REBALANCING OF ASSEMBLY LINES

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

BY

ECE SANCI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
INDUSTRIAL ENGINEERING

JULY 2015

Approval of the thesis:

REBALANCING OF ASSEMBLY LINES

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

Prof. Dr. Gülbin Dural Ünver
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Murat Köksalan
Head of Department, **Industrial Engineering** _____

Prof. Dr. Meral Azizoğlu
Supervisor, **Industrial Engineering Dept., METU** _____

Examining Committee Members:

Assist. Prof. Dr. Sakine Batun
Industrial Engineering Dept., METU _____

Prof. Dr. Meral Azizoğlu
Industrial Engineering Dept., METU _____

Assist. Prof. Dr. Banu Lokman
Industrial Engineering Dept., METU _____

Assist. Prof. Dr. Mustafa Kemal Tural
Industrial Engineering Dept., METU _____

Assist. Prof. Dr. Mustafa Alp Ertem
Industrial Engineering Dept., Cankaya University _____

Date: July 24, 2015

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ECE SANCI

Signature:

ABSTRACT

REBALANCING OF ASSEMBLY LINES

Sancı, Ece
M.S., Department of Industrial Engineering
Supervisor: Prof. Dr. Meral Azizoğlu

July 2015, 82 pages

In this study, we consider an assembly line rebalancing problem. We assume that there is a disruption on one or more workstations that makes the current solution infeasible. After the disruption, we aim to find a rebalance so as to catch the trade-off between the efficiency measure of cycle time and the stability measure of number of tasks assigned to different workstations in the original and new solutions.

We generate all nondominated objective vectors with respect to our efficiency and stability measures. We develop two optimization algorithms: classical approach and branch and bound algorithm. The results of our experiments show the favorable behaviors of both algorithms and superiority of branch and bound algorithm.

Keywords: Assembly Lines, Rebalancing, Branch and Bound Algorithm

ÖZ

MONTAJ HATLARININ YENİDEN DENGELENMESİ

Sancı, Ece
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Meral Azizoğlu

Temmuz 2015, 82 sayfa

Bu çalışmada, montaj hattı yeniden dengeleme problemini ele aldık. Bir ya da birden fazla iş istasyonunda oluşan bir aksamanın, mevcut dengelemeyi uygulanamaz duruma getirdiğini varsaydık. Aksamadan sonra, verimlilik ölçütümüz olan çevrim süresi, kararlılık ölçütümüz olan eski ve yeni atamalarda farklı istasyonlara atanan iş sayısının ödünleşimini düşünen, yeni bir denge bulmayı hedefledik.

Verimlilik ve kararlılık ölçütlerimize göre tüm bastırılmamış çözümleri yarattık. İki optimizasyon algoritması –klasik yaklaşım ve dal-sınır algoritması– geliştirdik. Deneysel sonuçlarımız, algoritmaların başarılı davranışlarını ve dal-sınır algoritmasının üstünlüğünü göstermektedir.

Anahtar Kelimeler: Montaj Hatları, Yeniden Dengeleme, Dal ve Sınır Algoritması

To our beloved Deniz

ACKNOWLEDGMENTS

I wish to express my deepest gratitude to my supervisor Dr. Meral Azizoğlu for the great contribution she made to this study with her brilliant ideas and broad knowledge. I would like to thank to her also for being such a caring, supportive and motivating supervisor from the first day we started to study together.

I would also like to thank to my family who always encourages me to pursue the life I dream. They were always there to support me whenever I felt like this dream was too challenging. Their love was the only resource I needed, and it was good to know it was not a scarce resource at all.

I would like to present my thanks to many other people who contributed to this study directly or indirectly starting from my dear friends Hannan Türeci, Ezgi Doğan, Ahmet Kuzucuoğlu and Nusret Köse with whom I started to this journey together. I would like to thank to Tolga Kalaycıoğlu for the support and confidence he provided. I would like to express my thanks to all of my friends from our lovely department Semih Ali Aksoy, Taha Yasin Çelik, Kağan Karataylı, Nermin Haşimova, Cansu Yılmaz, Melike İşbilir, Emre Eryiğit, Deniz Sun, Can Öz, Ekrem Duman, Fidem Koç, Başak Kaymaz. Also, I would like to thank to Ufuk Işık and Nur Çöllü for bringing my life a new point of view. I would like to thank to Anıl Yıldız, Berk Yıldız and Samet Söylemez for cheering me up when I needed most during the difficult times.

I would like to thank to my dear assistant friends, especially to Utku Can Kunter, for being the best roommate, and to Yasemin Limon, Gökhan Ceyhan and Gökçe Özkan, for being the most supportive colleagues.

I would like to express my gratitude once more to Dr. Meral Azizoğlu along with Dr. İsmail Serdar Bakal, Dr. Sakine Batun and Dr. Melih Çelik for being such thoughtful and helpful guides. It was my pleasure to assist them.

I would like to thank to Andaç Kürün for his help with coding and to Ali Cem Randa and Vugar Abdullayev for their help with my doctoral application processes.

Last but not least, I would like to thank to the examining committee members for their valuable contributions to this study.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

An assembly line is a flow-oriented production system which consists of a sequence of workstations performing repetitive tasks. Assembly lines are cost efficient means of production which are more typical in the production of high volume standardized commodities. An important problem arising in managing assembly lines regards the decision of task assignments to workstations in an efficient way. This problem is called the assembly line balancing (ALB) problem which is a well-studied combinatorial optimization problem. Although there is a vast amount of ALB studies in the literature, its research is still attractive. This is mainly due to the fact that the structures of assembly lines change dynamically with the requirements imposed by the changes in the industries. To illustrate, although the first assembly lines were designed to produce a single identical product, it is more common to produce customized products belonging to similar product families on a single assembly line in today's industries due to the changes in the customer requirements. Thus, there are still important aspects of assembly line balancing which researchers may find challenge.

The basic version of the ALB problem is first proposed by Salveson (1955). The balancing problem considered in this study is named simple assembly line balancing (SALB) problem which has several underlying assumptions such as:

- There is one homogeneous product to be assembled on the line.

- The line is paced with a fixed cycle time.
- The task times are deterministic.
- The only assignment restriction is due to the precedence relationships.
- The line has a serial layout with one-sided workstations.
- All workstations are equally equipped.

Considering this problem environment, Salveson (1955) proposed a mathematical model which assigns tasks to workstations without exceeding the cycle time and satisfying precedence relationships. After Salveson, ALB literature is extended by relaxing these assumptions which results in different variants of the ALB problems. Although different categorizations of the ALB problem are possible, the most fundamental one is based on the objective functions considered, i.e., how the efficiency of the balancing problem is defined. There are mainly two types of the ALB problem regarding the objective functions: Type I ALB problem aims to minimize the number of workstations for a given cycle time whereas Type II ALB problem aims to minimize the cycle time for a given number of workstations. Type I ALB problem is usually considered before the configuration of an assembly line when the quantity of the product to be produced is planned. On the other hand, Type II ALB problem usually arises for already configured assembly lines with a new problem environment.

The assembly line rebalancing problem is rather a neglected problem despite its practical importance. Most of the assembly lines are not installed from the scratch but reconfigured according to the changing circumstances. Thus, it is more convenient to rebalance the line by taking the initial configuration into account rather than solving a balancing problem all over again each time one of the input parameters changes, e.g. change in demand pattern, change in task times, technological restrictions or workstation breakdowns. These changes are referred

to as disruptions. After any disruption, the stability aspect should be concerned along with the efficiency of the assembly line, since costs associated with operators training, quality assurance and equipment installation incur once a task is removed from one workstation to another.

In this thesis, we consider a rebalancing problem where the tasks are already assigned to workstations. A disruption occurs and affects one or more workstations so that the tasks should be reassigned to the other workstations with respect to our efficiency and stability measures. Our efficiency measure is the cycle time and our stability measure is the number of disrupted tasks. We define a task as disrupted if it has to be moved to a different workstation after disruption. The problem we consider has a multi-criteria nature since it is possible to increase the efficiency of the line, i.e., decrease the cycle time, with a sacrifice in the stability measure, i.e., increase in the number of disrupted tasks. Our aim is to generate all nondominated objective vectors with respect to these two measures. We assume that the utility function of the decision maker is unknown or complex, and the decision maker would make a comparison between all nondominated objective vectors according to his/her preferences once they are all presented.

The related literature on assembly line rebalancing problem is very scarce and the existing studies propose only approximate solution procedures. The lack of exact solution procedures along with the complexity of the problem have motivated us to develop solution approaches for the rebalancing problem. We present two exact solution procedures: the classical approach which sequentially generates all nondominated objective vectors and the branch and bound algorithm which simultaneously generates all nondominated objective vectors. The superiority of the branch and bound algorithm is shown in the computational study, and the classical approach remains as an attractive alternative in particular when a decision maker prefers a solution in a defined space.

The organization of the thesis is as follows: In Chapter 2, we review the related literature on the assembly line balancing and rebalancing problems. In Chapter 3,

we define the problem and give the associated mixed integer linear program. We present the classical approach and the branch and bound algorithm in Chapter 4 and Chapter 5, respectively. In Chapter 6, we discuss the results of our preliminary and main experiments. In Chapter 7, we give our conclusion remarks and point out some future study directions.

CHAPTER 2

LITERATURE REVIEW

In this section, we first review simple assembly line balancing (SALB) literature and then discuss the previous studies on the assembly line rebalancing problems.

2.1 LITERATURE ON SIMPLE ASSEMBLY LINE BALANCING (SALB)

The SALB studies are of two types: Type I (SALB-I) and Type II (SALB-II). The Type I problem minimizes the number of workstations subject to the constraint that the cycle time is not exceeded. The Type II problem minimizes the cycle time subject to the constraint that the number of workstations is not exceeded.

2.1.1 LITERATURE ON SALB-I

SALB-I is an extensively studied problem with many variants. The problem is shown to be NP-hard in the strong sense (see Baybars,1986). Many exact and heuristic procedures are developed for its solution. Among the exact algorithms, branch and bound algorithms play a dominant place. The most prominent branch and bound algorithms developed for the SALB problem are due to Johnson (1988), Nourie and Venta (1991), Hoffmann (1992) and Scholl and Klein (1997, 1999). The algorithm presented in Johnson (1988), named FABLE, uses task-oriented depth first search strategy while incorporating several lower bounds and dominance rules. Nourie and Venta (1991) develop an algorithm called OptPack

with a compact tree structure which stores partial solutions. The algorithm called EUREKA is developed by Hoffmann (1982). EUREKA uses a heuristic along with a branch and bound algorithm that explores either by assigning tasks first to the first workstations or first to the last workstations. A branch and bound algorithm called SALOME is developed in Scholl and Klein (1997, 1999). SALOME uses a bidirectional search with effective lower bounds and dominance rules. The computational experiments performed for 269 problem instances show that SALOME is superior to FABLE and EUREKA. Indeed, SALOME remained the best exact algorithm for SALB-I for years. Sewell and Jacobson (2012) present a new algorithm which outperforms SALOME. They refer this algorithm as the branch, bound and remember algorithm. Sewell and Jacobson (2012)'s algorithm uses cyclic best first search strategy, good lower bounds and a memory based dominance rule. Morrison et al. (2014) extend the study of Sewell and Jacobson (2012) with a new backtracking procedure and report superior results.

2.1.2 LITERATURE ON SALB-II

Almost all SALB-II studies in the literature use successive solutions of the SALB-I problem. Different enumeration techniques for these solution procedures are utilized which are mainly based on iterating for a trial cycle time between a lower bound and an upper bound on the cycle time. One of these enumeration techniques is the lower bound method in which the cycle time is successively increased by one starting from the lower bound until a feasible assignment is obtained. The direct procedures for the SALB-II problems are due to Scholl (1994) and Klein and Scholl (1996). The branch and bound algorithm presented by Scholl (1994) employs a task-oriented depth first strategy along with several lower bounds and dominance rules. Klein and Scholl (1996) develop an algorithm based on SALOME and adapt it for directly solving SALB-II. They use an efficient enumeration technique that employs several lower and upper bounding schemes along with several dominance rules. They test the performance of their

algorithm on 302 problem instances and they compare the results with the solution procedures incorporating the solutions of SALB-I which are based on FABLE, EUREKA and SALOME. They also compare their results with Scholl (1994). The computational study shows the superiority of their procedure.

For extensive review of the assembly line balancing problem, we refer the reader to the survey study of Battaia and Dolgui (2013).

2.2 LITERATURE ON ASSEMBLY LINE REBALANCING

Unlike to the SALB problem, there are only a few studies on rebalancing of assembly lines in the literature. The rebalancing studies on assembly lines can be divided into two categories regarding to the nature of the task times.

2.2.1 LITERATURE ON REBALANCING PROBLEMS WITH DETERMINISTIC TASK TIMES

All deterministic studies consider heuristic approaches with different efficiency and stability measures. Those studies are due to Grangeon et al. (2011), Yang et al. (2013) and Zha and Yu (2014).

Grangeon et al. (2011) study a real life mixed model rebalancing problem in a French automotive firm. The tasks are assigned to workstations each month according to the master sequencing which includes the number of vehicles to be produced for each model. The task assignments done for a particular month may violate some of the constraints for the following months so that the line should be rebalanced. Grangeon et al. (2011) propose a heuristic that has three phases: The first phase aims to obtain a feasible solution, the second phase aims to decrease the number of workstations and the third phase aims to smooth the workload while transferring a minimum number of tasks. They apply their heuristic to five different industrial instances.

Yang et al. (2013) consider a mixed model rebalancing problem. Their problem is to reassign the tasks to the workstations for a given cycle time when demand structure or technological requirements change. As efficiency measures, they consider the number of workstations and workload variation of each workstation for different models. As stability measure, they take rebalancing cost that is defined as the total processing time of reassigned tasks. To generate the approximate set of nondominated objective vectors with respect to the efficiency and stability measures, they present a multi-objective genetic algorithm. The efficiency of the algorithm is improved by a local search procedure. They test the performance of their algorithm on 23 representative mixed model assembly line instances where the assembly line has to be rebalanced due to the change in demand structure of the models. They report the objective vectors obtained for each of the 23 instances along with the CPU times of their algorithm.

Zha and Yu (2014) propose a hybrid approach for balancing and rebalancing single model U-shaped assembly lines. Their solution procedure combines ant colony optimization algorithm with filtered beam search in order to minimize the total of moving cost of machines and labor costs and minimize the walking time of operators. The algorithm can be used both for the balancing and rebalancing problem such that moving cost is simply zero for the balancing problem and different than zero for the rebalancing problem. Their tests on 25 benchmark problems indicate that the algorithm performs quite efficient when compared to the existing solution procedures developed for the U-shaped assembly lines.

2.2.2 LITERATURE ON REBALANCING PROBLEMS WITH STOCHASTIC TASK TIMES

The rebalancing studies with stochastic task times also consider heuristic approaches with different efficiency and stability measures. Those studies are due to Gamberini et al. (2006), Gamberini et al. (2009) and Celik et al. (2014).

