FLEXIBLE U-SHAPED ASSEMBLY LINE DESIGN PROBLEM

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

BY

AYŞE DİLEK OĞAN

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

JUNE 2013

Approval of the thesis:

FLEXIBLE U-SHAPED ASSEMBLY LINE DESIGN PROBLEM

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

Prof. Dr. Canan Özgen _____
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:

Asst. Prof. Dr. Cem İyigün _____
Industrial Engineering Dept., METU

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

Assoc. Prof. Dr. Ferda Can Çetinkaya _____
Industrial Engineering Dept., Çankaya University

Assoc. Prof. Dr. Mehmet Rüştü Taner _____
Industrial Engineering Dept., TED University

Asst. Prof. Dr. Mustafa Tural _____
Industrial Engineering Dept., METU

Date: _____

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 : Ayşe Dilek OĞAN

Signature :

ABSTRACT

FLEXIBLE U-SHAPED ASSEMBLY LINE DESIGN PROBLEM

Oğan, Ayşe Dilek
M.S., Department of Industrial Engineering
Supervisor: Prof. Dr. Meral Azizoğlu

June 2013, 46 pages

This study considers a U-shaped assembly line where the operators are allowed to work on both legs of the U-shaped line. We assume that the number of workstations and the cycle time are fixed. Each task uses a specified set of equipments where each type of equipment has a specified cost. The problem is to assign the tasks together with their equipments to the workstations so as to minimize the total equipment cost. We propose a branch and bound algorithm that uses efficient precedence relations and lower bounds. We find that the algorithm is able to solve moderate size problem instances with up to 21 tasks-10 workstations and 30 tasks-8 workstations in reasonable time.

Keywords: U-shaped Assembly Line, Equipment Assignment, Branch and Bound Algorithm

ÖZ

U-TİPİ ESNEK MONTAJ HATTI TASARIM PROBLEMİ

Oğan, Ayşe Dilek
Yüksek Lisans, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Meral Azizoğlu

Haziran 2013, 46 sayfa

Bu çalışmada operatörlerin hattın her iki tarafında da çalışabildiği U-Tipi montaj hatlarını ele aldık. Çevrim zamanı ve istasyon sayısı ile işlerin gerektirdiği ekipman setinin ekipman maliyetleriyle beraber verilmiş olduğu varsayıldı. İşlerin ve ekipmanlarının istasyonlara toplam ekipman maliyetini en aza indirecek şekilde atanması hedeflendi. Etkin eleme mekanizmaları ve alt limitler kullanan bir dal-sınır algoritması geliştirildi. Algoritmanın 21 iş-10 istasyon ile 30 iş-8 istasyon içeren orta ölçekli problemleri makul sürelerde çözebildiği görüldü.

Anahtar Kelimeler: U-Tipi Montaj Hattı, Ekipman Ataması, Dal-Sınır Algoritması

ACKNOWLEDGEMENTS

First of all, I would like to express my gratefulness to my supervisor, Prof. Dr. Meral Azizoğlu. Throughout the study, she gives valuable advices and motivates me in my hardest times. Without her support, this study would have never been completed.

I am also grateful to my parents Fatih Oğan and Gülsüm Oğan and my brother Murat Oğan for their love and endless care. They always trust in me and stand by me throughout my life.

I owe thanks to Çağrı Ağçalı for his positive attribute and precious advices. He motivated me with his good sense of humor.

I would like to send special thanks to Yunus Emre Has for his interest and encouragement. He was exposed to my complaints, yet he was always with me especially at my desperate times. His presence was really relaxing for me.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Assembly lines are flow oriented production systems where different parts of a product are assembled through a series of workstations. Those lines have great importance in the industrial production of highly standardized commodities.

In assembly line terminology, a task is defined as the smallest, indivisible work element. Each workstation performs a subset of all tasks and the product moves through the line, from one workstation to the next one via a material handling system. The tasks are assigned to the workstations while taking the precedence relations and product demand into account. Precedence relations define technological restrictions, i.e., some tasks cannot start before the completion of some others. To guarantee the satisfaction of the demand, a product is to be completed at the end of each specified period, so called cycle time and the sum of the operation times of the tasks in a workstation cannot exceed the cycle time.

In the literature, there are several classification schemes for assembly lines, like single models and multi models, straight lines and U-shaped lines, fixed number of workstations and variable number of workstations (design and balancing problems), equipment requirement (flexible lines) and no equipment requirement. In this study, we consider a single model, flexible U-shaped assembly line design problem.

In straight assembly lines, operators perform tasks on a continuous portion of the line. A task can be performed only if all its predecessors are completed. On the other hand, in U-shaped assembly lines a task can be performed either all its predecessors or all its predecessors are completed. Operators can handle only adjacent tasks in straight lines while they are also allowed to work across both legs of the line in U-shaped lines. Therefore, U-shaped assembly lines provide flexibility by providing more alternatives for the assignments of the tasks to the workstations. In U-shaped lines operators are expected to have a higher skill level and perform a wider range of tasks than in straight lines. This provides quick response to the changing environments due to the possibility of re-allocating the skilled operators. U-shaped lines never require higher number of workstations than straight lines since U-shaped lines allow both the forward and backward assignment of tasks to the workstations. Moreover, in U-shaped lines the area is shared by the operators. Thus, the operators can communicate and help each other which facilitate problem solving.

