TR}_A)\in\text{AS}} \{Distance_{WL,TR}\}]$$

The smaller values of PM3 and PM4 are preferable. PM1, PM2, PM3 and PM4 are utilized in the study of Karsu and Azizoğlu (2014).

The average CPU times and maximum CPU times are also reported for the performances of the classical approach and the tabu search algorithm. We give the ratio of the CPU times (CPU time by the tabu search / CPU time by the classical approach) as well.

After all, in testing the tabu search algorithm section, we report average CPU time (in seconds) and average number of non-dominated solutions found.

6.3. Preliminary Experiments

To analyze the effect of variability of task times on the solution time and the effect of using earliest and latest possible task assignment information on the performance of the classical approach, a preliminary experiment is designed.

6.3.1. Effect of Task Times

For the effect of variability of task times, we try two discrete uniform distributions. U[1,10] and U[1,50], are selected to represent low and high task time variability, respectively. In doing so, we select two problem combinations, N=35, K_N=6 & K=7 and N=40, K_N=4 & K=5 and the average and maximum number of non-dominated solutions and the average and maximum CPU times are reported in Table 6.2.

Table 6.2. *The performance of CA for U[1,10] and U[1,50]*

		# of non-dominated objective vectors		**CPU time**	
Configuration	**Variability**	**Average**	**Maximum**	**Average**	**Maximum**
N=35 K_N=6 K=7	**U[1,10]**	9.80	15	3,441	4,640
	U[1,50]	14.20	19	5,678	8,304
N=40 K_N=4 K=5	**U[1,10]**	6.00	9	2,305	4,060
	U[1,50]	8.30	10	3,710	6,834

Table 6.2 shows that the problem instances who has high time variability (U[1,50]) are solved harder compared to problem instances who has low time variability (U[1,10]). This situation occurs when the task times are closer and the objective function values are similar. It leads to lower average and maximum number of non-dominated objective vectors. Average number of non-dominated objective vectors, maximum number of non-dominated objective vectors, average CPU times and maximum CPU times all increase when an increase in time variability occurs. When N=35 for K_N=6 & K=7, the average number of non-dominated objective vectors increases from 9.80 to 14.20 and the average CPU time increases from 3,441 to 5,678 with an increase in the variability of the task times. When N=40 for K_N=4 & K=5, the average number of non-dominated objective vectors increases from 6.00 to 8.30 and the average CPU time increases from 2,305 to 3,710 with an increase in the variability of the task times. For other configurations, this observation is monitored similarly. In our main runs, we continue with U[1,50] discrete uniform distribution to tackle with harder problem instances.

6.3.2. Effect of Earliest and Latest Possible Task Assignment Information

The earliest and latest workstations information that any task could be assigned is used in the mathematical model to define limits of the decision variables and reduce the number of constraints. Hence, these expressions hopefully would improve the efficiency of the mathematical models.

In our preliminary experiment, we try to see the effect of earliest and latest workstations information on the CPU times, we select two (K_N, K) combinations with 35 tasks. The selected combinations are (6,7) and (6,9) and the results are reported in Table 6.3. We remove the termination limit in this analysis.

Table 6.3. *The performance of the Classical Approach with and without E_i and L_i*

		CPU time (in seconds)	
Configuration	**Mechanism**	**Average**	**Maximum**
N=35 K_N=6 K=7	**with E_i and L_i**	3,905.84	7,063.98
	without E_i and L_i	5,678.74	8,304.96
N=35 K_N=6 K=9	**with E_i and L_i**	5,164.04	13,890.80
	without E_i and L_i	7,258.98	29,137.54

For K_N=6 & K=7, average CPU time decreases from 5,678.74 to 3,905.84 seconds and maximum CPU time decreases from 8,304.96 to 7,063.98 once the earliest (E_i) and latest (L_i) workstations information are used. Also for K_N=6 & K=9, average CPU time decreases from 7,258.98 to 5,164.04 seconds and maximum CPU time decreases from 29,137.54 to 13,890.80 seconds with the use of E_i and L_i values. Similar observations can be made for the other configurations.

Hence, we conclude that the time to find earliest and latest workstations information via the cycle time upper bound is justified because of the reduction in solution time of the mathematical models.