Gamberini et al. (2006) consider a rebalancing problem with a single model. They consider a problem environment where the assembly line has to be rebalanced for a given cycle time after some changes occurred in the input parameters. They define their efficiency measure as the unit total expected completion cost and stability measure as the tasks reassignment. They assume that the unit total expected completion cost is the sum of the total labor cost and the total expected incompletion cost using the idea of Kottas and Lau (1973). Moreover, they introduce an index called the task similarity factor in order to measure the similarity between the initial and the new balances. They propose a multi-objective heuristic algorithm in order to generate an approximate set of nondominated objective vectors. Their solution procedure integrates the well-known heuristic procedure Kottas and Lau (1973) developed for solving stochastic assembly line balancing problems and the technique for order preference by similarity to ideal solution (TOPSIS) proposed by Hwang and Yoon (1981). In the computational study conducted, they consider the technological changes in the product assembled resulting in an altered precedence network. They compare the performance of their algorithm with Kottas and Lau (1973) on 2160 test problems and they report that they obtain improved results for both of the objectives in more than half of the problems and a reduction in either of the objectives is achieved in the rest of the problems.

Gamberini et al. (2009) consider the same problem context stated in Gamberini et al. (2006). They propose a multiple single-pass heuristic algorithm and a multi-objective genetic algorithm (MOGA) in order to find a representative Pareto front. In the multiple single-pass heuristic, they use four different single-pass heuristics

mainly differing in how the attribute related to the completion cost is defined. They fine tune the multiple single-pass heuristic experimenting on 240 problems and compare their algorithm with the MOGA. For all of the problems, the solutions obtained by the MOGA are dominated by those of the multiple single-pass heuristic. They also compare the performance of the multiple single-pass algorithm with Kottas and Lau (1973) and Gamberini et al. (2006) on a single case study and report outperforming behavior of the multiple single-pass algorithm.

Celik et al. (2014) propose an ant colony optimization algorithm to solve the rebalancing problem for single model U-shaped assembly lines. They consider only a single objective function called the total cost of rebalancing which includes task transposition costs, workstation opening or closing costs and workstation operating cost over a definite planning horizon. They conduct experiments on 1320 test problems to test the performance of their algorithm under different problem settings.

The most closely related published work to ours is Gamberini et al. (2006)'s study. We take the number of workstations as parameters whereas they decide on the number of workstations. Both their study and our study aim to generate the set of nondominated objective vectors. We generate the set exactly whereas their generation process is approximate.

CHAPTER 3

PROBLEM DEFINITION

We consider N tasks that are already assigned to K' workstations. A disruption occurs and affects a defined set of workstations. After the disruption, the tasks are to be assigned to the remaining workstations, i.e., nondisrupted workstations. We let K denote the set of these nondisrupted workstations that are the workstations in the new configuration.

We assume that there is a single product that is to be assembled. The assembly line is serial with one-sided workstations. The parameters after disruptions are predefined and are not subject to any change, i.e., the system is deterministic and static. All tasks can be assigned to all workstations; however, there is a penalty of assignment if any task is assigned to a different workstation than its original.

We assume that the initial configuration is already known. We use terms 'initial', 'original' and 'old' configuration, assignment and workstation, interchangeably. The configuration after the disruption is a decision and is referred to as new configuration, assignment and workstation, as sometimes simply configuration, assignment and workstation.

The processing time of task i is defined as t_i and IP_i is the immediate predecessors set of task i. The processing of task i can start only when all tasks in IP_i are complete.

We use binary variable x_{ik} to explain our workstation assignment decisions where

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

for i=1,...,N and k=1,...,K

Note x_{ik}'s explain the new configuration.

We let CT be the cycle time. CT corresponds to the maximum workload of overall workstations. The workload of a particular workstation is the sum of the processing times of the tasks assigned to that workstation.

We let ND be the number of disrupted tasks.

We define the rebalancing problem with the following two criteria:

- Minimize CT
- Minimize $\sum_{i=1}^{N} \sum_{k=1}^{K} c_{ik} x_{ik} = ND$

where $c_{ik} = \begin{cases} 1, \text{if task } i \text{ is not assigned to workstation } k \text{ in the old assignment} \\ 0, \text{otherwise} \end{cases}$

for i=1,...,N and k=1,...,K

Hence, $\sum_{i=1}^{N} \sum_{k=1}^{K} c_{ik} x_{ik}$ can be explained as the number of tasks assigned to different workstations in the new assignment (assignment that considers the effect of disruption) and the old assignment (assignment that was used before the disruption).

Our constraint set is as defined below:

$$\sum_{k=1}^{K} x_{ik} = 1 \qquad i = 1, \dots, N \qquad (1)$$

$$\sum_{i=1}^{N} t_i x_{ik} \leq CT \qquad k = 1, \dots, K \qquad (2)$$

$$\sum_{k=1}^{K} k\, x_{ik} \leq \sum_{k=1}^{K} k\, x_{jk} \quad j = 1, \ldots, N \text{ and } \forall i \in IP_j \tag{3}$$

$$x_{ik} = 0 \text{ or } 1 \qquad i = 1, \ldots, N \text{ and } k = 1, \ldots, K \tag{4}$$

Constraint Set (1) ensures that each task is assigned. Constraint Set (2) defines the maximum workload, i.e., $Max_k\{\sum_i t_i x_{ik}\}$. The precedence relations are controlled by Constraint Set (3). According to Constraint Set (3), the index of the workstation that task j is assigned is no smaller than those of its immediate predecessors. The assignment restrictions are given by Constraint Set (4).

We hereafter refer to Constraint Sets (1) through (4) as $x \in X$.

A solution s in set X is called efficient if there is no other solution t in set X with $CT^t \leq CT^s$ and $ND^t \leq ND^s$ with strict inequality holding at least once. The resulting objective vector (CT^s, ND^s) is said to be nondominated. Assume that there is a solution t such that $(CT^t, ND^t) \leq (CT^s, ND^s)$, i.e., $CT^t \leq CT^s$and $ND^t < ND^s$ or $CT^t < CT^s$and $ND^t \leq ND^s$. In such a case we say solution t dominates solution s and the objective vector (CT^t, ND^t) dominates the objective vector (CT^s, ND^s).

Our aim is to generate the set of nondominated objective vectors together with their corresponding efficient solutions. Our criteria CT and ND may be conflicting in the sense that reducing CT may lead to increases in ND and vice versa.

In the next two chapters, we present our procedures that return the exact set of nondominated objective vectors. Chapter 4 presents the classical approach that solves the problem sequentially, via integer models. Chapter 5 discusses the branch and bound algorithm that generates all nondominated objective vectors using a single tree.

CHAPTER 4

CLASSICAL APPROACH

In this chapter, we first discuss the generation of extreme nondominated objective vectors and then present the generation of all nondominated objective vectors.

4.1 THE EXTREME NONDOMINATED OBJECTIVE VECTORS

Consider the following problem:

Min CT

s.t. $x \in X$

Let CT^* be its optimal CT value.

CT^* is a valid lower bound on the CT values of all efficient solutions. However, any optimal solution to the above problem may not be efficient as there may exist alternative solutions having smaller ND values.

Among all alternative optimal solutions to the Min CT s.t. $x \in X$ problem, the one having minimum ND value can be found by setting $CT = CT^*$ and solving the below problem:

Min ND

s.t. $x \in X$

$CT = CT^*$

Hence, the efficient solution with CT value of CT^* can be found in two steps:

1. Solve Min CT s.t. $x \in X$.

   Let CT^* be the solution.

2. Solve Min ND s.t. $x \in X$ and $CT = CT^*$.

Instead of solving two optimization problems, one can modify the objective function as $CT + \varepsilon_{ND} ND$ for a sufficiently small value of ε_{ND}. The resulting problem is

Min $CT + \varepsilon_{ND} ND$

s.t. $x \in X$

ε_{ND} should be set small enough that the cycle time should not increase even for the largest possible value of the ND value, which is N. This follows

$$CT^* + \varepsilon_{ND} N \leq CT^* + 1 + \varepsilon_{ND} ND_{min} \quad (1)$$

where ND_{min} is the smallest possible value of ND, i.e., the number of tasks on the disrupted workstations.

Rearranging (1) gives

$$\varepsilon_{ND}(N - ND_{min}) \leq 1,$$

$$\varepsilon_{ND} \leq \frac{1}{N - ND_{min}}$$

In our experiments, we set ε_{ND} to $\frac{1}{N - ND_{min} + 1}$ and solve the following problem:

(P_1) Min $CT + \frac{1}{N - ND_{min} + 1} ND$

s.t. $x \in X$

Note that the optimal solution to P_1 is an efficient solution with the smallest CT value, and the resulting objective vector (CT, ND) is nondominated.

On the other hand, the efficient solution with the smallest ND value can be found through the following problem:

(P_2) Min CT

s.t. $x \in X$

$ND = ND^*$

Note that ND^* is the number of tasks over all disrupted workstations, hence it is known beforehand. This follows, an efficient solution with the smallest ND value can be found using a single optimization problem, P_2.

We provide an example to illustrate our decisions for P_1 and P_2. We take the data set from Rosenberg and Ziegler (1992) which is commonly used in the assembly line balancing literature. The sample instance has 25 tasks and the precedence network is as given below along with the task times:

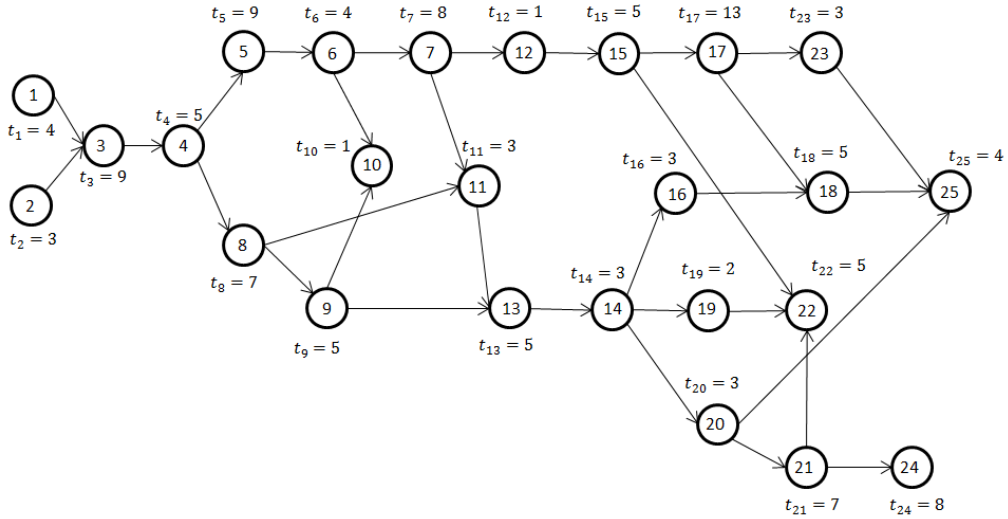

Figure 4.1 The precedence network of the sample problem instance

Let us assume that there are six workstations in the initial configuration, i.e., K'=6 and the tasks are assigned to the workstations so that the cycle time is minimized. We assume that a Type II simple assembly line balancing problem is solved in order to find the initial optimal assignment, i.e., the initial configuration is optimal for the single objective of cycle time minimization. The initial solution of this problem is given in the following figure:

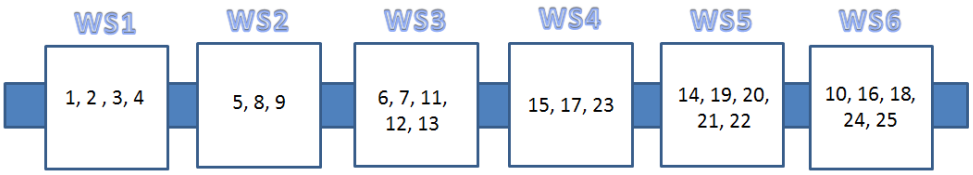

Figure 4.2 The initial assignment for the sample problem instance

The cycle time of the initial configuration is 21.

Now, assume that a disruption occurs on the workstations 2 and 3 and the assembly line needs to be rebalanced.

Let us solve P_1 and find the efficient solution with the smallest cycle time value. Before solving the problem, we should find the value of ε_{ND}. Note that ND_{min}=8 since there are eight tasks assigned to the workstation 2 and 3 initially. Hence,

$$\varepsilon_{ND} = \frac{1}{N - ND_{min} + 1} = \frac{1}{25 - 8 + 1} \cong 0.056$$

When P_1 is solved with ε_{ND} value of 0.056, the resulting nondominated objective vector is (CT, ND)=(32,12) and the optimal assignment is as follows:

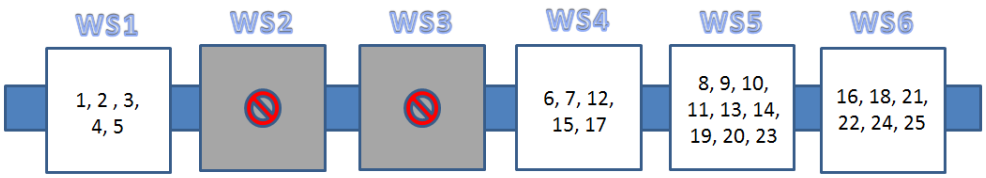

Figure 4.3 The optimal assignment for the efficient solution with the smallest CT

Note that tasks 5, 6, 7, 8, 9, 11, 12 and 13 are the eight tasks that should be assigned to a different workstation other than their original workstations. In addition to these eight tasks, tasks 10, 21, 22 and 23 are also disrupted in this efficient solution.

Let us also find an efficient solution with the smallest ND value by solving P_2. The resulting nondominated objective vector is (CT, ND)=(37,8) and the optimal assignments are as follows:

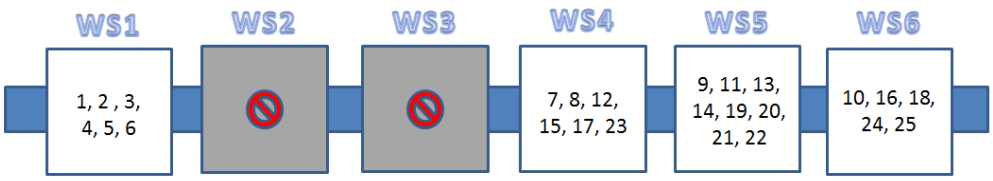

Figure 4.4 The optimal assignment for the efficient solution with the smallest ND

Note that tasks 5, 6, 7, 8, 9, 11, 12 and 13 are the only tasks that are disrupted in this solution. The other tasks remain in their original workstations.

Through the optimal solutions of P_1 and P_2, two efficient solutions with the smallest possible objective function values are found. We call these solutions as extreme efficient solutions and the corresponding objective vectors as extreme nondominated objective vectors.

The problem P_2 can be defined in a more efficient way using only the disrupted tasks. Recall that the optimal solution P_2 keeps the nondisrupted tasks at their original workstations.