In today's competitive environment, assembly lines are to be flexible to respond changing customer demands quickly. Recognizing this fact, many companies switch from traditional straight lines to the U-shaped assembly lines and put just-in-time principles into practice.

The importance of the U-shaped assembly lines, has also triggered the advances in the research part. The researches related to the U-shaped assembly lines have been grown quickly in the recent years.

Assembly Line Balancing Problem (ALBP) is a decision problem of the assignment of the tasks to the workstations with regard to the pre-specified objective. U-shaped Assembly Line Balancing Problem (ULBP) is an extension of the traditional assembly line balancing problem. According to the objective functions, three types of assembly line balancing problems, namely Type I, Type II and Type E are considered in the literature. In Type I problems, the objective is to minimize the number of workstations which is equivalent to minimize the total slack time for a pre-determined cycle time. Type I problems usually arise when a new assembly line is to be designed. In Type II problems, the aim is to minimize the cycle time for a given number of workstations. Companies usually handle Type II problems when they want to increase their production rates. In Type E problems, the tasks require a subset of equipments where a subset is either known or a decision variable. In the literature, the types of equipments are usually taken as decision variable in Type E problems. The aim is to minimize the total equipment cost. Type E problems are also referred to as flexible assembly line design problems.

In this study, we consider a flexible U-shaped line where each task requires a specified set of equipments and the number of workstations is fixed. Our problem is to assign the tasks together with their equipments to the workstations so as to minimize the total equipment cost. To the best of our knowledge, our study is the first attempt for solving a flexible U-shaped line design problem.

This thesis includes five chapters and organized as follows. In Chapter 2, we define our problem, and then give our mathematical formulation. Finally, a brief review of the literature is presented. In Chapter 3, we introduce our branch and bound algorithm with the reduction and bounding mechanisms. We present our computational experiment and discuss its results in Chapter 4. We conclude in Chapter 5, by emphasizing the main results and offering suggestions for future research.

CHAPTER 2

PROBLEM STATEMENT

In this chapter, we first define our problem and present the mathematical formulation. We then define the complexity of our problem. Finally, we give a review of the related literature.

2.1. PROBLEM DEFINITION

Consider a single product that has to be assembled on a U-shaped assembly line having *K* workstations. In U-shaped lines, the workstations are arranged in such a way that two units can be assembled at the same cycle at different positions of the line. Figure 2.1 taken from Becker and Scholl (2006), depicts a U-shaped line.

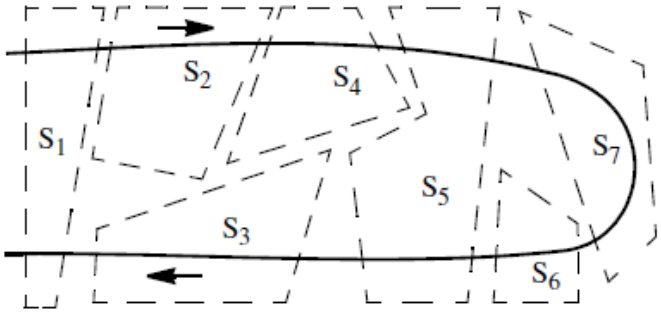

Figure 2.1 U-shaped assembly line

Note from the figure that, at workstation 1 the first task of a unit and the last task of another unit are executed at the same cycle. This is due to the fact that U-shaped lines not only allow assignment of the tasks whose predecessors are already assigned but also the tasks whose predecessors are to be finished until the product is returned to the same workstation for the second time. This type of stations where the same unit is handled in two different cycles is called crossover station.

The product has a target production rate that specifies the pre-specified cycle time, *CT* and requires *N* indivisible tasks each of which should be assigned to one of the workstations. There are *L* equipment types, a subset of which is required to process each task.

The problem is to determine the assignment of the tasks to each workstation together with the required equipment sets. Our objective is to minimize the total equipment cost overall the workstations.

The following additional assumptions are made:

- Processing times, equipment costs, precedence relations that specify the tasks, are known with certainty and are not subject to change, i.e., the system is deterministic and static.
- Equipments required for processing the tasks are known and each equipment has a specific cost. These parameters are deterministic and not subject to any change, as well.
- All tasks can be performed in all workstations and all equipments can be assigned to all workstations.

2.2. MATHEMATICAL FORMULATION

In this subsection, we give the mathematical representation of the problem that we defined in Section 2.1. We use the following indices, parameters and decision variables.

Indices:
i: task index, $i=1,2,\ldots,N$
l: equipment index, $l=1,2,\ldots,L$
k: workstation index, $k=1,2,\ldots,K$

Parameters:
t_i: processing time of task i
C_l: cost of equipment l
l_i: equipments required by task i
P_i: set of predecessors of task i
S_i: set of successors of task i
CT: cycle time

Decision Variables:

$$X_{ik} = \begin{cases} 1, \text{if task i is assigned to workstation k} \\ 0, \text{otherwise} \end{cases}$$

$$Z_{lk} = \begin{cases} 1, \text{if equipment l is assigned to workstation k} \\ 0, \text{otherwise} \end{cases}$$

$$Y_i = \begin{cases} 1, \text{if all predecessors of task i are assigned to the same or earlier workstations} \\ 0, \text{if all successors of task i are assigned to the same or earlier workstations} \end{cases}$$

The mathematical representation of our problem is as given below.