6.4. Main Experiments

In this chapter, we discuss the main experiment that is based on our preliminary experiment results.

6.4.1. Experiments with Classical Approach (CA)

Preliminary experiments have revealed the power of the earliest and latest possible workstation information in increasing the efficiency of the CA; hence we use them in our main experiment. As mentioned before, we select four different well-known data sets, take precedence networks of data sets and generate new task times by using

U[1,50] discrete uniform distribution. As stated before, we generate six different configurations by using different number of workstations in new (K_N) and initial (K) and configurations.

First, we analyze the average and maximum number of non-dominated objective vectors and report the results in Table 6.4 and Table 6.5. The average CPU times and the maximum CPU times are also reported in Table 6.6 and Table 6.7.

Table 6.4. *The average and maximum number of non-dominated objective vectors for K_N=4 & K=5 and K_N=4 & K=7*

	K_N=4 & K=5		**K_N=4 & K=7**	
	Average	**Maximum**	**Average**	**Maximum**
N=25	6.00	9	4.10	7
N=30	7.20	11	5.00	8
N=35	8.00	10	5.10	7
N=40	8.30	10	5.80	8

Table 6.5. *The average and maximum number of non-dominated objective vectors for K_N=5 & K=6, K_N=5 & K=8, K_N=6 & K=7 and K_N=6 & K=9*

	K=5 & K'=6		**K=5 & K'=8**		**K=6 & K'=7**		**K=6 & K'= 9**	
	Avg.	**Max**	**Avg.**	**Max**	**Avg.**	**Max**	**Avg.**	**Max**
N=25	8.50	13	7.30	12	9.90	15	9.40	16
N=30	10.70	14	10.00	13	13.70	19	10.40	22
N=35	12.00	15	11.00	15	14.20	19	11.67	18

Table 6.6. *The average and maximum CPU times for K_N=4 & K=5 and K_N=4 & K=7*

	K_N=4 & K=5		**K_N=4 & K=7**	
	Average	**Maximum**	**Average**	**Maximum**
N=25	12.74	18.05	17.94	25.54
N=30	35.22	58.06	45.41	101.75
N=35	72.25	101.58	70.04	137.42
N=40	2,597.29	4,574.39	3,017.95	5,671.70

Table 6.7. *The average and maximum CPU times (in seconds) for K_N=5 & K=6, K_N=5 & K=8, K_N=6 & K=7 and K_N=6 & K=9*

	K_N=5 & K=6		**K_N=5 & K=8**		**K_N=6 & K=7**		**K_N=6 & K=9**	
	Avg.	**Max**	**Avg.**	**Max**	**Avg.**	**Max**	**Avg.**	**Max**
N=25	43.22	114.63	49.22	149.41	126.56	259.56	137.09	230.44
N=30	197.03	578.09	245.03	492.28	763.58	1.594.19	1,252.22	4,212.89
N=35	589.96	1,292.70	665.78	1,174.34	3,905.84	7,063.98	4,494.96	7,200 (1)*

* The number in the paranthesis gives the number of unsolved instances in 2 hours

For K_N=4 & K=5 and K_N=4 & K=7, we can solve the problem instances up to N=40 and for K_N=5 & K=6, K_N=5 & K=8, K_N=6 & K=7 and K_N=6 & K=9, we can solve the problem instances up to N=35.

As you can see from Table 6.4 and Table 6.5, the non-dominated objective vectors are increasing with increases in N in all of the six configurations. The CPU times are also increasing as displayed in Table 6.6 and 6.7. For instance, for K_N=4 & K=5, the average number of non-dominated objective vectors are 6.00, 7.20, 8.00 and 8.30 and the average CPU times are 12.74, 35.22, 72.25, 2,597.29, for N=25, N=30, N=35 and

N=40, respectively. Approximate results could be observed in remaining combinations, too. We observe that increasing the number of tasks affects CPU time very significantly and this increase is much more significant than the increase caused by the number of workstations. This result is expected by reason of increasing the number of tasks increases the number of decision variables and constraints more than increasing the number of workstations.

The effect of number of workstation in new configuration (K_N) and the effect of number of workstations in initial configuration (K) can be also observed in Tables 6.4 – 6.7. For this purpose, two combinations are selected.