The alternative formulation is as given below:

The parameters are set only for the tasks of the disrupted workstations. We let D denote the set of disrupted tasks and use the following parameters:

w_k= workload of workstation k in the original configuration (sum of the task times on workstation k in the original configuration)

E_i= the earliest workstation that task i can be assigned (the latest workstation that resides any nondisrupted predecessor of task i)

L_i= the latest workstation that task i can be assigned (the earliest workstation that resides any nondisrupted successor of task i)

The decision variables are also defined only for the disrupted tasks:

CT= cycle time

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

for i=1,...,$|D|$ and k=1,...,K

The objective function is the minimization of the cycle time.

Min CT

The constraints are defined only for the disrupted tasks.

$$\sum_{k=E_i}^{L_i} x_{ik} = 1 \qquad i = 1, \dots, |D|$$

$$\sum_{i=1}^{|D|} t_i x_{ik} + w_k \leq CT \qquad k = 1, \dots, K$$

$$\sum_{k=E_i}^{L_i} k x_{ik} \leq \sum_{k=E_j}^{L_j} k x_{jk} \qquad j = 1, \dots, |D| \; and \; i \in IP_j$$

$$x_{ik} = 0 \; or \; 1 \qquad i = 1, \dots, |D| \; and \; k = 1, \dots, K$$

4.2 ALL NONDOMINATED OBJECTIVE VECTORS

An optimal solution to the following constrained optimization problem is efficient (see Haimes et al. (1971) for the bicriteria problem):

Min $CT + \varepsilon_{ND} ND$

s.t. $x \in X$

$ND \leq nd$

where nd is between ND_{min} and N.

We use $\varepsilon_{ND} = \frac{1}{nd - ND_{min} + 1}$ in place of $\varepsilon_{ND} = \frac{1}{N - ND_{min} + 1}$, as the number of disrupted tasks is now bounded by nd.

Procedure 1 below generates all nondominated objective vectors by varying the nd value between N and ND_{min}.

Procedure 1. Generating All Nondominated Objective Vectors

Step 0. Find ND_{min}= the number of tasks on disrupted workstations.

$r = 0$

$nd = N$

$\varepsilon_{ND} = \frac{1}{nd - ND_{min} + 1}$

Step 1. Solve the following problem

Min $CT + \varepsilon_{ND} ND$

s.t. $x \in X$

$ND \leq nd$

Let the optimal solution be (CT^*, ND^*).

$r = r + 1$

Step 2. If $ND^* = ND_{min}$, then stop.

Otherwise, let $nd = ND^* - 1$.

Update ε_{ND} for the updated nd value, $\varepsilon_{ND} = \frac{1}{nd - ND_{min} + 1}$.

Go to Step 1.

Note that each step of the above procedure generates a nondominated objective vector together with its efficient solution. The procedure terminates when all r nondominated objective vectors are reached.

The following example illustrates the execution of Procedure 1.

Example 4.1: We use the instance given in Figure 4.1.

Step 0. We initialize the values of ND_{min}, r, nd and ε_{ND}.

ND_{min}=8

r=0

nd=25

$\varepsilon_{ND} = \frac{1}{25-8+1} \cong 0.056$

Step 1. We solve the model

$$\text{Min } CT + 0.056\, ND$$

$$\text{s.t. } x \in X$$

$$ND \leq 25$$

The optimal solution is $(CT^*, ND^*) = (32,12)$

$r = 1$

Step 2. We update the value of nd and ε_{ND}.

nd=12-1=11

$\varepsilon_{ND} = \frac{1}{11-8+1} = 0.25$

Step 1. We solve the model

Min $CT + 0.25ND$

s.t. $x \in X$

$ND \leq 11$

The optimal solution is $(CT^*, ND^*) = (33,11)$

$r = 2$

Step 2. We update the value of nd and ε_{ND}.

nd=11-1=10

$\varepsilon_{ND} = \frac{1}{10-8+1}$ =0.33

Step 1. We solve the model

Min $CT + 0.33ND$

s.t. $x \in X$

$ND \leq 10$

The optimal solution is $(CT^*, ND^*) = (34,9)$

$r = 3$

Step 2. We update the value of nd and ε_{ND}.

nd=9-1=8

$\varepsilon_{ND} = \frac{1}{8-8+1}$=1

Step 1. We solve the model

We solve the model

Min $CT + ND$

s.t. $x \in X$

$ND \leq 8$

The optimal solution is $(CT^*, ND^*) = (37,8)$

$r = 4$

Step 2. $ND^* = ND_{min}$, then stop.

The four nondominated objective vectors can be presented in the objective space as in Figure 4.5.

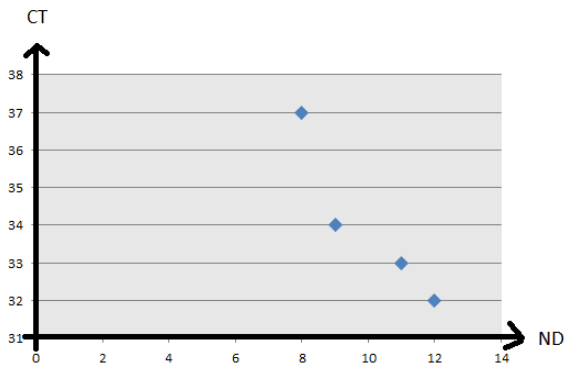

Figure 4.5 The efficient set for the sample problem instance

There can be at most $N - ND_{min} + 1$, hence N nondominated objective vectors, hence the procedure iterates polynomial number of times.

We now discuss the complexity of generating all nondominated objective vectors.

The problem of generating the efficient solution with the smallest CT value, i.e., Min $CT + \varepsilon_{ND} ND$ s.t. $x \in X$ reduces to the Type II assembly line balancing (ALB) problem when $\varepsilon_{ND} = 0$. Type II ALB problem is strongly NP-hard (see Baybars,1986) so is the problem of generating even a single nondominated vector. We hereafter refer to Procedure 1 as the classical approach, CA.

CHAPTER 5

BRANCH AND BOUND ALGORITHM

The complexity of our problem justifies the use of implicit enumeration techniques to arrive at the set of exact nondominated objective vectors. We present a branch and bound (BAB) algorithm that simultaneously generates all nondominated objective vectors.

We start with an approximate set of nondominated objective vectors. We update this initial set whenever a nondominated objective vector is found, i.e., we include the nondominated objective vector found to the set and we remove the objective vectors from the set if any of them is dominated by this newly added nondominated objective vector. We let Incumbent Set, IS denote the current set of nondominated objective vectors. IS gives the exact set of nondominated objective vectors and one efficient solution corresponding to each nondominated objective vector when the BAB algorithm terminates.

5.1 BRANCHING SCHEME

We start with a solution with all empty workstations. Starting from the first workstation, we form the complete solution each time adding a task to the current partial solution. Given a partial assignment with the first k workstations, we let S be the set of not yet assigned, i.e. unassigned, tasks. We say a task in S is eligible for the current workstation if all predecessor tasks are already assigned.

The tasks are indexed such that $i<j$ implies task i is not a successor of task j. While adding a task to the current workstation, we only consider the tasks having higher indices than the last assigned task. Our aim is to avoid the duplication of the partial solutions.

For each selected task, we open branches from the task nodes to each higher indexed eligible task and branch to a close node that represents closing the current workstation. If a close node is selected, we open branches using the result of the following theorem:

Theorem: There exists an efficient solution with no empty workstations.

Proof: Consider a solution s in which workstation k is empty and assume task q was assigned to workstation k in the original assignment. Let k_q be the workstation index of task q in the new solution. Two cases arise:

Case 1: $k_q < k$

Case 2: $k_q > k$

Case 1: $k_q < k$

Let S_{qk} be the set of successors of task q that are assigned to workstations k_q, k_{q+1},..., k-1 in the new assignment. Two subcases arise:

Case 1.1: $S_{qk} = \emptyset$

Case 1.2: $S_{qk} \neq \emptyset$ and task $s_q \in S_{qk}$ be a task with no successor in workstations k_q, k_{q+1},..., k-1.

Case 1.1: $S_{qk} = \emptyset$

Taking task q from k_q and assigning it to workstation k decreases ND by 1 and any single shift never increases CT. Hence, s cannot be efficient.

Case 1.2: $S_{qk} \neq \emptyset$

Taking task s_q from its current workstation and assigning it to workstation k never increases ND, as s_q was already disrupted. Moreover, such a single shift never inceases CT. Hence, the new schedule is no worse and s cannot be a unique efficient solution.

Case 2: $k_q > k$

Let P_{qk} be the set of predecessors of task q that are assigned to workstations k+1, k+2,..., k_q in the new assignment. Two cases arise:

Case 2.1: $P_{qk} = \emptyset$

Case 2.2: $P_{qk} \neq \emptyset$ and task $p_q \in P_{qk}$ be a task with no predecessor in workstations k+1, k+2,..., k_q.

Case 2.1: $P_{qk} = \emptyset$

Taking task q from k_q and assigning it to workstation k decreases ND by 1 and any single shift never increases CT. Hence, s cannot be efficient.

Case 2.2: $P_{qk} \neq \emptyset$

Taking task p_q from its current workstation and assigning it to workstation k never increases ND as p_q is already disrupted. Moreover, such a single shift never increases CT. Hence, the new schedule is no worse and s cannot be a unique efficient solution.

Note that in all cases, a new schedule is no worse than s in terms of CT and ND. Hence, a solution s that resides an empty solution cannot be a unique efficient solution.

Using the result of the above theorem, from a close node, we only open branches to task nodes and do not open a branch to another close node.

Figure 5.1 illustrates the branch and bound tree of the sample instance from Rosenberg and Ziegler (1992). The precedence network and task times are given in Figure 4.1.

Note that task 1 and task 2 have no predecessors and they are the only eligible tasks at level 1. Thus, task 1 and task 2 nodes take place at the first level of the BAB tree. First five levels of the BAB tree are as illustrated in Figure 5.1.

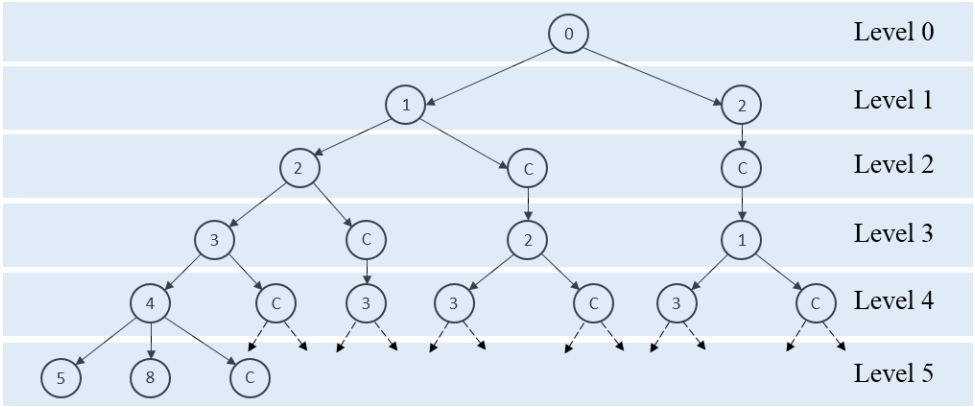

Figure 5.1 A representation of the BAB tree for the sample problem instance

For selected task 1 at level 1, we open two branches: one branch to task 2 which is the only eligible task and one branch to the close node. Note that if task 2 is selected at level 1, we just open one branch to the close node since a branch to task 1 cannot be opened due to the higher index rule and the other tasks are not eligible due to the precedence relationships.

For each node selected, there is a corresponding partial solution s and a corresponding objective vector (CT^s, ND^s). For each task node representing the addition of the task to the current workstation k, (CT^s, ND^s) is equal to $(max_{r \le k}\{w_r^s\},\ n_1^s + n_2^s)$ where

w_r^s= the workload of workstation r

$max_{r\leq k}\{w_r^s\}$ = the maximum workload already observed

n_1^s = the number of tasks that are already disrupted

n_2^s = the number of unassigned tasks that were placed in workstation 1,...,k-1 in the original assignment

We fathom the node if (CT^s, ND^s) is dominated by any objective vector in IS.

Example 5.1: Let us assume that there are three nondominated objective vectors in IS currently: (37,8), (34,9) and (33,11).

Let s_1 be a partial solution such that tasks 1, 2, 3, 4 and 5 are assigned to workstation 1, and tasks 6, 7, 8, 9, 11, 12 and 13 are assigned to the second nondisrupted workstation which is workstation 4. This workstation is not closed yet. Thus, the branching is to be continued to the task nodes for eligible tasks and also to the close node.

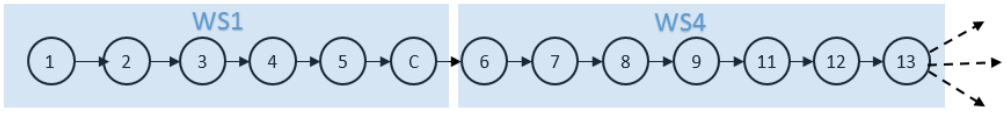

Figure 5.2 The representation of partial solution s_1

Note that $w_1^{s_1}$=30 and $w_2^{s_1}$=33. Thus, $max_{r\leq 2}\{w_r^{s_1}\}$ is equal to 33.

Tasks 5, 6, 7, 8, 9, 11, 12 and 13 are already disrupted since they were assigned to the disrupted workstations initially and tasks 1, 2, 3 and 4 were already assigned to workstation 1 initially. Thus, $n_1^{s_1}$=8 and $n_2^{s_1}$=0. All in all, (CT^{s_1}, ND^{s_1})=(33,8).

Now, let us open a new branch from node 13 to node 14, i.e., we assign task 14 to workstation 4. Let the new partial solution be s_2.

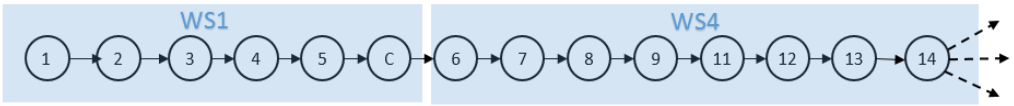

Figure 5.3 The representation of partial solution s_2

Note that $w_1^{s_2}$=30 and $w_2^{s_2}$=36 resulting in $max_{r\leq 2}\{w_r^{s_2}\}$=36 when task 14 is assigned to workstation 4. In addition, $n_1^{s_2}$ is set to $n_1^{s_1}+1$ since the original workstation of task 14 is workstation 5. Thus, (CT^{s_2}, ND^{s_2}) is equal to (36,9). We fathom this node since the nondominated objective vector (34,9) in IS dominates this partial solution. Hence, we remove this node from further consideration and we do not branch it any further.

We employ a depth first strategy due to its relatively low memory requirements. According to this strategy, we explore from the node having the smallest index. We continue branching until a node is fathomed or until a complete solution is reached. Note that a complete solution is obtained as soon as the assignments to the first $K-1$ workstations are complete since all of the unassigned tasks are to be assigned to K^{th} workstation. Once further branching is not possible, we backtrack to the previous level. We terminate whenever we reach level 0.