Mathematical Model:

$$Min \sum_{l=1}^{L}\sum_{k=1}^{K} C_l \times Z_{lk} \quad (1)$$

$$X_{ik} \leq \frac{\sum_{j\in S_i}\sum_{r\leq k} Xjr}{|S_i|} + Y_i \qquad \forall i,k \quad (2)$$

$$X_{ik} \leq \frac{\sum_{j\in P_i}\sum_{r\leq k} Xjr}{|P_i|} + (1 - Y_i) \qquad \forall i,k \quad (3)$$

$$\sum_{k=1}^{K} X_{ik} = 1 \qquad \forall i \quad (4)$$

$$\sum_{i=1}^{N} (t_i \times X_{ik}) \leq CT \qquad \forall k \quad (5)$$

$$X_{ik} \leq \frac{\sum_{l\in l_i} Z_{lk}}{|l_i|} \qquad \forall i,k \quad (6)$$

$$X_{ik} \epsilon \{0,1\} \qquad \forall i,k \quad (7)$$

$$Z_{lk} \in \{0,1\} \qquad \forall l,k \quad (8)$$

$$Y_i \in \{0,1\} \qquad \forall i \quad (9)$$

The objective function (1) minimizes the total equipment cost over all workstations.

Constraint sets (2) and (3) define the precedence relations. They ensure that a task can be assigned to a workstation if either all its successors or all its predecessors have been assigned to the same workstation or one of the earlier workstations. Whenever task *i* is assigned to a workstation, Y_i takes value 1 if all its predecessors are already assigned (*forward assignment*) and value 0 if all its successors are already assigned (*backward assignment*).

Constraint set (4) ensures that each task is assigned to exactly one workstation.

Constraint set (5) guarantees that the total load assigned to any workstation cannot exceed the cycle time. If the target production rate defines the cycle time, then the constraint implies that the target production is met.

Constraint set (6) ensures that if a task is assigned to a workstation, its equipment set should also be assigned to that workstation.

Constraints (7) and (8) are the binary assignment constraints on the variables that explain our decisions.

Y_i values in Constraint set (9) are binary indicator variables that create either / or constraints in the formulation.

2.3. COMPLEXITY OF THE PROBLEM

Our problem reduces to the classical Type I ALB problem when there is a single equipment type required by all tasks. Hence the complexity of the Type I ALB problem defines the complexity of our problem. We state this result formally through Theorem 1.

Theorem 1: Our problem is strongly NP-hard.

Proof: Assume a special case of our problem with a single equipment, say equipment r. That is $l_i=\{r\}$ for all i. For this special case, our objective function reduces to $C_r \sum_k Z_{rk}$ which is equivalent to minimizing $\sum_k Z_{rk}$. As all workstations use the same equipment r, $\sum_k Z_{rk}$ gives the number of workstations. The problem of minimizing the number of workstations given the cycle time is a type I ALB problem. Baybars (1986) shows that the Type I ALB problem is strongly NP-hard problem. This follows, our problem with an additional complexity brought by multi equipment types, is strongly NP-hard.

□

2.4. LITERATURE REVIEW

In the literature there are several studies concerning task or worker assignment in U-shaped assembly lines. There are also studies dealing with flexible assembly lines with equipment selection. However, the studies about tasks and equipment assignments in U-shaped assembly lines are quite scarce.

All U-Shaped Line Balancing problems consider Type I Line Balancing Problem, i.e., minimizing number of workstations subject to an upper bound on the cycle time. Some noteworthy of those studies are due to Miltenburg and Wijngaard (1994), Urban (1998), Aase et al. (2003), Gökçen and Ağpak (2006) and Erel et al. (2001).

Miltenburg and Wijngaard (1994) study the line balancing problem in U-Shaped lines. They propose a dynamic programming algorithm and find optimal solutions to the instances with up to 11 tasks. They solve the larger-sized problem instances using the ranked positional weight heuristic.

Urban (1998) proposes an integer programming formulation for the U-Shaped line balancing problem. Urban compares the heuristic results of Miltenburg and Wijngaard (1994) with the optimal solutions returned by the model. Urban (1998) introduces a phantom network that allows forward and backward assignment of tasks to workstations. The phantom network appends the original network to the end node, as depicted in Figure 2.2 for 4 tasks instance.

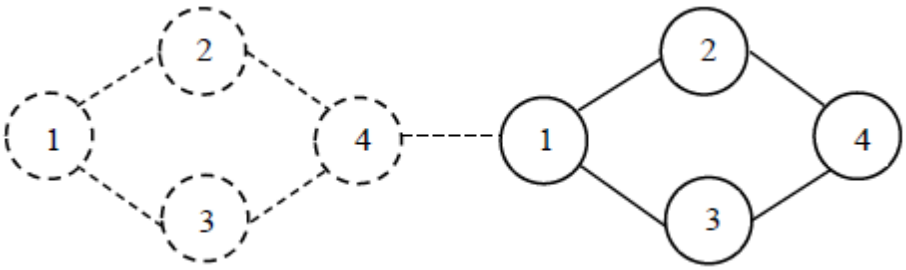

Figure 2.2 The precedence diagram with the phantom network

From any node in the network, assignments can be made forward through the original network or backward through the phantom network. To reduce the size of the model, Urban (1998) defines earliest and latest workstations that a particular task may be assigned.