- Combination 1:
  - for the non-dominated objective vectors: first column in Table 6.4 and first column & fifth column in Table 6.5
  - for CPU times: first column in Table 6.6 and first column & fifth column in Table 6.7
- Combination 2:
  - for the non-dominated objective vectors: third column in Table 6.4 and third column & seventh column in Table 6.5
  - for CPU times: third column in Table 6.6 and third column & seventh column in Table 6.7

In both combinations, increasing the number of workstation in new (K_N) and initial configurations (K) lead to an increase in the non-dominated objective vectors and an increase in CPU times. For instance, when N=35 in Combination 1, the average number of non-dominated objective vectors are 8.00, 12.00 and 14.20 for K_N=4 & K=5, K_N=5 & K=6 and K_N=6 & K=7 respectively and the associated average CPU times are 72.25, 589.96 and 3,905.84 seconds, respectively. When N=35 in Combination 2, the average number of non-dominated objective vectors are 5.10, 11.00 and 11.67 for K_N=4 & K=7, K_N=5 & K=8 and K_N=6 & K=9, respectively and the associated average CPU times are 70.04, 665.78 and 4,494.96 seconds, respectively. This result is expected as when the disrupted number of workstations is fixed, increasing the number of workstations for both initial and new configuration leads to an increase in the number of decision variables and constraints. The assembly lines with higher number of workstations may have more optimal solution alternatives (more non-dominated objective vectors) when disruption occurs and higher CPU time is needed to solve the mathematical models.

The effect of the number of disrupted workstations (K-K_N) for fixed number of workstations in new configuration (K_N) is also investigated from Table 6.4, Table 6.5, Table 6.6 and Table 6.7. For this purpose, three combinations are selected.

- Combination 1:
  - for the non-dominated objective vectors: first column & third column in Table 6.4
  - for CPU times: first column & third column in Table 6.6
- Combination 2:
  - for the non-dominated objective vectors: first column & third column in Table 6.5
  - for CPU times: first column & third column in Table 6.7
- Combination 3:
  - for the non-dominated objective vectors: fifth column & seventh column in Table 6.5
  - for CPU times: fifth column & seventh column in Table 6.7

In all selected combinations, increasing K-K_N (the number of disrupted workstations) for fixed K_N (number of workstations in new configuration) leads to a reduction in the number of non-dominated objective vectors although CPU time needed to achieve a non-dominated vector is increased. For instance, when N=25 in Combination 1, the average number of non-dominated objective vectors decreases from 6.00 to 4.10, 8.50 to 7.30 and 9.90 to 9.40 for K_N=4 & K=5 to K_N=4 & K=7, K_N=5 & K=6 to K_N=5 & K=8 and K_N=6 & K=7 to K_N=5 & K=8, respectively and the respective average CPU times increase from 12.74 to 17.94, 43.22 to 49.22 and 126.56 to 137.09. Similar results hold for in all other number of tasks and over all combinations. Hence, we can say that the assembly lines with higher number of disrupted workstations, and higher number of initial workstations have less optimal solution alternatives (fewer number of non-dominated objective vectors) although the CPU time to achieve a non-dominated objective vector is higher.

The effect of K-K_N (the number of disrupted workstations) for fixed K (number of workstations in initial configuration) can be also observed in Tables 6.4 – 6.7. For this purpose, third column in Table 6.4 & fifth column in Table 6.5 and third column in Table 6.6 & fifth column in Table 6.7 can be used. In all combinations of the number of tasks, increasing the number of workstations (from 1 to 3) causes a decrease in both the number of the non-dominated objective vectors and in CPU times. Specifically, from K_N=6 & K=7 to K_N=4 & K=7, the average number of non-dominated objective vectors decreases from 9.90 to 4.10, 13.70 to 5.00 and 14.20 to 5.10 for N=25, N=30 and N=35, respectively and the respective average CPU times decreases from 126.56 to 17.94, 763.58 to 45.41 and 3,905.84 to 70.04. Therefore, we can say that higher number of disrupted workstations for fixed number of workstations in initial configuration causes less optimal solution alternatives (fewer number of non-dominated objective vectors) and significantly smaller CPU time is needed to achieve a non-dominated objective vector.