5.2 LOWER BOUNDS

For each close node generated, we let $(LB_{CT}^{s}, LB_{ND}^{s})$=$(CT^{s}, ND^{s})$ and we improve these bounds through the calculation of several lower bounds.

5.2.1 LOWER BOUND ON NUMBER OF DISRUPTED TASKS (LB_{ND})

We enhance the performance of $n_1^s + n_2^s$ in the close nodes using the cycle time values of the objective vectors in IS. We set an upper bound on the cycle time (UB_{CT}^s) that would lead to an efficient solution once s is branched further.

Example 5.2: Let us again assume that there are three nondominated objective vectors in IS currently: (37,8), (34,9) and (33,11).

Let s be a partial solution such that tasks 1, 2, 3, 4 and 5 are assigned to workstation 1, and tasks 6, 7, 8, 9, 11, 12 and 15 are assigned to the second nondisrupted workstation which is workstation 4. The assignment to this workstation is also completed so that the last selected node is a close node.

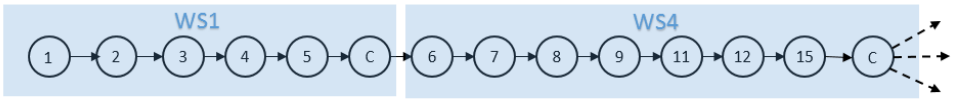

Figure 5.4 The representation of partial solution s

Note that tasks 5, 6, 7, 8, 9, 11 and 12 are already disrupted since their original workstations are disrupted workstations. Also, task 13 is an already disrupted task even though it has not been assigned yet. Thus, n_1^s=8. Moreover, n_2^s=2 since tasks 17 and 23 have not been assigned to workstation 4 although they were initially assigned to this workstation. Hence, ND^s=10. In IS, (34,9) and (33,11) are the two nondominated objective vectors such that ND^s=10 falls in between. Cycle time value should be at most 33 so that s would have a chance to be included in IS once it is a complete solution. Thus, UB_{CT}^s is set to 33.

The upper bound information on the cycle times allows us to define the earliest and latest workstations that a task can be assigned.

We let E_i and L_i denote the earliest and latest workstations that a task can be assigned. Given that the assignment of tasks to the first k workstations is completed, E_i and L_i are found accordingly:

$E_i = k + \left\lceil \frac{t_i + \sum_{j \in S, j \in P_i} t_j}{UB_{CT}} \right\rceil$ where P_i is the set of predecessors of i.

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

We increase LB_{ND} by 1 for each task $i \in S$ if the original workstation of task i is before E_i or after L_i.

Example 5.3: Let us again assume that there are three nondominated objective vectors in IS currently: (37,8), (34,9) and (33,11).

Let s_1 be a partial solution such that tasks 1, 2, 3 and 4 are assigned to workstation 1 and this workstation is not closed yet. Thus, the branching is to be continued to the task nodes for eligible tasks and also to the close node.

Note that (CT^{s_1}, ND^{s_1}) is equal to (21,8) since $w_1^{s_1}$=21 and $n_1^{s_1}$=8. Thus, there does not exist any nondominated objective vector in IS which dominates s_1. We can continue branching from the current node.

Let us select the close node as the next node and let the partial solution be s_2. We first let (CT^{s_2}, ND^{s_2})=(21,8). Then, the earliest and latest workstations for each unassigned task are found given that $UB_{CT}^{s_2}$=37 which is the corresponding upper bound to ND^{s_2}=8. Two of the unassigned tasks are detected to be disrupted since their original workstations are not in between the earliest and latest workstations to which they should be assigned:

- EW_{17}=5 and LW_{17}=6 whereas task 17 was assigned to workstation 4 initially

- EW_{23}=5 and LW_{23}=6 whereas task 23 was assigned to workstation 4 initially

Thus, ND^{S_2} is increased by 2. Since the new ND^{S_2} is 10, the corresponding $UB_{CT}^{S_2}$ value is updated to 33. For the updated upper bound, the earliest and latest workstations are also updated and task 22 is also detected to be disrupted.

- EW_{22}=6 and LW_{22}=6 whereas task 22 was assigned to workstation 5 initially

All in all, ND^{S_2}=11 since three more tasks will be disrupted if s_2 is branched further.

5.2.2 LOWER BOUND ON CYCLE TIME (LB_{CT})

We propose three lower bounds on the cycle time.

i. Lower Bound 1 ($LB1_{CT}$):

The lower bound is found through task preemption idea. The minimum cycle time is found by splitting the tasks equally among the workstations. The optimal assignment is then $\frac{\sum_{i=1}^{N} t_i}{K}$ given that N tasks are to be assigned to K workstations. Since any task cannot be split between workstations, CT should be greater than or equal to the task time of each task i. This follows that CT should be equal to at least the maximum of all task times. Hence, a valid lower bound on the cycle time is

$$LB1_{CT}=max\left\{\left\lceil\frac{\sum_{i=1}^{N} t_i}{K}\right\rceil, max_i\{t_i\}\right\}$$

This lower bound is presented in Klein and Scholl (1996).

For a given partial assignment, the lower bound is rearranged as follows:

$$LB1_{CT}=max\left\{\left\lceil\frac{\sum_{i\in S}t_i}{K-k}\right\rceil, max_{i\in S}\{t_i\}\right\}$$

ii. Lower Bound 2 ($LB2_{CT}$):

$LB2_{CT}$ uses the idea of the cardinality of the number of the tasks in any workstation. For N tasks and K workstations, there exists at least one workstation with $\left\lceil\frac{N}{K}\right\rceil$ or more tasks. If we assume workstation r resides $\left\lceil\frac{N}{K}\right\rceil$ or more tasks,

$$w_r \geq \sum_{i=1}^{\left\lceil\frac{N}{K}\right\rceil} t_{[i]} \text{ where } t_{[i]} \leq t_{[i+1]} \text{ for } i=1,\ldots, N\text{-}1$$

Since CT is equal to the maximum workload of all workstations, we can extend the above expression as follows:

$$CT \geq w_r \geq \sum_{i=1}^{\left\lceil\frac{N}{K}\right\rceil} t_{[i]} \text{ where } t_{[i]} \leq t_{[i+1]} \text{ for } i=1,\ldots, N\text{-}1$$

Hence, $\sum_{i=1}^{\left\lceil\frac{N}{K}\right\rceil} t_{[i]}$ is a valid lower bound on the cycle time.

$$LB2_{CT}=\sum_{i=1}^{\left\lceil\frac{N}{K}\right\rceil} t_{[i]} \text{ where } t_{[i]} \leq t_{[i+1]} \text{ for } i=1,\ldots, N\text{-}1$$

For a given partial assignment, the lower bound is rearranged as follows:

$$LB2_{CT}=\sum_{i=1}^{\left\lceil\frac{|S|}{K}\right\rceil} t_{[i]} \text{ where } t_{[i]} \leq t_{[i+1]} \text{ for } i=1,\ldots, |S|\text{-}1$$

This lower bound is also presented in Klein and Scholl (1996) with an extended version.

iii. Lower Bound 3 ($LB3_{CT}$):

Recall that $LB1_{CT}$ and $LB2_{CT}$ ignore the precedence relationship information. Recognizing this fact, we introduce a lower bound that splits the tasks into two subsets with reference to a particular workstation. The first subset, FS resides the unassigned tasks whose latest workstation is before or on workstation w. The second subset, SS, resides the unassigned tasks whose earliest workstation is after w. The maximum workloads due to FS and SS are $\left\lceil \frac{\sum_{i\in FS} t_i}{w} \right\rceil$ and $\left\lceil \frac{\sum_{i\in SS} t_i}{(K-k)-w} \right\rceil$, respectively. Hence,

$$CT \geq max\left\{ \left\lceil \frac{\sum_{i\in FS} t_i}{w} \right\rceil , \left\lceil \frac{\sum_{i\in SS} t_i}{(K-k)-w} \right\rceil \right\}$$

Accordingly,

$$LB3'_{CT}=max\left\{ \left\lceil \frac{\sum_{i\in FS} t_i}{w} \right\rceil , \left\lceil \frac{\sum_{i\in SS} t_i}{(K-k)-w} \right\rceil \right\}$$

We can improve $LB3'_{CT}$ by considering other unassigned tasks that are not in FS or SS.

Let $TS=S/\{FS \cup SS\}$.

The third set TS resides the unassigned tasks that are not covered by FS and SS and hence have not contributed to the cycle time. We distribute TS between all workstations preemptively while minimizing cycle time.

$LB3'_{CT} \times (K-k) - \sum_{i\in\{FS\cup SS\}} t_i$ is the total processing time that can be filled without increasing $LB3'_{CT}$, and $\sum_{i\in TS} t_i - \left(LB3'_{CT} \times (K-k) - \sum_{i\in\{FS\cup SS\}} t_i\right)$ is the minimum total amount required to improve $LB3'_{CT}$.

We can rearrange the latter as $\sum_{i\in S} t_i - LB3'_{CT} \times (K-k)$. This amount when distributed equally will increase the cycle time, smallest. The resulting lower bound, $LB3_{CT}$, is stated as follows:

$$LB3_{CT} = LB3'_{CT} + \left\lceil \frac{max\{0, \sum_{i \in S} t_i - LB3'_{CT} \times (K-k)\}}{K-k} \right\rceil$$

$LB3_{CT}$ is improved even further by only considering the workstations to which the tasks in TS can be assigned. In order to accomplish this, we find $E_{min} = min_{i \in TS}\{E_i\}$ and $L_{max} = max_{i \in TS}\{L_i\}$ and we modify the lower bound accordingly. $LB3_{CT}$ is

$$LB3'_{CT} + \left\lceil \frac{max\{0, \sum_{i \in S, i:E_i \geq E_{min}, L_i \leq L_{max}} t_i - LB3'_{CT} \times (L_{max} - E_{min})\ \}}{L_{max} - E_{min}} \right\rceil$$

For each close node, we first calculate $LB1_{CT}$ and then update (LB^s_{CT}, LB^s_{ND}) if $LB1_{CT} \geq LB^s_{CT}$ for the current partial solution s. If (LB^s_{CT}, LB^s_{ND}) is not dominated by any objective vector in IS, then we calculate $LB2_{CT}$ and we update (LB^s_{CT}, LB^s_{ND}) if $LB2_{CT} \geq LB^s_{CT}$. Note that $LB1_{CT}$ and $LB2_{CT}$ are rather simpler lower bounds and their computations do not require the earliest and latest workstation information. On the other hand, both LB_{ND} and $LB3_{CT}$ take into account the precedence relationships and the earliest and latest workstations for each unassigned task should be found prior to the calculation of these lower bounds. If (LB^s_{CT}, LB^s_{ND}) is not dominated by any objective vector in IS after the calculation of $LB2_{CT}$, we calculate LB_{ND} and we update (LB^s_{CT}, LB^s_{ND}) if there exists any task $i \in S$ such that the original workstation of task i is not in between E_i and L_i. If (LB^s_{CT}, LB^s_{ND}) is not dominated by any objective vector in IS, then we calculate $LB3_{CT}$ and we update (LB^s_{CT}, LB^s_{ND}) if $LB3_{CT} \geq LB^s_{CT}$. If (LB^s_{CT}, LB^s_{ND}) is dominated by any objective vector in IS after an update due to any lower bound, the current node is eliminated and there is no need to calculate other lower bounds.

5.3 DOMINANCE RULE

When closing a workstation, we use a dominance rule. Our dominance rule is based on comparing two partial solutions according to their objective vectors. These partial solutions include either the same set of tasks or one of the task sets is the subset of the other. If the set of tasks that have been assigned to the current partial solution is the same set or the subset of any previously stored partial solution from a common parent node, then these two partial solutions can be compared by our dominance rule.

Figure 5.5 below represents two partial solutions s_1 (left) and s_2 (right) emanating from a common parent node C.

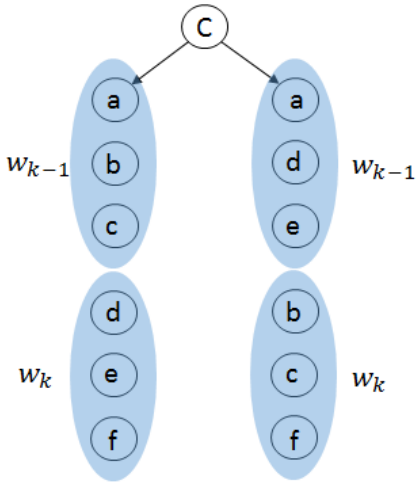

Figure 5.5 A representation of two partial solutions compared by the dominance rule

Note from the figure that s_1 is a previously stored partial solution whose set of tasks is completely the same with the current partial solution s_2. Also note that

there are k workstations open in both of the partial solutions and the assignments to only the last two workstations, i.e., workstation $k-1$ and workstation k, differ for these two partial solutions. Thus, it is enough to compare these two workstations while using our dominance rule. To generalize, we compare the last r workstations of two partial solutions while using the dominance rule given that the assignments to the first $k-r$ workstations are the same.

Formally, we let

s_1= the partial solution stored that is emanated from a parent node C

A_1= the set of tasks in s_1

s_2 = the current partial solution that is emanated from node C

A_2= the set of tasks in s_2

ND^{s_i} = the number of tasks disrupted for partial solution s_i

CT^{s_i} = the maximum of the maximum workload and the lower bound on the maximum loads of the unassigned tasks in partial solution s_i

We consider the unassigned tasks in CT^{s_i} computations as one of the next workstations may define the cycle time. For the lower bound on the maximum loads of the unassigned tasks, we simply use the lower bounds discussed in Section 5.2.2.

We compare s_1 and s_2 when $A_1 \supseteq A_2$.

We say s_1 dominates s_2 if $ND^{s_1} \leq ND^{s_2}$ and $CT^{s_1} \leq CT^{s_2}$ implying that a nondominated objective vector emanating from node C cannot be obtained by further branching s_2 once s_1 exists. If s_1 dominates s_2, we fathom the current node.

We illustrate the implementation of the dominance rule via two examples:

Example 5.4: Figure 5.6 represents two partial solutions s_1 (left) and s_2 (right) both emanating from node 0. If the tasks assigned so far are compared, it is observed that A_1=A_2. For s_1, $w_1^{s_1}$=30 and $w_2^{s_1}$=33, and $n_1^{s_1}$=8 and $n_2^{s_1}$=2 resulting in (CT^{s_1}, ND^{s_1})=(33,10). For s_2, $w_1^{s_2}$=33 and $w_2^{s_2}$=30, and $n_1^{s_2}$=8 and $n_2^{s_2}$=2 resulting in (CT^{s_2}, ND^{s_2})=(33,10). Since $ND^{s_1} \leq ND^{s_2}$ and $CT^{s_1} \leq CT^{s_2}$, s_1 dominates s_2 and there is no need to further branch s_2.