They test their results on 25 random instances with 21, 28, 30 and 45 tasks. The results reveal that the integer program solves 23 instances in less than one hour and all instances in less than two hours, on a Pentium 90 MHz computer using CPLEX 3.0. They also show that, the heuristic procedure by Miltenburg and Wijngaard (1994) returns optimal solutions only 11 out of 25 instances.

Aase et al. (2003) develop a branch and bound (B&B) procedures for the deterministic, single-model, paced SULB problem. They use three branching strategies as deep-sea-troll, laser-type and best-first and refer the B&B algorithms as U-OPT1, U-OPT2 and U-OPT-3, respectively. They develop two mechanisms to improve the efficiency of their algorithms. The paired tasks lower bound is based on the general bin-packing idea and focuses on problems where on average one or two tasks are assigned to each workstation. The immediate task assignment dominance rule allocates a workstation to a single task that cannot be grouped with any other.

Their experimental set includes thousands of problem instances including 64 instances from Talbot (1986) and 25 instances from Urban (1998). The results on a 400 MHz PC with 256 MB RAM have revealed that U-OPT1 performs significantly faster than U-OPT2 and slightly faster than U-OPT3. The best of U-OPT1, U-OPT2 and U-OPT3, called U-OPT, is compared with the ULINO algorithm of Scholl and Klein (1999) and the IP procedure of Urban (1998) and U-OPT is shown to be significantly superior in terms of solution speed and solution quality. The newly developed fathoming rules especially Paired Task lower bound are responsible for this superior performance. It is observed that the paired tasks lower bound is effective in verifying the optimality of the solutions whereas the immediate task assignment rule is helpful in reducing the problem sizes.

Gökçen and Ağpak (2006) develop a goal programming model for the deterministic, simple U-shaped line balancing problem based on the integer programming formulation developed by Urban (1998) and the goal model of Deckro and Rangachari (1990). The model is the first multi-criteria decision making approach to the U-shaped line and provides increased flexibility to the decision maker when several conflicting goals are to be considered simultaneously.

They propose a preemptive approach where the goals are defined by their priorities where the higher priority goals are satisfied first. They illustrate their model on 3 problem instances with 11, 20 and 30 tasks and having three goals with the following priority levels.

Priority Level 1- The number of workstations should not exceed 3
Priority Level 2- The cycle time should not exceed 15
Priority Level 3- The number of tasks per workstations should not exceed 3

Their objective is to minimize the deviations from the specified goal values. The problems are solved using GAMS on a Pentium 4-2.0 GHz computer. The results reveal that all the goals cannot be satisfied simultaneously. It also shows that, priority 1 and 2 goals, and priority 1 and 3 goals are conflicting goals. They observe that as the conflicting goals are tried to be satisfied the number of iterations increases correspondingly.

Erel et al. (2001) develop a new simulated annealing (SA) -based algorithm to minimize the number of workstations in a U-shaped assembly line. The proposed algorithm consists of two components: a solution generator module and SA module. Solution generator module generates a new (better) solution by relaxing the cycle time constraint. Then, SA module takes this solution and obtains feasible task assignments with the objective of minimizing the maximum station time. These stages are repeated until prespecified time limit is exceeded, the optimal solution is obtained or a feasible allocation cannot be reconstructed by the SA module (in that case the previous feasible solution is returned).

The performance of the algorithm is compared with the DP algorithm of Miltenburg and Wijngaard (1994), IP formulation of Urban (1998) and several heuristic procedures. The results of the computational experiments on the benchmark problem instances taken from the literature have revealed that the algorithm returns superior solutions than the previous heuristic algorithms and in many instances the optimal solutions are reached. Moreover the solutions, on a Pentium II 266 kHz machine with 32 MB RAM, are obtained much quicker than the optimization algorithms. This computational success is to be attributed to the intelligent search mechanism of a large solution space.

Recently due to the fast-changing customer demands, flexible assembly lines having equipment alternatives, have gained considerable importance. The flexible assembly line problems consider equipment selection and assignment decisions in addition to the classical task assignment decisions, hence it offers more challenge for the researches. Despite its practical importance and theoretical challenge the research on the flexible assembly lines is quiet scarce. The most noteworthy of those studies are due to Bukchin and Tzur (2000), Buckhin and Rubinovitz (2003), Barutçuoğlu and Azizoğlu (2011) and Ege et al. (2009).