6.4.2. Comparison Between Classical Approach and Tabu Search Algorithm

Note that, our classical approach could solve the problem instances with K_N=4 & K=7 and N=40. For bigger problem instances, we propose a multi-objective tabu search (MOTS) algorithm. The performance of our MOTS algorithm according to the exact available solutions by the classical approach are also compared. We use the performance measures mentioned in Chapter 6.2 and report the results in Table 6.8 and Table 6.9.

Table 6.8. *Comparison between Classical Approach and Tabu Search Algorithm for N=25 and N=30*

	N=25		N=30	
Performance Measures	**K_N=5 & K=6**	**K_N=5 & K=8**	**K_N=6 & K=7**	**K_N=6 & K=9**
The average number of exact non-dominated objective vectors	9.40	7.40	12.60	8.00
PM1	8.60	6.00	8.20	6.40
PM2	91.49%	81.08%	65.08%	80.00%
PM3	0.46%	0.41%	0.78%	0.75%
PM4	4.35%	3.04%	5.36%	5.05%
Average CPU Time of Tabu Search Algorithm (in seconds)	20.21	20.28	144.27	154.77
Average CPU Time of Classical Approach (in seconds)	59.88	55.85	641.61	1,205.08
% Average (CPU Time of Tabu Search Algorithm / CPU Time of Classical Approach) (in seconds)	33.75%	36.31%	22.49%	12.84%

Table 6.9. *Comparison between Classical Approach and Tabu Search Algorithm for N=35 and N=40*

	N=35		**N=40**	
Performance Measures	**K_N=6 & K=7**	**K_N=6 & K=9**	**K_N=4 & K=5**	**K_N=4 & K=7**
The average number of exact non-dominated objective vectors	15.60	11.40	8.00	5.40
PM1	8.40	6.60	5.60	4.00
PM2	53.85%	57.89%	70.00%	74.07%
PM3	0.85%	0.96%	0.98%	1.06%
PM4	5.46%	5.84%	5.59%	5.20%
Average CPU Time of Tabu Search Algorithm (in seconds)	207.74	197.86	87.91	94.72
Average CPU Time of Classical Approach (in seconds)	4,261.76	3,743.06	2,952.36	3,045.80
% Average (CPU Time of Tabu Search Algorithm / CPU Time of Classical Approach) (in seconds)	4.87%	5.29%	2.98%	3.11%

Recall that PM2 shows the percentage of the number of exact non-dominated objective vectors returned by the tabu search algorithm. Table 6.8 shows that average PM2 value is higher than 80% for K_N=5 & K=8 and 90% for K_N=5 & K=6 when N=25. These percentages for N=30 and K_N=6 are about 65% and 80% when K=7 and K=9, respectively. For higher N values, the PM2 values deteriorate, however still they are about 55% and more than 70%, when N=35 and 40, respectively. Those figures altogether show the satisfactory performance of tabu search algorithm in generating the exact points of the efficient set. Moreover, for different values of K, we observe the consistently well performance of the tabu search algorithm.

Performance measures PM3 and PM4 measure the average and the maximum distance between the non-dominated point found by classical approach and corresponding point found with tabu search algorithm. In our study, we give more importance to PM3 and PM4 performance measures more than CPU time. Therefore, we reach 0.46 and 0.41 percentages when N=25, 0.78 and 0.75 percentages when N=30, 0.85 and 0.96 percentages when N=35 and 0.98 and 1.06 percentages when N=40 for performance measure PM3. Moreover, we reach 4.35 and 3.04 percentages when N=25, 5.36 and 5.05 percentages when N=30, 5.46 and 5.84 percentages when N=35 and 5.59 and 5.20 percentages when N=40 for performance measure PM4. Similar to performance measure PM2, our algorithm shows stable performance for PM3 and PM4, according to the number of disrupted workstations, too.

The low values of PM3 and PM4 over all problem instances verify that the non-dominated objective vectors returned by the tabu search algorithm are very close to their exact counterparts.

We observe that the CPU times of the tabu search algorithm are not significantly affected by the number of disrupted workstations. For example, the CPU times are 144.27 and 154.77 seconds (7 percent difference occur) for K_N=6 & K=7 and K_N=6 & K=9, respectively, while corresponding CPU times for classical approach are

641.61 and 1,205.08 (88 percent difference occur) for N=30. We can say that the tabu search algorithm CPU times are more consistent than those of the classical approach.