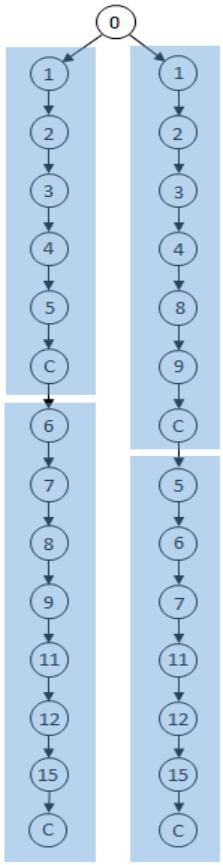

Figure 5.6 A representation for the comparison between two partial solutions with A_1=A_2

Example 5.5: Figure 5.7 represents two partial solutions s_1 (left) and s_2 (right) both emanating from node 0. If the tasks assigned so far are compared, it is observed that $A_1 \supset A_2$, i.e., task 11 has not been assigned yet in s_2. For s_1, $w_1^{s_1}$=30 and $w_2^{s_1}$=33, and $n_1^{s_1}$=8 and $n_2^{s_1}$=2 resulting in (CT^{s_1}, ND^{s_1})=(33,10). For s_2, $w_1^{s_2}$=28 and $w_2^{s_2}$=32, and $n_1^{s_2}$=8 and $n_2^{s_2}$=2 resulting in (CT^{s_2}, ND^{s_2})=(32,10). However, $LB1_{CT}^{s_2}$ is 33 which improves (CT^{s_2}, ND^{s_2}) to (33,10). Since $ND^{s_1} \leq ND^{s_2}$ and $CT^{s_1} \leq CT^{s_2}$, s_1 dominates s_2 and there is no need to further branch s_2.

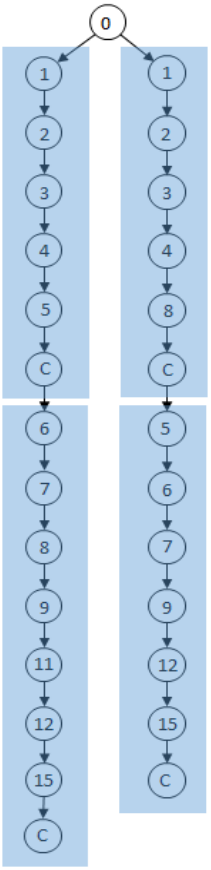

Figure 5.7 A representation for the comparison between two partial solutions with $A_1 \supset A_2$

Below we give the formal description of storing, comparing and deleting the partial solutions.

i. Storing partial solutions

To store partial solutions, the number of partial solutions stored so far for the common parents should be known. For each number of workstations to be compared (r), the number of partial solutions associated with a common parent should be kept separately.

Initialize the number of partial solutions associated with each common parent for each r to 0.

- If the current node is a close node and the number of opened workstations $k \geq 2$

  Set r=2.

  - While $r \leq k$
    - Set common parent node to the close node which represents closing the $(k-r)^{th}$ workstation if $k-r>0$, to node 0 if $k-r=0$.
    - Increase the number of partial solutions associated with this common parent by 1 for r.
    - Store the set of tasks assigned to the partial solution s.
    - Find CT^s and ND^s.
    - $r=r+1$.
  - End while

- End if

ii. Comparing partial solutions

- If the current node is a close node and the number of opened workstations $k \geq 2$

  Set $r = k$.

  - While $r \geq 2$

    - Set common parent node to the close node which represents closing the $(k-r)^{th}$ workstation if $k - r > 0$, to node 0 if $k - r = 0$.

    - Let the current partial solution associated with the common parent be s_2.

    - Check previously stored partial solutions associated with the common parent node.

    Two checks are done:

    Check 1: Same set of assigned tasks

    - If there is such a partial solution with the same set of assigned tasks

      $(A_1 = A_2)$

      - Decrease the number of partial solutions associated with the common parent by 1 for r.

      - If $ND^{s_1} \leq ND^{s_2}$ and $CT^{s_1} \leq CT^{s_2}$, s_1 dominates s_2. The current node is fathomed. Stop.

      - If $ND^{s_1} \geq ND^{s_2}$ and $CT^{s_1} \geq CT^{s_2}$, s_2 dominates s_1. Update ND^{s_1}=ND^{s_2} and CT^{s_1}=CT^{s_2}.

    - End if

- If there is not such a partial solution with the same set of assigned tasks, perform second check.

Check 2: Supersets of assigned tasks

- If there is such a partial solution with the super set of assigned tasks ($A_1 \supset A_2$)
  - If $ND^{s_1} \leq ND^{s_2}$ and $CT^{s_1} \leq CT^{s_2}$, s_1 dominates s_2. The current node is fathomed. Decrease the number of partial solutions associated with the common parent by 1 for r. Stop.
- End if

$r = r - 1$.

- End while

- End if

iii. Deleting partial solutions

Whenever a close node is removed from further consideration, all of the partial solutions associated with it are deleted due to the memory requirements.

5.4 THE INITIAL SET OF NONDOMINATED OBJECTIVE VECTORS

The nondominated objective vectors in IS provide upper bounds on the objective values and help removing an unpromising partial solution from further consideration. Thus, instead of starting with an empty IS, we start with an approximate set of nondominated objective vectors so that the BAB algorithm directs to more promising portions of the BAB tree from the beginning. When

forming this approximate set, we first generate initial feasible configurations with the minimum number of disrupted tasks in Step 0. In the following steps, we try to find other objective vectors with improved cycle time values by allowing more tasks to get disrupted.

Step 0: The disrupted tasks on the disrupted machines are assigned to other workstations in order to construct an initial feasible configuration. Three different procedures are used to obtain these initial assignments.

1. The disrupted tasks are assigned according to the optimal configuration found by the model generating the extreme efficient solution with the minimum number of disrupted tasks.

2. The disrupted tasks are assigned to the workstations with the minimum workload while satisfying the precedence relationships.

3. The disrupted tasks are randomly assigned to the workstations while satisfying the precedence relationships. This procedure is applied for 200 times.

Hence, 202 different initial configurations are created. The following steps are applied to each of these initial configurations individually in order to find candidate nondominated objective vectors with improved cycle time values.

In step 1, we change the workstations of tasks on the bottleneck workstation, i.e., workstation having the maximum workload, one by one and we add the nondominated objective vectors to IS if any of them is found and then we assign the tasks back to their current workstations. In Step 2, if at least one nondominated objective vector is added to IS, we only change the workstation of the task resulting in the best improvement in cycle time. If such an objective vector is not found in Step 1, this means that an improvement cannot be achieved by changing the workstation of only one task. Thus, in Step 3, we use a "look

ahead" approach by changing the workstation of a task on the bottleneck workstation even though no improvement is achieved and then we change the workstation of another task on the updated bottleneck workstation. We add the nondominated objective vectors to IS if any of them is found during this step and then we assign the tasks back to their current workstations. If at least one nondominated objective vector is added to IS, we change the workstations of the two tasks resulting in the best improvement in cycle time. If Step 3 also fails to find a nondominated objective vector by improving the cycle time value, we change the workstation of a randomly selected task which is currently assigned to the bottleneck workstation and then we return to Step 1.

Below is the pseudo code of our improvement step.

Initialize *the number of non-improving moves* as 0.

While *non-improving moves* $\leq$50

Step 1:

- For each task on the bottleneck workstation
    - Remove the task from the bottleneck workstation.
    - For each workstation satisfying the precedence relationships
        - Assign the task to this workstation.
        - Update the workload of each workstation.
        - Find CT and ND.
        - If none of the objective vectors in IS can dominate the current objective vector, add the objective vector to IS.
        - If the current objective vector dominates any of the objective vectors in IS, remove the dominated objective vectors from IS.

        - Assign the task back to its original workstation.
        - Update the workload of each workstation.
    - End for
- End for

If at least one objective vector is added to IS in step 1, update *the number of non-improving moves* as 0 and go to Step 2.

Otherwise, go to step 3.

Step 2: Select the objective vector with the minimum cycle time among all objective vectors added to IS in step 1. Update the assignments and workloads of each workstation accordingly. Update the bottleneck workstation. Go to Step 1.

Step 3:

- For each task on the bottleneck workstation
    - Remove the task from the bottleneck workstation.
    - For each workstation satisfying the precedence relationships
        - Assign the task to the workstation.
        - Update the workload of each workstation and update the bottleneck workstation.
        - Remove a task from the bottleneck workstation
            - For each workstation satisfying the precedence relationships
                - Assign the task to the workstation.
                - Update the workload of each workstation.

- Find CT and ND.
- If none of the objective vectors in IS can dominate the current objective vector, add the objective vector to IS.
- If the current objective vector dominates any of the objective vectors in IS, remove the dominated objective vectors from IS.
- Assign the task back to its original workstation.
- Update the workload of each workstation.

- End for
- Assign the task back to its original workstation.
- Update the workload of each workstation.

- End for

- End for

If at least one objective vector is added to the list in step 3, update *the number of non-improving moves* as 0 and go to Step 2.

Otherwise, go to step 4.

Step 4:

Update *the number of non-improving moves=the number of non-improving moves*+1.

Initialize $it=0$.

- While any task cannot be replaced and $it \leq 100$

- Update $it=it+1$.
- Select a random task.
- If the selected task is assigned to the bottleneck workstation
  - If there exists at least one workstation satisfying precedence relationships
    - Remove the task from the bottleneck workstation and assign it to any workstation satisfying precedence relationships.
    - Update the workload of each workstation.
  - End if
- End if

- End while

CHAPTER 6

COMPUTATIONAL EXPERIMENT

We perform a computational study to evaluate the performance of our BAB algorithm. We state our performance measures and compare the BAB algorithm and the CA using these measures. We make preliminary experiments to assess the effects of several parameters on the performance of the BAB algorithm before designing the main experiment. We make some preliminary runs also to interpret the power of the dominance rule, lower bounds and upper bounds.

In this section, we first give our data generation scheme and then we discuss the results of our computational study. While analyzing the results, we first state the performance measures and then we present the results of our preliminary and main experiments.

6.1 DATA GENERATION

We assume that the tasks are already assigned to the workstations before the disruption. Thus, the initial assignment of the tasks is one of the major inputs to the problem. In order to obtain this initial assignment, we take the well-known data sets from the simple assembly line balancing literature and we solve the Type II simple assembly line balancing problems. There are numerous data sets which are extensively used in computational experiments of the studies in the simple assembly line balancing literature. In this study, we conduct our main experiment on four of these data sets, from Günther et al. (1983) with N=35 tasks, from

Kilbridge and Wester (1962) with N=45 tasks, from Hahn (1972) with N=53 tasks and from Tonge (1961) with N=70 tasks. We take these data sets from the website http://www.assembly-line-balancing.de. We choose the data sets with different N values since the preliminary runs indicate that N has an effect on the performance of the BAB algorithm. These data sets include the task time information and the precedence networks; however, we only take the precedence networks and generate the task times using discrete uniform distribution. See Appendix for the precedence networks. We decide on the parameters of the uniform distribution according to the results obtained from the preliminary runs.

We choose these four data sets from the study of Scholl and Klein (1997). Scholl and Klein (1997) define a complexity measure to examine the influence of the precedence network on the performance of their algorithm. This complexity measure is called order strength (OS) and it is equal to the number of all precedence relationships divided by $N(N-1)/2$. They report the OS values for the precedence networks of Günther et al. (1983), Kilbridge and Wester (1962), Hahn (1972) and Tonge (1961) as 59.5%, 44.6%, 83.8% and 59.4%, respectively. Note that higher OS values indicate more intense precedence networks.

Preliminary runs reveal that the number of workstations in the old configuration (K'), the number of workstations in the new configuration (K) and the number of disrupted workstations $(K'-K)$ have an impact on the performance of the BAB algorithm. In order to capture the effects of these parameters, we use five different settings in our main experiment:

Table 6.1 Settings with K and K'

Setting	K	K'
1	8	12
2	10	12
3	10	14
4	5	6
5	5	7

The disrupted workstations are chosen arbitrarily, for each instance.

For each data set, we repeat the experiment under these five different settings. Hence, we obtain 4*5=20 combinations. Moreover, we generate 10 instances for each combination. All in all, our main experiment set has 200 problem instances.

The BAB algorithm is coded in C++ using Microsoft Visual Studio 2013. The mathematical models are solved by IBM ILOG CPLEX 12.6. The BAB algorithm and mathematical models are run on a computer with Intel(R)Core(TM)i7-4770S CPU @ 3.10 GHz, 16 GB RAM and Windows 7.

6.2 ANALYSIS OF THE RESULTS

We use four performance measures to evaluate the performance of the BAB algorithm:

1) Average Central Processing Unit (CPU) Time (in seconds)
2) Maximum Central Processing Unit (CPU) Time (in seconds)
3) Average number of nodes
4) Maximum number of nodes

We also use average CPU time and maximum CPU time to generate the set of all nondominated objective vectors when evaluating the performance of the CA.

We set a termination limit of one hour for the execution of instances using both the BAB algorithm and the CA; however, even if the execution of the BAB algorithm or the CA is terminated due to this limit, the results are included in the calculation of the performance measures.

For the preliminary experiments, we choose instances with N=35 and with N=53. We run our BAB algorithm for these instances under setting 1 (K=8 & K'=12), setting 2 (K=10 & K'=12) and setting 3 (K=10 & K'=14).

6.2.1 EFFECT OF PARAMETERS

We repeat our experiments for task times distributed by U[1,10] indicating low variability and by U[1,50] indicating high variability.

Table 6.2 and Table 6.3 immediately reveal that the instances with high task time variability are harder to solve compared to the instances with low task time variability. The number of nondominated objective vectors, the number of nodes and the CPU times all increase for the case when t_i~U[1,50]. For N=35 when K=8 & K'=12, the average number of nondominated objective vectors increases from 5.2 to 7.5, the average number of nodes increases from 18,759.8 to 44,353.2 and the average CPU time increases from 0.74 to 0.79 seconds when the variability in the task times increases. For N=53 when K=8 & K'=12, the average number of nondominated objective vectors increases from 8 to 11.2, the average number of nodes increases from 75,490.5 to 145,684.9 and the average CPU time increases from 2.05 to 2.92 seconds when the variability in the task times increases. Similar observations can be made for the other settings. As our concern is to solve harder problem instances, in our main runs we decide to generate task times from discrete uniform distribution U[1,50].

Table 6.2 The performances of N=35 and N=53, t_i~U[1,10]

		# of nondominated objective vectors		BAB			
				Number of Nodes		CPU Time	
Setting	N	Average	Maximum	Average	Maximum	Average	Maximum
K=8 K'=12	35	5.2	8	18,759.8	47,617	0.74	1.58
	53	8	12	75,490.5	213,908	2.05	7.75
K=10 K'=12	35	5.4	8	20,226.7	29,812	0.28	0.51
	53	7.1	10	105,807.4	312,339	1.59	3.65
K=10 K'=14	35	4.6	6	15,432.0	41,874	0.53	1.98
	53	7.9	13	143,004.1	732,862	2.85	11.28

Table 6.3 The performances of N=35 and N=53, t_i~U[1,50]

		# of nondominated objective vectors		BAB			
				Number of Nodes		CPU Time	
Setting	N	Average	Maximum	Average	Maximum	Average	Maximum
K=8 K'=12	35	7.5	10	44,353.2	129,769	0.79	1.58
	53	11.2	19	145,684.9	277,543	2.92	7.50
K=10 K'=12	35	10.2	12	66,577.8	117,286	0.60	1.45
	53	12.9	15	307,632.4	1,438,863	5.00	26.07
K=10 K'=14	35	9.1	12	55,651.5	120,667	0.80	1.72
	53	13.1	19	205,748.2	516,778	3.94	13.82

We continue our preliminary runs where task times are distributed by U[1,50].