Bukchin and Tzur (2000) study a flexible line design problem where each task requires a single equipment and there are several equipment alternatives with their specified costs. They assume a single equipment can be assigned to each workstation and the cycle time is specified by the production rate. Their objective is to assign the tasks to the workstations together with their equipment so as to minimize the total equipment cost. They develop a

frontier search branch and bound algorithm which is capable of solving moderate problem sizes with up to 30 tasks and 5 workstations using Pentium II 266 MHz processor. The effectiveness of the algorithm is increased by using of some dominance properties and powerful lower bounds. They offer a heuristic branch and bound procedure, for the larger sized problem instances. Their procedure eliminates the nodes that are unlikely to produce the optimal solution in the earlier steps. It uses a control parameter K that establishes the trade-off between solution quality and solution speed. As the value of K decreases, the gap between the optimal solution and heuristic solution decreases, however at an expense of the increased CPU time. Bukchin and Tzur (2000) recommend starting with large K and gradually decreasing it till the required time or quality accuracy is achieved. The heuristic procedure could solve the problems that could not be solved by the exact branch and bound algorithm.

Buckhin and Rubinovitz (2003) study the assembly line design problem with station paralleling and equipment selection. The system studied includes several equipment types as well as human operators that are able to perform the assembly operations and paralleling of stations is allowed. In such systems the associated cost is highly dependent on the number of parallel stations due to the equipment duplication. Therefore, Buckhin and Rubinovitz (2003) first analyze the parallel station problem and then add weights (also considered as control parameters) associated with the system costs to the objective function. Different line configurations such as human intensive and capital intensive assembly line systems are obtained by changing the weights. Then, they show the parallel station problem is similar to the assembly design problem with equipment selection, discussed in Bukchin and Tzur (2000). Furthermore, they reveal the former problem is a special case to the latter and it can be solved by the branch and bound algorithm developed by Bukchin and Tzur (2000). Finally, the combined problem is addressed and solved by the same procedure. The results show that parallel stations are required especially when a good balance is difficult to obtain due to small cycle time or low flexibility in the assembly process. It is also shown that there is a relationship between the paralleling limitation (maximum number of stations in parallel in a stage) and the balancing improvement. 288 problem instances are solved and it is seen that significant contribution (8.4% improvement on average) is obtained by adding the first parallel station. The marginal improvement decreases to 0.95% and 0.26% for adding the second and third parallel stations, respectively. The B&B algorithm performance is highly dependent on the paralleling limitation parameter. This parameter is to be selected low (one or two) for practical problems. In situations where many parallel stations are needed due to long cycle times, it is reasonable to solve this part of the problem separately without combining additional task with short task times into such stations.

Barutçuoğlu and Azizoğlu (2011), study the flexible assembly line design problem with equipment alternatives and fixed number of workstations. They assume that the task times and equipment costs are correlated in the sense that cheaper equipment never gives smaller task times. They aim to minimize the total equipment cost given the cycle time and develop a branch and bound algorithm that uses powerful lower bounds and reduction mechanisms. They utilize two lower bounds; the first lower bound is related to the number of workstations (feasibility) while the other one considers the total equipment cost (optimality) and adapted

from Bukchin and Tzur (2000). The results of their extensive computational study show that the algorithm can solve large-sized problem instances with up to 80 tasks with 6 workstations and 70 tasks with 8 workstations in two hours on Intel Core 2 Duo, 2.33 GHz, and 980 MB of RAM PC.

Ege et al. (2009) consider the assembly line balancing problem with parallel workstations at each stage. They assume each task requires a specified equipment that should be available in all parallel workstations of the stage to which the task is assigned. Their aim is to assign the tasks to the stages so as to minimize the sum of station opening and equipment costs and meeting the target production rate. They propose two branch and bound algorithms; one for the optimal solutions and one for the high quality heuristic solutions. They find that the exact branch and bound algorithm could solve instances with up to 50 tasks in a termination limit of three hours on a Pentium III 550 MHz PC with 64 MB. At the same termination limit, their heuristic branch and bound algorithm could solve problems with up to 70 tasks. Moreover they observe that the heuristic branch and bound procedure gives many optimal solutions in reasonable time especially when the equipment cost is low.

Simaria et al. (2007) study the worker allocation problem in flexible U-shaped assembly lines. They make a distinction between workstation (physical place where the task is performed) and operator (the person who performs the task). They assume that the workstations are already fixed and the equipment cannot be moved from the workstation. Their decision is on the number of operators that would cope with extended product ranges and uncertain demand.

They develop a heuristic based on the ant colony algorithm developed by Vilarinho and Simaria (2006). The algorithm has two decision levels. In Level 1, tasks and their equipments/tools are assigned to the workstations so as to cope with the worst case demand scenario. In level 2, the operators are assigned to the tasks/workstations according to the configuration defined in level 1. The second level is repeated for each demand scenario. They test the algorithm on 50 random instances on a 1000 MHz Pentium III computer using Visual C++. Their computational results reveal the satisfactory performance of the algorithm. They find that 33 out of 50 solutions are optimal and the solutions are found in less than 2 minutes.