6.4.3. Testing Limits of Tabu Search Algorithm

By using data sets from the ALB literature, we test the limits of our multi-objective tabu search algorithm. We use (K_N, K) = (6,7) and repeat the experiment 5 times for each value of N. The 10 N values used are 30, 35, 45, 53, 58, 70, 75, 83, 89 and 94. Hence, we solve 10 * 5 = 50 problem instances and the average CPU times and the average number of non-dominated objective vectors are reported in Table 6.10.

Table 6.10. *Average CPU times and number of non-dominated objective vectors found by Tabu Search Algorithm*

N	Name of Dataset	Avg. CPU Time	Avg. # of non-dominated objective vectors
30	Sawyer	145	8.20
35	Gunther	210	8.40
45	Kilbrid	870	9.20
53	Hahn	1,130	12.20
58	Warnecke	1,740	16.80
70	Tonge70	2,800	17.40
75	Wee-Mag	3,540	21.40
83	Arc83	4,710	24.00
89	Lutz2	5,650	26.60
94	Mukherje	6,630	30.20

Note that, the CPU times increase, almost linearly, with increasing in number of task (N) values. We set a termination limit as 2 hours elapsed time and find that the instance with more than 100 tasks could not be solved in the limit.

We also observe that the number of non-dominated objective vectors increases while number of tasks (N) increases.

CHAPTER 7

CONCLUSIONS

In this study, an assembly line rebalancing problem is considered. It is assumed that the disruption occurs on one or more workstations which causes the current line balance become infeasible. We measure the performance of the disrupted line balance in two ways: minimizing total squared workloads and minimizing the total distance traveled due to the changes in task assignments after the disruption.

We aim to compose all non-dominated objective vectors according to total distance traveled and total squared workload. We propose two approaches: a classical approach that returns the exact solutions and a tabu search algorithm that returns the approximate solutions.

Results obtained in our computational experiment reveal that the performance of the classical approach is sensitive to the earliest and latest possible workstation information and the approach can handle problem instances with up to 40 tasks, 7 workstations in initial configuration and 4 workstations in new (after disruption) workstations.

We observe that the tabu search approach could handle problem instances with up to about 100 tasks, 7 workstations in initial configuration and 6 workstations in new (after disruption) workstations. The performance of the solutions is found to be close to their exact counterparts.

To the best of our knowledge we propose the first rebalancing study with total squared workload and total distance travel objectives.

Our study might stimulate new research directions some of which are mentioned below:

- New efficiency and stability measures can be used.
- Implicit enumeration techniques can be developed to solve the sub-problems of the classical approach
- Our results can be extended to more general (like U-shaped, mixed model, parallel station, flexible) assembly lines.
- Multi-objective decision making techniques can be used to evaluate and classify non-dominated objective solutions.
- Different task characteristics (like task displacement weights, stochastic task times and dependent tasks) can be used.

REFERENCES

Azizoglu, M., & Imat, S. (2018). Workload smoothing in simple assembly line balancing. *Computers & Operations Research*, *89*, 51–57.

Battaia, O., & Dolgui, A. (2013). A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, *142*(2), 259–277.

Baybars, I. (1986). A survey of exact algorithms for the simple assembly line balancing problem. *Management Science*, *32*(8), 909–932.

Belassiria, I., Mazouzi, M., Elfezazi, S., Cherrafi, A., & Elmaskaoui, Z. (2018). An integrated model for assembly line re-balancing problem. *International Journal of Production Research*, *56*(16), 5324–5344.

Boysen, N., Fliedner, M., & Scholl, A. (2007). A classification of assembly line balancing problems. *European Journal of Operational Research*, *183*(2), 674–693.

Celik, E., Kara, Y., & Atasagun, Y. (2014). A new approach for rebalancing of U-lines with stochastic task times using ant colony optimisation algorithm. *International Journal of Production Research*, *52*(24), 7262–7275.

Czyzzak, P., & Jaszkiewicz, A. (1998). Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of Multi-Criteria Decision Analysis*, *7*(1), 34–47.