6.2.2 EFFECT OF MECHANISMS

We also test the power of the dominance rule, lower bounds and upper bounds. We again use instances with N=35 and N=53 when K=8 & K'=12 and K=10 & K'=12 for this purpose.

i. Dominance Rules

We first investigate the power of the dominance rule. The following tables summarize the results of the preliminary runs when the dominance rule is used and is not used.

Table 6.4 The performance of the BAB algorithm with and without the dominance rule, N=35

		K=8 and K'=12		K=10 and K'=12	
		Average	Maximum	Average	Maximum
with dominance rule	CPU	0.79	1.58	0.60	1.45
	Nodes	44,353.2	129,769	66,577.8	117,286
without dominance rule	CPU	1.85	5.99	6.51	23.28
	Nodes	579,747.5	2,790,304	2,577,738.0	10,417,170

Table 6.5 The performance of the BAB algorithm with and without the dominance rule, N=53

		K=8 and K'=12		K=10 and K'=12	
		Average	Maximum	Average	Maximum
with dominance rule	CPU	2.92	7.50	5.00	26.07
	Nodes	145,684.9	277,543	307,632.4	1,438,863
without dominance rule	CPU	44.97	324.03	53.61	330.60
	Nodes	8,999,801	74,380,905	8,243,175	59,057,043

As Table 6.4 and Table 6.5 suggest, the effect of the dominance rule is extremely significant on the BAB algorithm. Both the average CPU times and the average

number of nodes reduce significantly when dominance rule is applied. For N=35 when K=8 & K'=12, the average CPU time decreases from 1.85 to 0.79 seconds and average number of nodes decreases from 579,747.5 to 44,353.2. When K=10 & K'=12, the average CPU time decreases from 6.51 to 0.60 seconds and average number of nodes decreases from 2,577,738 to 66,577.8. The results are even more drastic for N=53. When K=8 & K'=12, the average CPU time decreases from 44.97 to 2.92 seconds and average number of nodes decreases from 8,999,801 to 145,684.9. When K=10 & K'=12, the average CPU time decreases from 53.61 to 5.00 seconds and average number of nodes decreases from 8,243,175 to 307,632.4.

The results for the maximum CPU times and the maximum number of nodes also reveal the ability of the dominance rule to avoid extremely large CPU times and number of nodes for the worst cases. For example, the maximum number of nodes would be 59,057,043 resulting in 330.6 CPU seconds for N=53 when K=10 & K'=12 if no dominance rule was applied. On the other hand, it is reduced to 1,438,863 nodes and 26.07 CPU seconds for the same instance when the dominance rule is used.

ii. Lower Bounds

We test the effects of the lower bounds on the number of disrupted tasks and cycle time both separately and together.

In our tests, we do not consider simple lower bounds. We test the power of the lower bound on the number of disrupted tasks which uses the earliest/latest workstations information (LB_{ND}) and two lower bounds on the cycle time which use the cardinality of number of tasks in any workstation information ($LB2_{CT}$) and the earliest/latest workstations information ($LB3_{CT}$), respectively.

The results for the preliminary runs with and without the lower bounds are summarized in the following two tables.

Table 6.6 The performance of the BAB algorithm with and without the lower bounds, N=35

		K=8 and K'=12		K=10 and K'=12	
		Average	Maximum	Average	Maximum
with all LBs	CPU	0.79	1.58	0.60	1.45
	Nodes	44,353.2	129,769	66,577.8	117,286
without LB_{ND}	CPU	0.81	1.47	0.87	1.50
	Nodes	60,593.8	151,339	88,941.8	184,355
without $LB2_{CT}$ $LB3_{CT}$	CPU	0.74	1.48	0.64	1.39
	Nodes	44,409.7	130,033	68,181.4	117,437
without LBs	CPU	0.57	1.09	0.49	0.78
	Nodes	60,715.4	151,655	91,159.5	185,133

Table 6.7 The performance of the BAB algorithm with and without the lower bounds, N=53

		K=8 and K'=12		K=10 and K'=12	
		Average	Maximum	Average	Maximum
with all LBs	CPU	2.92	7.50	5.00	26.07
	Nodes	145,684.9	277,543	307,632.4	1,438,863
without LB_{ND}	CPU	38.64	125.39	11.89	33.06
	Nodes	1,128,062.5	2,858,333	570,313.4	1,624,420
without $LB2_{CT}$ $LB3_{CT}$	CPU	3.32	9.17	7.35	46.29
	Nodes	174,458.8	338,009	394,372.4	2,058,472
without LBs	CPU	20.88	53.07	16.84	94.43
	Nodes	1,262,645.3	2,963,328	914,740	3,539,538

Table 6.6 and Table 6.7 show that the lower bound on the number of disrupted tasks has a significant effect on the CPU time and the number of nodes. For N=35 when K=8 & K'=12, the average CPU time decreases from 0.81 to 0.79 seconds and average number of nodes decreases from 60,593.8 to 44,353.2. When K=10 & K'=12, the average CPU time decreases from 0.87 to 0.60 seconds and average number of nodes decreases from 88,941.8 to 66,577.8. The results are again more obvious for N=53. When K=8 & K'=12, the average CPU time decreases from 38.64 to 2.92 seconds and average number of nodes decreases from 1,128,062.5 to 145,684.9. When K=10 & K'=12, the average CPU time decreases from 11.89 to 5.00 seconds and average number of nodes decreases from 570,313.4 to 307,632.4.

The effect of the lower bounds on the cycle time is slightly significant when the number of nodes is compared. For N=35 when K=8 & K'=12, the average number of nodes is reduced from 44,409.7 to 44,353.2. When K=10 & K'=12, the average number of nodes is reduced from 68,181.4 to 66,577.8. For N=53 when K=8 & K'=12, the average number of nodes is reduced from 174,458.8 to 145,684.9. When K=10 & K'=12, the average number of nodes is reduced from 394,372.4 to 307,632.4. However, when the average CPU times are compared, the results are not in line with the ones obtained for the average number of nodes. For example, for N=35 when K=8 & K'=12, the average CPU time increases from 0.74 to 0.79 seconds when the lower bounds on the cycle time are used. One can be suspicious whether the computation of the lower bounds on the cycle time is worth or not; however, the results for N=53 show that the effect of these lower bounds might be more significant for problem instances with larger size. To illustrate, for N=53 when K=10 & K'=12, the average CPU time decreases from 7.35 to 5.00 seconds when these lower bounds are used.

In the computation of LB_{ND} and $LB3_{CT}$, the earliest/latest workstations should be found for each of the unassigned tasks. Thus, even if only one of the lower bounds is used, the earliest and latest workstations should be found. We also investigate

the situation where none of the lower bounds are used so that the earliest and latest workstation information is no longer necessary. From Table 6.6, although the number of nodes is significantly greater when the lower bounds are not used, the CPU times are smaller when K=8 & K'=12 and K=10 & K'=12 for N=35. To illustrate, the average number of nodes increases from 66,577.8 to 91,159.5 when none of the lower bounds are employed whereas the average CPU time decreases from 0.60 to 0.49 seconds when K=10 & K'=12. This may be because the earliest/latest workstations are not computed and it is not worth to use the lower bounds even if the computation time of the lower bounds takes less than a second since the total time itself is less than a second. On the other hand, we observe the effect of the lower bounds from the results of N=53 when K=8 & K'=12 and K=10 & K'=12. To illustrate, for N=53 when K=10 & K'=12, the average CPU time decreases from 16.84 to 5.00 seconds when all lower bounds are used.

iii. Upper Bounds

We make preliminary experiments to understand the effect of the initial set generated, so called the upper bounds. To do so, only one candidate nondominated objective vector is generated for the initial set by using a well-known heuristic called the Largest Candidate Rule that gives priority to the task having larger task time. Using this rule, we try to find a feasible solution for the theoretically minimum cycle time. If such a solution cannot be found due to the fact that the required number of workstations is greater than the actual number of workstations in the new configuration, the cycle time is increased by one and the rule is applied again. By this way, we end up with a candidate nondominated objective vector with a low cycle time; however, the number of disrupted tasks in this solution is usually very high resulting in a very poor initial set.

Table 6.8 The performance of the BAB algorithm with and without the upper bounds, N=35

		K=8 and K'=12		K=10 and K'=12	
		Average	Maximum	Average	Maximum
with UBs	CPU	0.79	1.58	0.60	1.45
	Nodes	44,353.2	129,769	66,577.8	117,286
without UBs	CPU	0.33	1.19	0.56	1.05
	Nodes	71,955.8	247,555	100,792.2	192,767

Table 6.9 The performance of the BAB algorithm with and without the upper bounds, N=53

		K=8 and K'=12		K=10 and K'=12	
		Average	Maximum	Average	Maximum
with UBs	CPU	2.92	7.50	5.00	26.07
	Nodes	145,684.9	277,543	307,632.4	1,438,863
without UBs	CPU	2.86	7.94	5.16	25.58
	Nodes	195,811.4	364,043	367,618.4	1,441,123

Table 6.8 and Table 6.9 suggest that the generation of the initial set reduces the number of nodes whereas whether it is worth or not, is not clear in terms of CPU times. To illustrate, for N=35 when K=8 & K'=12, the average number of nodes decreases from 71,955.8 to 44,353.2 whereas the average CPU time increases from 0.33 to 0.79 seconds when the initial set is generated. Another example can be given for N=53. When K=8 & K'=12, the average number of nodes decreases from 195,811.4 to 145,684.9 whereas the average CPU time increases from 2.86 to 2.92 seconds if the initial set is used. Although it seems it is not worth to generate the initial set for both N=35 and N=53, we suspect that its effect might be perceived for larger instances. Thus, we repeat the experiment for Tonge (1961) which has 70 tasks.

Table 6.10 The performance of the BAB algorithm with and without the upper bounds, N=70

		Setting 1 K=8 and K'=12		Setting 2 K=10 and K'=12	
		Average	Maximum	Average	Maximum
with UBs	CPU	439.00	1702.59	154.02	1095.41
	Nodes	13,905,229.3	55,262,946	9,857,298.0	36,236,721
without UBs	CPU	748.10	3600.00	175.33	1212.63
	Nodes	33,270,872	204,517,755	12,664,664	42,980,361

Table 6.10 reveals that the upper bounds actually have a significant effect on reducing the CPU times. The average CPU time decreases from 748.10 to 439.00 seconds and from 175.33 to 154.02 seconds when K=8 & K'=12 and K=10 & K'=12, respectively. Moreover, note that the termination limit would be exceeded for one of the instances generated for the setting with K=8 & K'=12 if the upper bounds were not employed.

6.2.3 MAIN EXPERIMENT

Based on the results of our preliminary runs, we ensure the power of the dominance rule, lower bounds and upper bounds, and continue our main experiment. As stated, we take four different data sets from the simple assembly line balancing literature and use their precedence networks as they are. We generate task times using discrete uniform distribution U[1,50] again based on the results of the preliminary experiment. We also use five different settings for K and K' combinations.

We first analyze the number of nondominated objective vectors and report the results in Table 6.11 and Table 6.12. Table 6.11 and Table 6.12 give the average and maximum number of nondominated objective vectors.

Table 6.11 The number of nondominated objective vectors for K=8 & K'=12, K=10 & K'=12 and K=10 & K'=14

	K=8 and K'=12		K=10 and K'=12		K=10 and K'=14	
	Average	Maximum	Average	Maximum	Average	Maximum
N=35	7.5	10	10.2	12	9.1	12
N=45	7.2	11	9.5	12	9	12
N=53	11.2	19	12.9	15	13.1	19
N=70	10	17	11.3	15	10.7	20

Table 6.12 The number of nondominated objective vectors for K=5 & K'=6 and K=5 & K'=7

	K=5 and K'=6		K=5 and K'=7	
	Average	Maximum	Average	Maximum
N=35	6.9	11	6.3	9
N=45	5.5	9	4.6	7
N=53	9.6	18	9.4	16
N=70	10.1	13	6.6	14

As can be observed from Table 6.11 and Table 6.12, the nondominated objective vectors for the instances with N=35 and N=45 are considerably less compared to the instances with N=53 and N=70. Note from Table 6.11 when K=8 & K'=12 that the average number of nondominated objective vectors is 7.5 and 7.2 for N=35 and N=45, respectively whereas there are 11.2 and 10 objective vectors for N=53 and N=70, respectively. Similar results can also be observed for the other settings. This is due to the fact that the number of nondominated objective vectors is bounded by $N - ND_{min} + 1$. We expect the number of nondominated objective vectors to increase when the number of tasks increases. On the other hand, for fixed number of workstations, increasing N values would lead to more tasks assigned to each workstation, hence to higher ND_{min} values. Since the number of

nondominated objective vectors has a reverse relationship with ND_{min}, the number of nondominated objective vectors might also decrease when N increases. For example, the average number of nondominated objective vectors decreases from 7.5 to 7.2 when N increases from 35 to 45 when K=8 & K'=12. However, since N grows more rapidly than ND_{min}, the set of nondominated objective vectors is expected to get larger for significant increments in N.

The effects of the number of workstations in the initial configuration (K') and the number of disrupted workstations ($K' - K$) can be also observed from Table 6.11 and Table 6.12. It can be noted that, for fixed K', smaller values of $K' - K$ lead to increases in the number of nondominated objective vectors. To illustrate, for N=35, the average number of nondominated objective vectors is 7.5 when K=8 & K'=12, and it is 10.2 when K=10 & K'=12. For N=45, the average number of nondominated objective vectors is 7.2 when K=8 & K'=12 and it increases to 9.5 when K=10 & K'=12. For N=53, the average number of nondominated objective vectors is 11.2 when K=8 & K'=12 and it increases to 12.9 when K=10 & K'=12. For N=70, the average number of nondominated objective vectors is 10 when K=8 & K'=12 and it increases to 11.3 when K=10 & K'=12. As mentioned above, the number of nondominated objective vectors is bounded by $N - ND_{min} + 1$ and ND_{min} is greater when K=8 & K'=12 since there are more disrupted workstations. This leads to a smaller set of nondominated objective vectors when more workstations are disrupted given that the initial configuration is the same.