Shewchuck (2006) studies worker allocation problem in lean U-shaped production lines. His problem differs from the classical worker allocation problem on assembly lines in three aspects: the task locations are fixed, walking times are considered and workers are allowed to cross the paths, i.e., there are no restrictions on which machines can be assigned to a given worker. The aim is to minimize the number of the workers on the line. His mathematical model allows any allocation of the workers to the machines as long as workers do not cross paths and considers the walking times where workers could follow circular paths and walk around other workers. Moreover, a simple heuristic algorithm is developed to find quick solutions to the worker allocation problem. The algorithm first sets the number of operators to its lower bound, and then increments it by 1 until a feasible solution is reached. Only few iterations are required as each additional worker provides a substantial capacity increase and

the number of workers is limited by the number of machines. The algorithm is coded in C language on an IBM RS/6000 Unix-based server while the mathematical model is coded by AMPL programming language. The problems are solved using CPLEX 9.0.0 on a 1.7 GHz PC. 480 instances are solved for small- medium- large line size and low-medium-high takt time. The mathematical model finds solutions to the problems with up to 20 machines in less than an hour. The heuristic algorithm returns optimal solutions to 420 out 480 problem instances. He finds that his lower bound provides excellent estimates on the number of the workers and finds optimal number of workers in 372 instances.

Sirovetnukul and Chutima (2009) deal with the multi-objective worker allocation problem of single and mixed-model assembly lines. Three objective functions; the number of workers, the deviation of operation times of workers and the walking time, are simultaneously minimized. They use two multi objective evolutionary algorithms, namely NSGA-II and COIN to define the pareto set, i.e., set of nondominated solutions. NSGA-II is an evolutionary optimization algorithm that searches only good solutions from the population and discards the solutions that are below average. It finds the pareto set in a single run. COIN algorithm solves single and multi-objective problems by using a generator that represents a probabilistic model of the required solution. It uses reward and punishment schemes for updating the generator. The performances of the algorithms are evaluated with respect to four performance measures: the convergence to the Pareto-optimal set, spread of their non-dominated solutions, ratio of non-dominated solutions and CPU time. The algorithms are coded by MATLAB R2008a and the instances are solved on an AMD Athlon™ 64 processor 3500+2.21 GHz PC with 960 MB DDR-SDRAM. The results on 7 to 297 tasks instances show that the algorithms produce almost the same number of workers and COIN performs better than NSGA-II in majority of the cases.

The basic assumption of assembly line balancing is that the task times are fixed. However, there may be different processing alternatives to process a task with different times and costs. Assigning only one worker to each workstation is another assumption of ALB in many studies. Yet, some tasks cannot be processed by only one worker. Recognizing these facts, Kara et al. (2010) study the problem of assigning tasks and resources to the workstations so as to minimize total cost and they refer this problem as resource dependent assembly line balancing problem. They propose an integer programming formulation for the classical straight lines and extend the model to U-shaped lines. They test the formulation on a total of 135 instances having 10, 20, 30, 40 and 50 tasks, using Gurobi LP/MIP Solver Engine V10.0 on an IntelCore2 Duo 2.00 GHz and 2 GB RAM computer. The results reveal that when the classical straight line is switched to the U-shaped configuration, a significant improvement in the total processing cost is obtained. The computational results show that the difficulty of the attaining optimal solutions to the models is sensitive to the number of tasks and many 50 task instances could not be solved in 3 hours.

CHAPTER 3

SOLUTION APPROACH

In this chapter we first present our branch and bound algorithm together with its reduction and bounding mechanisms. We then illustrate the algorithm on an example problem.

3.1. BRANCH AND BOUND ALGORITHM

The complexity of our problem justifies the use of an implicit enumeration technique like branch and bound algorithm. In this study, we propose a branch and bound algorithm to find the optimal assignment of the tasks together with their equipments to each workstation.

3.1.1 BRANCHING SCHEME

Given a partial sequence with first k workstations opened, let S be the set of not yet assigned tasks. We say a task in S is eligible to the current workstation if either of the following conditions holds.

i. If all predecessor tasks are already assigned
ii. If all successor tasks are already assigned

Among the eligible tasks, we select the one having the smallest lower bound for further branching.

We index the tasks such that $i < j$ implies task i is not successor of task j.

In order to avoid the duplication of the solutions, we open branches from a particular task j in three ways:

i. Branch from task j to the nodes of tasks having higher index
ii. Branch to the nodes of tasks that become eligible by the assignment of task j
iii. Branch to the nodes representing closing the current workstation

The nodes at level k, emanating from node (task) j, include all the nodes at level $k-1$ such that $i > j$ and the nodes that become eligible with the assignment of node j.

Consider the network depicted in Figure 3.1.

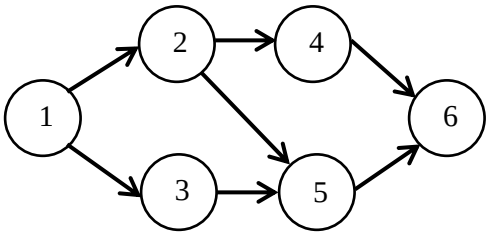

Figure 3.1 The precedence diagram

At level 1, tasks 1 and 6 are eligible, and level 2 nodes are as shown below.