Esmaeilbeigi, R., Naderi, B., & Charkhgard, P. (2015). The type E simple assembly line balancing problem: A mixed integer linear programming formulation. *Computers & Operations Research*, *64*, 168–177.

Finco, S., Battini, D., Delorme, X., Persona, A., & Sgarbossa, F. (2019). Workers' rest allowance and smoothing of the workload in assembly lines. *International Journal of Production Research*, 1–16.

Gamberini, R., Gebennini, E., Grassi, A., & Regattieri, A. (2009). A multiple single-pass heuristic algorithm solving the stochastic assembly line rebalancing problem. *International Journal of Production Research*, *47*(8), 2141–2164.

Gamberini, R., Grassi, A., & Rimini, B. (2006). A new multi-objective heuristic algorithm for solving the stochastic assembly line re-balancing problem. *International Journal of Production Economics*, *102*(2), 226–243.

Grangeon, N., Leclaire, P., & Norre, S. (2011). Heuristics for the re-balancing of a vehicle assembly line. *International Journal of Production Research*, *49*(22), 6609–6628.

Groover, M. P. (2013). *Principles of modern manufacturing: SI version*. New Delhi: Wiley.

Haimes, Y. V., Lasdon, L. S., & Wismer, D. A. (1971). On a bicriterion formulation of the problems of integrated system identification and system optimization. *IEEE Transactions on Systems, Man, and Cybernetics*, *3*, 296–297.

Karsu, O., & Azizoglu, M. (2014). Bicriteria multiresource generalized assignment problem. *Naval Research Logistics (NRL)*, *61*(8), 621–636.

Kim, Y. J., Kim, Y. K., & Cho, Y. (1998). A heuristic-based genetic algorithm for workload smoothing in assembly lines. *Computers & Operations Research*, *25*(2), 99–111.

Makssoud, F., Battaia, O., Dolgui, A., Mpofu, K., & Olabanji, O. (2015). Re-balancing problem for assembly lines: new mathematical model and exact solution method. *Assembly Automation*, *35*(1), 16–21.

Mozdgir, A., Mahdavi, I., Badeleh, I. S., & Solimanpur, M. (2013). Using the Taguchi method to optimize the differential evolution algorithm parameters for minimizing the workload smoothness index in simple assembly line balancing. *Mathematical and Computer Modelling*, *57*(1-2), 137–151.

Nearchou, A. C. (2011). Maximizing production rate and workload smoothing in assembly lines using particle swarm optimization. *International Journal of Production Economics*, *129*(2), 242–250.

Ponnambalam, S. G., Aravindan, P., & Naidu, G. M. (2000). A multi-objective genetic algorithm for solving assembly line balancing problem. *The International Journal of Advanced Manufacturing Technology*, *16*(5), 341–352.

Rachamadugu, R., & Talbot, B. (1991). Improving the equality of workload assignments in assembly lines. *International Journal of Production Research*, *29*(3), 619–633.

Rosenberg, O., & Ziegler, H. (1992). A comparison of heuristic algorithms for cost-oriented assembly line balancing. *Zeitschrift Für Operations Research Methods and Models of Operations Research*, *36*(6), 477–495.

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

Sanci, E., & Azizoglu, M. (2017). Rebalancing the assembly lines: exact solution approaches. *International Journal of Production Research*, *55*(20), 5991–6010.

Scholl, A. (2007). Data sets for SALBP. Retrieved from http://www.assembly-line-balancing.de/

Sewell, E. C., & Jacobson, S. H. (2012). A branch, bound, and remember algorithm for the simple assembly line balancing problem. *INFORMS Journal on Computing*, *24*(3), 433–442.

Sivasankaran, P., & Shahabudeen, P. (2014). Literature review of assembly line balancing problems. *The International Journal of Advanced Manufacturing Technology*, *73*(9-12), 1665–1694.

Smunt, T. L., & Perkins, W. C. (1985). Stochastic unpaced line design: Review and further experimental results. *Journal of Operations Management*, *5*(3), 351–373.

Yang, C., Gao, J., & Sun, L. (2013). A multi-objective genetic algorithm for mixed-model assembly line rebalancing. *Computers & Industrial Engineering*, *65*(1), 109–116.