For fixed $K' - K$, larger values of K' lead to increases in the number of nondominated objective vectors. It is already noted that when K=8 & K'=12, the average number of nondominated objective vectors is 7.5 for N=35. When K=10 & K'=14, the average number of nondominated objective vectors is 9.1. Likewise, when K=10 & K'=14, the average number of nondominated objective vectors increases from 7.2 to 9 for N=45, from 11.2 to 13.1 for N=53 and from 10 to 10.7 for N=70. This difference is again related with ND_{min}. For K'=14, there are less number of tasks assigned to each workstation when compared to K'=12. If the

same number of workstations are disrupted for K'=12 and K'=14, the ND_{min} value would be less for K'=14. Thus, for fixed $K' - K$, the number of nondominated objective vectors are expected to be smaller for smaller K' values. Same observation can also be made if the results obtained for K=10 & K'=12 and K=5 & K'=7 are compared.

One can also compare the cases with the same K values; however, it is not possible to draw a conclusion that holds for all combinations. To illustrate, let us compare K=10 & K'=12 with K=10 & K'=14. From Table 6.11, the average number of nondominated objective vectors is 10.2 when K=10 & K'=12, and it is 9.1 when K=10 & K'=14 for N=35. On the other hand, the average number of nondominated objective vectors is 12.9 when K=10 & K'=12, and it is 13.1 when K=10 & K'=14 for N=53. Since there are less tasks assigned to each workstation in the original assignment when K'=14, the number of nondominated objective vectors might be expected to decrease when a fixed number of workstations is disrupted. However, as there are more affected workstations, the number of nondominated objective vectors might increase as well.

We now discuss the performance of our BAB algorithm. We report the performance results in Table 6.13 and Table 6.14. The tables give the average and maximum number of nodes and CPU times of the BAB algorithm. The average and maximum CPU times by the CA are also included in the tables.

Table 6.13 The performance of the BAB algorithm and the CA when K=8 & K'=12, K=10 & K'=12 and K=10 & K'=14

		BAB				CA	
		Number of Nodes		CPU Time		CPU Time	
Setting	N	Average	Maximum	Average	Maximum	Average	Maximum
K=8 K'=12	35	44,353.2	129,769	0.79	1.58	20.79	43.67
	45	5,565,824.1	15,284,852	19.36	46.18	139.92	348.51
	53	145,684.9	277,543	2.92	7.50	22.37	45.77
	70	13,905,229.3	55,262,946	439.00	1702.59	802.90	3600.00
K=10 K'=12	35	66,577.8	117,286	0.60	1.45	84.42	135.57
	45	35,271,957.5	62,193,762	658.28	3600.00	316.97	819.61
	53	307,632.4	1,438,863	5.00	26.07	64.14	126.87
	70	9,857,298.0	36,236,721	154.02	1095.41	2626.31	3600.00
K=10 K'=14	35	55,651.5	120,667	0.80	1.72	47.10	80.74
	45	40,777,397.0	97,633,896	1018.07	3600.00	403.55	1585.27
	53	205,748.2	516,778	3.94	13.82	52.11	106.96
	70	7,258,017.8	26,547,649	87.46	337.94	2216.89	3600.00

Table 6.14 The performance of the BAB algorithm and the CA when K=5 & K'=6 and K=5 & K'=7

		BAB				CA	
		Number of Nodes		CPU Time		CPU Time	
Setting	N	Average	Maximum	Average	Maximum	Average	Maximum
K=5 K'=6	35	21,203.9	37,329	0.53	1.31	4.96	7.65
	45	907,845.4	6,900,911	2.65	15.79	8.03	24.59
	53	64,396.2	205,580	0.89	1.98	4.77	7.93
	70	6,599,851.8	27,131,040	24.72	73.07	16.27	33.34
K=5 K'=7	35	15,161.6	26,134	0.69	1.51	4.87	6.92
	45	1,174,489.2	6,335,691	3.55	14.21	9.66	62.61
	53	48,501.2	97,284	0.81	1.47	4.78	8.03
	70	2,123,071.0	8,519,811	10.72	34.94	14.59	28.04

As can be observed from Table 6.13 and Table 6.14, the CPU times and the number of nodes of the BAB algorithm increase significantly with increases in N. This is due to the increase in the number of nondominated objective vectors and effort spent to find each nondominated objective vector.

Note from Table 6.13 that, for the BAB algorithm, when K=8 & K'=12, the average CPU times are 0.79, 19.36, 2.92 and 439.00 seconds, and the average number of nodes are 44,353.2, 5,565,824.1, 145,684.9 and 13,905,229.3 for N=35, N=45, N=53 and N=70, respectively. When K=10 & K'=12, the average CPU times are 0.60, 658.28, 5.00 and 154.02 seconds, and the average number of nodes are 66,577.8, 35,271,957.5, 307,632.4 and 9,857,298.0 for N=35, N=45, N=53 and N=70, respectively. When K=10 & K'=14, the average CPU times are 0.80, 1018.07, 3.94 and 87.46 seconds, and the average number of nodes are 55,651.1, 40,777,397.0, 205,748.2 and 7,258,017.8 for N=35, N=45, N=53 and N=70, respectively. The average CPU times and the average number of nodes for K=5 & K'=6 and K=5 & K'=7 can be found in Table 6.14. Except for N=53, the results are in line with our expectation. The reason why the average CPU time and the average number of nodes are smaller than what was expected for N=53 is due to the intense precedence network of Hahn (1972). Since this instance has relatively more precedence relationships, there is less number of tasks considered at each level of the BAB because there are less tasks satisfying feasibility conditions. Two more exceptions are identified such that even N is smaller, the average CPU time and the average number of nodes are greater. For K=10 & K'=12, the average CPU times are 658.28 and 154.02 seconds, and the average number of nodes are 35,271,957.5 and 9,857,298.0 for N=45 and N=70, respectively. A similar result is obtained for N=45 and N=70 when K=10 & K'=14. These exceptions can again be explained by the structure of the precedence network. The precedence relationships for Kilbridge and Wester (1962) is relatively sparse and there are more tasks considered at each level of the BAB tree.

Recall that K' and $K' - K$ have an impact on the number of nondominated objective vectors, they also affect the CPU times and the number of nodes. For example, for N=45, the average CPU times are 19.36 and 658.28 seconds, and the average number of nodes are 5,565,824.1 and 35,271,957.5 when K=8 & K'=12 and K=10 & K'=12, respectively. This is an expected result since the number of nondominated objective vectors is greater for K=10 & K'=12. A similar observation can be made when K=8 & K'=12 and K=10 & K'=14 are compared. For N=45, the average CPU time and the average number of nodes are 1018.07 seconds and 40,777,397.0 when K=10 & K'=14, respectively. Both the average CPU time and the average number of nodes are greater when K=10 & K'=14 compared to K=8 & K'=12 since the set of nondominated objective vectors is larger when K=10 & K'=14. There is only one exception reported in Table 6.13 for N=70. The average CPU time and the average number of nodes when K=8 & K'=12 are greater compared to K=10 & K'=12 and K=10 & K'=14 for N=70; however, if two instances of when K=8 & K'=12 with maximum CPU times and number of nodes are not considered when taking the averages, the results are consistent with our expectations.

The observations from Table 6.13 and Table 6.14 suggest that K also has an effect on the performance of the BAB algorithm. Let us compare K=10 & K'=12 and K=5 & K'=7 where the number of disrupted workstations is the same but the number of workstations in the new configuration is half in the latter setting. For N=70, the average number of nodes is 9,857,298.0 and the average CPU time is 154.02 seconds when K=10 & K'=12 whereas they decrease to 2,123,071 and 10.72 seconds, respectively when K=5 & K'=7. This observation is true for all settings. This is an expected result since the size of the BAB gets smaller for smaller K values as we stop to branch once the assignment of tasks to the $K - 1$ workstations is completed.

Another observation regarding the performance of the BAB algorithm is that the results obtained by the BAB algorithm seem consistent since the differences

between the average and the maximum values of the CPU times are relatively small for almost all combinations reported in Table 6.13 and Table 6.14. The termination limit of one hour is only exceeded when K=10 & K'=12 and K=10 & K'=14 for N=45. If these instances are ignored, the average CPU time reduces to 331.43 seconds and the maximum CPU time reduces to 793.76 seconds when K=10 & K'=12, and the average CPU time reduces to 372.59 seconds and the maximum CPU time reduces to 1749.11 seconds when K=10 & K'=14.

We also investigate the effects that are analyzed for the BAB algorithm on the CA. The performance of the CA also deteriorates with increases in the N value. The increase with N values is more pronounced compared to the BAB algorithm, as in the CA the number of tasks increases the number of binary variables, exponentially. Note from Table 6.13 that, when K=8 & K'=12, the average CPU times are 20.79, 139.92, 22.37 and 802.90 seconds for N=35, N=45, N=53 and N=70, respectively. When K=10 & K'=12, the average CPU times are 84.42, 316.97, 64.14 and 2626.31 seconds for N=35, N=45, N=53 and N=70, respectively. When K=10 & K'=14, the average CPU times are 47.10, 403.55, 52.11 and 2216.89 seconds for N=35, N=45, N=53 and N=70, respectively. The average CPU time for N=53 is again smaller, which can be explained by its intense precedence network. Similar observations can be made for the settings with K=5 & K'=6 and K=5 & K'=7.

The K' and $K' - K$ values also affect the performance of the CA through their effect on the number of nondominated objective vectors. Since each of the nondominated objective vectors is found by solving an NP-hard problem in the CA, the effect of K' and $K' - K$ on the CPU time of the CA is more significant than that of the BAB algorithm. For example, for N=45, the average CPU times are 139.92, 316.97 and 403.55 seconds when K=8 & K'=12, K=10 & K'=12 and K=10 & K'=14, respectively. The average CPU time when K=8 & K'=12 is less than those of when K=10 & K'=12 and K=10 & K'=14 due to the fact that the set of nondominated objective vectors is smaller when K=8 & K'=12. The difference

between the average CPU times when K=5 & K'=6 and K=5 & K'=7 reported in Table 6.14 and when K=8 & K'=12, K=10 & K'=12 and K=10 & K'=14 reported in Table 6.13 are again significant. As in the BAB algorithm, K has an effect on the effort spent to find each nondominated objective vector in the CA. This is due to the exponential increase of the number of decision variables for increasing values of K.

We give the average CPU times for the BAB algorithm and the CA, in Table 6.15 and Table 6.16, for different settings.

Table 6.15 The average CPU times of the BAB algorithm and the CA when K=8 & K'=12, K=10 & K'=12 and K=10 & K'=14

	K=8, K'=12		K=10, K'=12		K=10, K'=14	
	BAB	CA	BAB	CA	BAB	CA
N=35	0.79	20.79	0.60	84.42	0.80	47.10
N=45	19.36	139.92	658.28	316.97	1018.07	403.55
N=53	2.92	22.37	5.00	64.14	3.94	52.11
N=70	439.00	802.90	154.02	2626.31	87.46	2216.89

Table 6.16 The average CPU times of the BAB algorithm and the CA when K=5 & K'=6 and K=5 & K'=7

	K=5, K'=6		K=5, K'=7	
	BAB	CA	BAB	CA
N=35	0.53	4.96	0.69	4.87
N=45	2.65	8.03	3.55	9.66
N=53	0.89	4.77	0.81	4.78
N=70	24.72	16.27	10.72	14.59

Note that in almost all combinations, the BAB algorithm produces smaller CPU times and the differences between the performances of the BAB algorithm and the CA become more significant as N increases. From Table 6.15, the average CPU time to generate all nondominated objective vectors is 0.79 seconds for the BAB algorithm and 20.79 seconds for the CA, 19.36 seconds for the BAB algorithm and 139.92 seconds for the CA, 2.92 seconds for the BAB algorithm and 22.37 seconds for the CA, 439.00 seconds for the BAB algorithm and 802.90 seconds for the CA when K=8 & K'=12 for N=35, N=45, N=53 and N=70, respectively. Under all of the other settings, the BAB algorithm generates the set of all nondominated objective vectors faster with three exceptions. For N=45, although the BAB algorithm is faster than the CA on the average when K=8 & K'=12, K=5 & K'=6 and K=5 & K'=7, the average CPU times are greater for the BAB algorithm when K=10 & K'=12 and K=10 & K'=14. The average CPU time for the BAB algorithm is 658.28 seconds whereas it is 316.97 seconds for the CA when K=10 & K'=12, and the average CPU time for the BAB algorithm is 1018.07 seconds whereas it is 403.55 seconds for the CA when K=10 & K'=14. When we investigate these results, we see that in the BAB algorithm, the termination limit is exceeded in one of the ten instances when K=10 & K'=12 and in two of the ten instances when K=10 & K'=14.

The main experiment shows that in vast majority of the instances, the BAB algorithm is superior to the CA. The outweighing performance of the BAB algorithm over the CA can be explained by its lower sensitivity to the effects of the problem size parameters and the number of nondominated objective vectors.

The exponential nature of the BAB search is somewhat dispelled by powerful lower bounding mechanisms and efficient dominance rules.

CHAPTER 7

CONCLUSIONS

In this thesis, we consider an assembly line rebalancing problem. We assume that there are disruptions on a particular number of workstations that makes the original task assignment plan infeasible to implement. Hence, the line has to be rebalanced, i.e., the tasks should be reassigned, considering only nondisrupted workstations and precedence relations. We consider a bicriteria problem of generating all nondominated objective vectors with respect to our efficiency measure and stability measure. We consider cycle time, hence maximizing production rate, as an efficiency measure. Our stability measure is the number of disrupted tasks assigned to different workstations than their original workstations.

To find the exact set of nondominated objective vectors, we propose two algorithms: classical approach (CA) and branch and bound (BAB) algorithm. The CA generates all nondominated objective vectors by successive solutions of mixed integer linear programs. The BAB algorithm generates the nondominated set simultaneously employing efficient branching scheme, bounding and dominance mechanisms.

The results of our computational experiment show that the lower bounding schemes and dominance mechanisms have significant effect on the performance of the BAB algorithm. Using those mechanisms, the BAB algorithm solves the instances with up to 70 tasks in reasonable times and performs superior to the CA.

To the best of our knowledge, we present the first optimization algorithm for the assembly line rebalancing problem. Future research may benefit from our results to fill the gaps in the related literature. Defining new efficiency and stability measures and finding efficient solutions that catch a trade-off between the defined measures might be an interesting research direction. One reasonable stability measure is the weighted number of different assignments between the original and new plans or the weighted distance between the original and new workstations. An efficiency measure may be related to the workload balancing between the workstations like the absolute or squared deviation of the workstation loads around a target workload.

Another worth-studying research direction might be developing optimization and approximation algorithms for the variants of the classical assembly line rebalancing problem. The variants may include, but not limited to, U-shaped assembly lines, mixed model lines, parallel lines and flexible assembly lines.

In future research, different types of disruptions like addition of a new workstation, partial disruption of some workstations might be considered.

REFERENCES

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

Baybars, I., 1986, A survey of exact algorithms for the simple assembly line balancing, *Management Science*, 32, 909-932.

Celik, E., Y. Kara and Y. Atasagun, 2014, A new approach for rebalancing of U-lines with stochastic task times using ant colony optimization algorithm, *International Journal of Production Research*, DOI: 10.1080/00207543.2014.917768

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

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

Grangeon, N., P. Leclaire and S. Norre, 2011, Heuristic for the re-balancing of a vehicle assembly line, *International Journal of Production Research*, 22, 6609-6628.