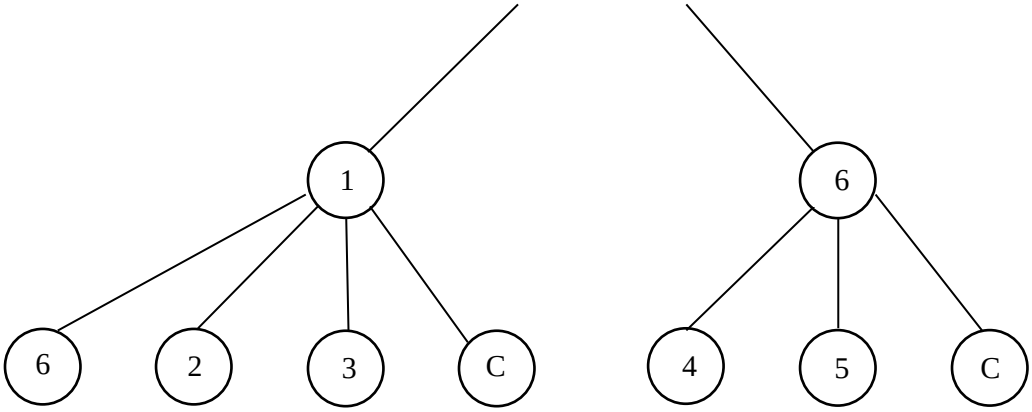

Figure 3.2 The branching tree

Note that from task 1, four nodes are opened. Node 6, is a type-1 branch, the branch at level 1 with higher index. Nodes 2 and 3 are type-2 branches that represent the tasks that become eligible with the assignment of task 1. Close node is a type-3 branch, represents closing that workstation and opening the second workstation.

From task 6, three nodes are opened. Nodes 4 and 5 become eligible with the assignment of task 6 and there is a close branch.

For the sake of completeness, we give the complete branch and bound tree in Figure 3.3, in which all feasible solutions are given. Each solution returned by tree is a different one and all feasible solutions without any exception, are returned.

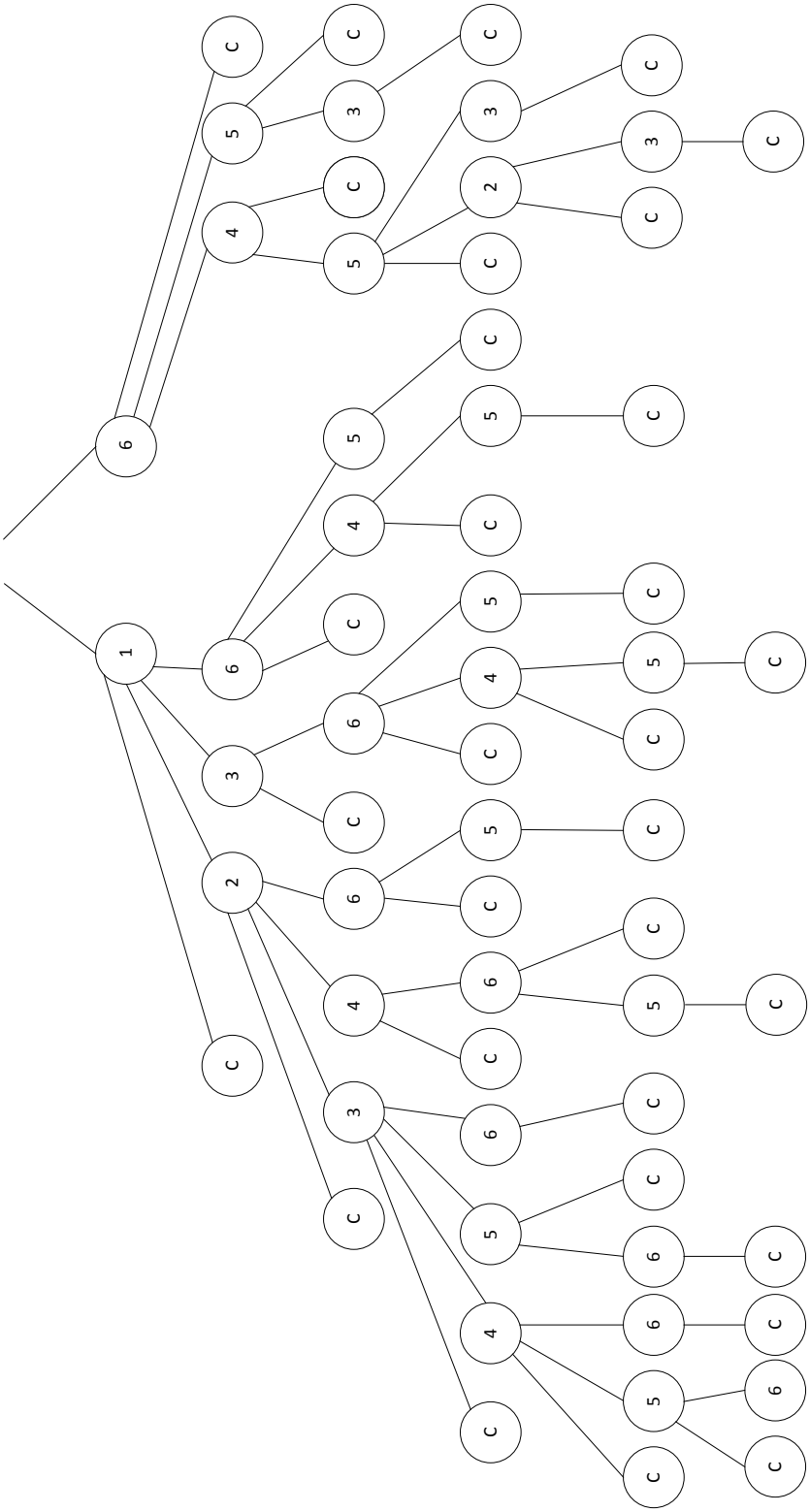

Figure 3.3 The branch and bound tree

3.1.2 REDUCTION PROPERTIES

We use the following property to reduce the size of the subproblem. The following property defines the tasks that should be put to one workstation alone.