Günther, R. E., G. D. Johnson and R. S. Peterson, 1983, Currently practiced formulations for the assembly line balance problem, *Journal of Operations Management*, 3, 209-221.

Hahn, R., 1972, Produktionsplanung bei Linienfertigung, de Gruyter, Berlin.

Haimes, Y. Y., L. S. Ladson and D. A. Wismer, 1971, Bicriterion formulation of problems of integrated system identification and system optimization, IEEE Transactions on Systems Man and Cybernetics, SMC 1, 296-&.

Hoffmann, T. R., 1992, EUREKA: A hybrid system for assembly line balancing, *Management Science*, 38, 39-47.

Hwang, C. L. and K. Yoon, 1981, Multiple attribute decision making, Methods and Applications, Springer, New York.

Johnson, R. V., 1988, Optimally balancing large assembly lines with "FABLE", *Management Science*, 34, 240-253.

Kilbridge, M. D. and L. Wester, 1962, A review of analytical systems of line balancing, *Operations Research*, 10, 626-638.

Klein, R. and A. Scholl, 1996, Maximizing the production rate in simple assembly line balancing – A branch and bound procedure, *European Journal of Operational Research*, 91, 367-385.

Kottas, J. F. and H. S. Lau, 1973, A cost oriented approach to stochastic line balancing, *AIIE Transactions*, 8, 234-240.

Morrison D. R., E. C. Sewell and S. H. Jacobson, 2014, An application of the branch, bound, and remember algorithm to a new simple assembly line balancing dataset, *European Journal of Operational Research*, 236, 403-409.

Nourie, F. J. and E. R. Venta, 1991, Finding optimal line balances with OptPack, *Operations Research Letters*, 10, 165-171.

Rosenberg, O. and H. Ziegler, 1992, A comparison of heuristic algorithms for cost-oriented assembly line balancing, *Zeitschrift für Operations Research*, 36, 477-495.

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

Scholl, A., 1994, Ein B&B-Verfahren zur abstimmung von fiessbaendern bei gegebener stationsanzahl, *Operations Research Proceedings 1993*, Springer-Verlag, Berlin, 175-181.

Scholl, A. and R. Klein, 1997, SALOME: A bidirectional branch-and-bound procedure for assembly line balancing, *INFORMS Journal on Computing*, 9, 319-334.

Scholl, A. and R. Klein, 1999, Balancing assembly lines effectively – A computational comparison, *European Journal of Operational Research*, 114, 50-58.

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

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

Tonge, F. M., 1961, A heuristic program for assembly line balancing, Prentice Hall, Englewood Cliffs, NJ.

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

Zha, J. and J. J. Yu, 2014, A hybrid ant colony algorithm for U-line balancing and rebalancing in just-in-time production environment, *Journal of Manufacturing Systems*, 33, 93-102.

APPENDIX

PRECEDENCE NETWORKS

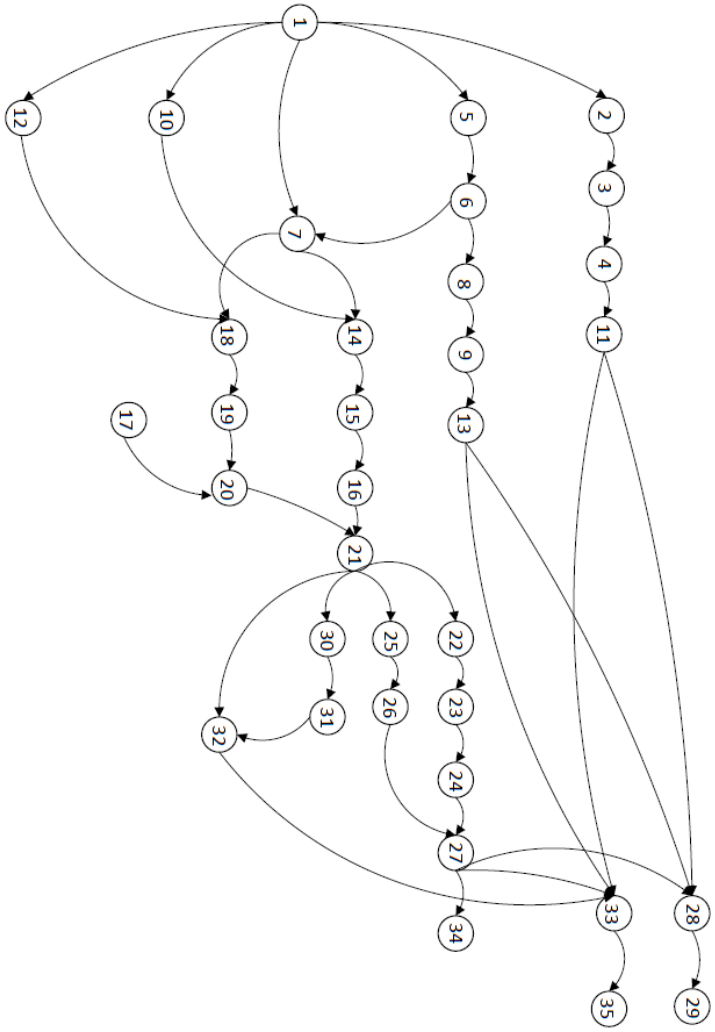

Figure A.1 Precedence network of Günther et al. (1983)

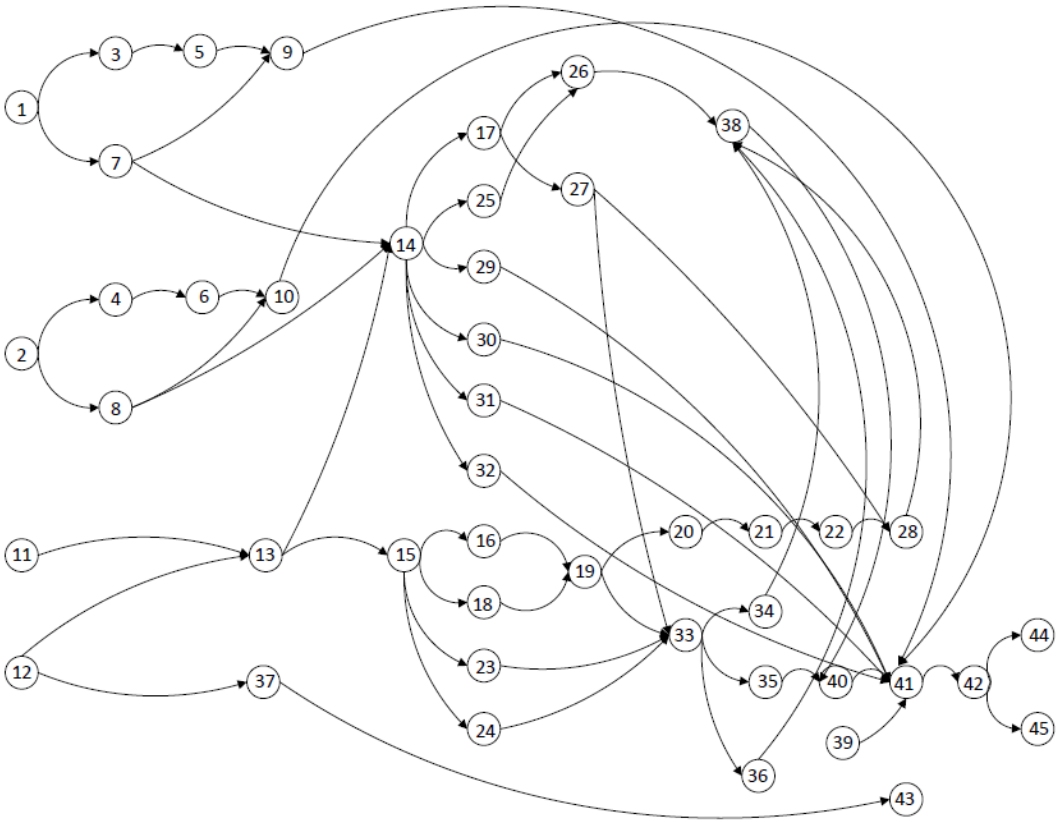

Figure A.2 Precedence network of Kilbridge and Wester (1962)

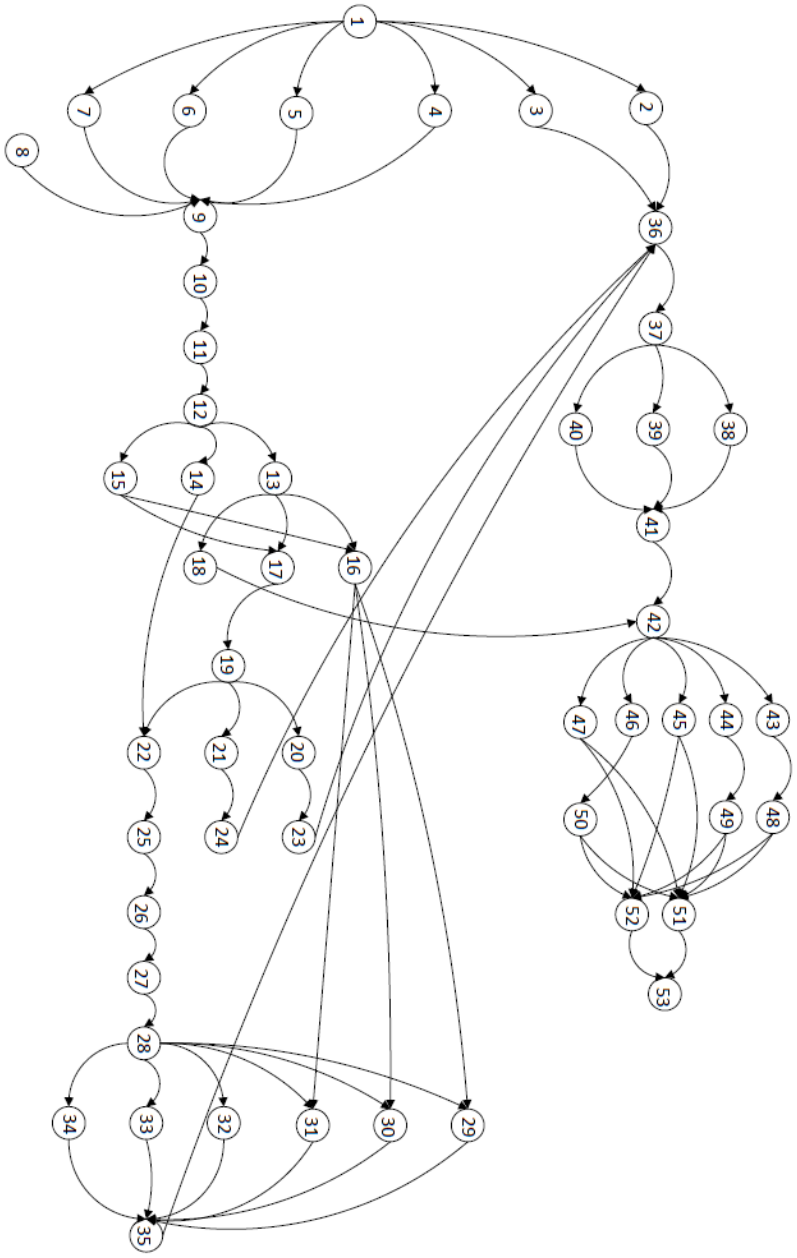

Figure A.3 Precedence network of Hahn (1972)

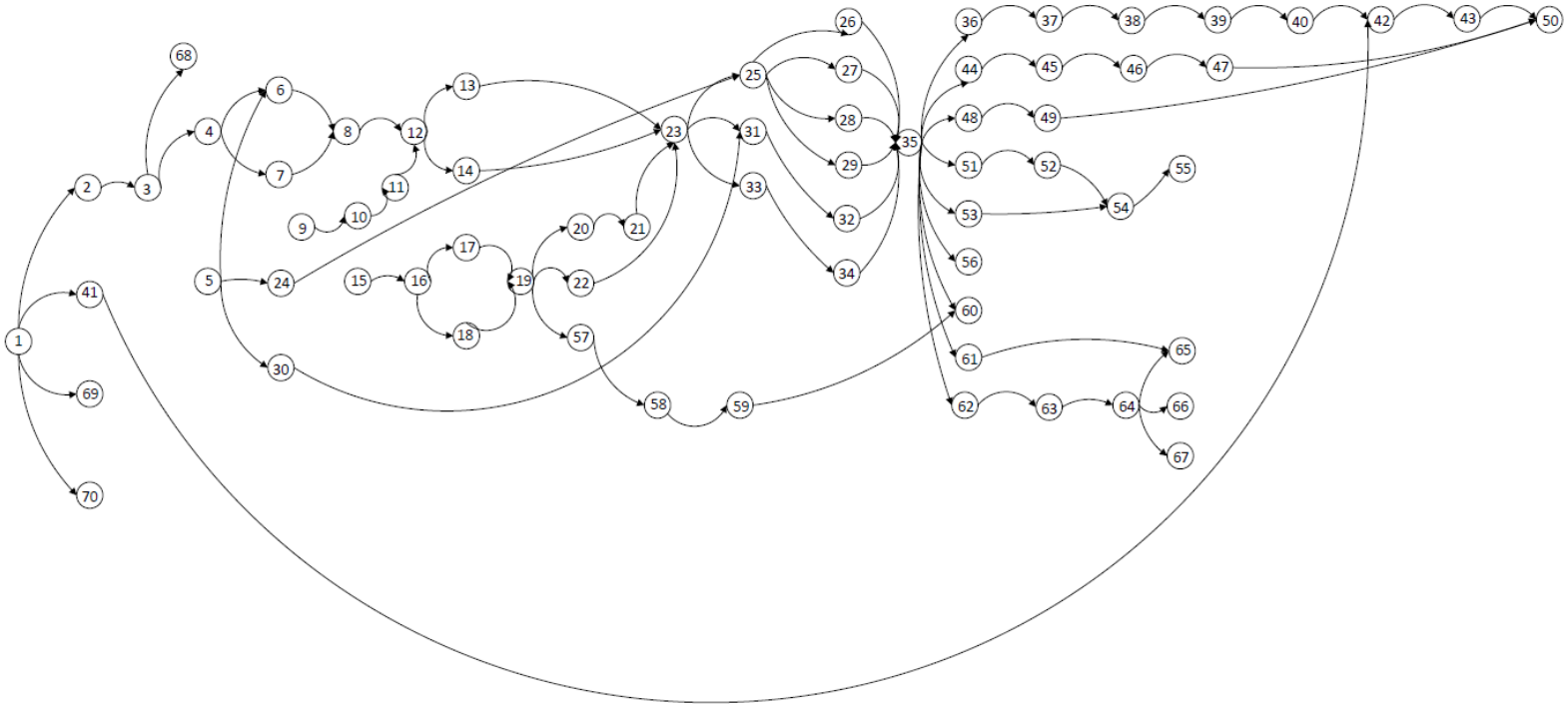

Figure A.4 Precedence network of Tonge (1961)