Property 1: If $t_j + Min_{i \in S}\{t_i\} > CT$ then put task j to one workstation alone.
The proof follows from the fact that assigning task j even with the shortest task violates the cycle time constraint.

We check Property 1 at each close node. If task j that satisfies the conditions of the property exists then we update $K=K-1$ and let Total Cost=Total Cost + $\sum_{r \in l_j} C_r$. Such a task is considered while checking the precedence relations and it is placed after the first close node at which either all its predecessors or successors are assigned.

At each close node, we check for the feasibility imposed by the number of workstations. We let $\left\lceil \frac{\sum_i t_i}{CT} \right\rceil$ be a lower bound on the number of workstations.

For each node that represents closing workstation k with set S of unassigned task, the following expression gives a lower bound on the number of workstations.

$$LB_k = \left\lceil \frac{\sum_{i \epsilon S} t_i}{CT} \right\rceil$$

Property 1 below states the feasibility check formally.

Property 2: If $k+LB_k > K$ then the close option can never lead to a feasible solution.

Proof: There are K workstations, k of which are already opened. The remaining tasks can be scheduled by at least LB_k workstations. Hence if $LB_k > K-k$, the current partial solution cannot produce a feasible solution.

□

En route to finding all optimal solutions, we ignore some feasible solutions that cannot lead to an optimal solution. The following property helps us to define such nonpromising solutions.

Property 3: If there is a fittable task with no extra equipment requirement or requiring extra equipments that are not required by any unscheduled task then a close option cannot lead to a unique optimal solution.

Proof: Assume workstation k is closed although there is a fittable task, say task j that satisfies the conditions of the property and task j is assigned to workstation $k+r$. Task j is removed from its current workstation and put into workstation k such a change never increases the number of workstations and total equipment cost since task j fits to workstation k with no extra equipment i.e, cost, or the extra equipments are not required by any task in

workstation $k+r$. Hence a solution in which task j is assigned to workstation k is never worse.

□

3.1.3 LOWER BOUNDS

We now discuss our lower bounds used to eliminate the nonpromising partial solutions.

A lower bound on the number of workstations that require equipment r can be stated as:

$$\left\lceil \frac{\sum_{r \in l_i} t_i}{CT} \right\rceil = LB_r$$

This follows a valid lower bound on the total equipment cost Z, which is $\sum_r C_r \times LB_r = LB$ We state this result formally through Theorem 2.

Theorem 2: *LB* is a valid lower bound on the optimal total equipment cost.

Proof: Assume Z^* is the optimal equipment cost and Z_r^* is the optimal number of equipment type r, i.e.,

$$Z^* = \sum_r C_r \times Z_r^*$$

$Z_r^* \geq LB_r$ follows that $\sum_r C_r \times Z_r^* \geq \sum_r C_r \times LB_r$. Hence, $Z^* \geq \sum_r C_r * LB_r = LB$ and LB is a valid lower bound on Z^*.

□

We extend the lower bound to each partial schedule S.

S_k = set of assigned tasks to the current workstation k
W_k = load of the current workstation k
S = set of unassigned tasks

If equipment r is not already assigned to the current workstation;

$$LB_r = \left\lceil \frac{\sum_{\substack{i \in S \\ r \in l_i}} t_i}{CT} \right\rceil$$

If equipment r is already assigned to the current workstation;

$$LB_r = \left\lceil \frac{\sum_{\substack{i \in S \\ r \in l_i \\ i \notin S_k}} t_i - (CT - W_k)}{CT} \right\rceil$$

While calculating LB_r, we assume that the slack time of the current workstation is used by the tasks that use equipment r.

$$LB(S) = \sum_r C_r * LB_r$$

The expression LB(S) gives a lower bound on the total equipment cost of unassigned tasks.

We eliminate the partial schedule whenever TC(S) + LB(S) ≥ UB where,

TC(S) = total equipment cost incurred in workstations 1 through k
UB = total equipment cost of the best known feasible solution, i.e, incumbent solution

Our initial experimentation has shown that the initial upper bound does not have significant effect on the performance. Hence, initially we set UB to a very big value, and update UB whenever a complete solution with a smaller total equipment cost is reached.

We employ a depth-first strategy due to its relatively low memory requirements. According to this strategy we start from root node and explore a branching from the node having the smallest TC(S)+LB(S) value. We continue to branching until we reach to the last level with a feasible complete solution or to any level having no feasible or promising partial solutions. If further branching is not possible, we backtrack to the previous level. We terminate whenever we reach to level 0. The incumbent solution at termination is the optimal solution.

3.2. EXAMPLE PROBLEM

We illustrate our branch and bound procedure by the following 3 equipments example. The precedence diagram is given below where t_i and l_i represents the processing times and the required equipment set of task i, respectively. Initially, upper bound is set to infinity.

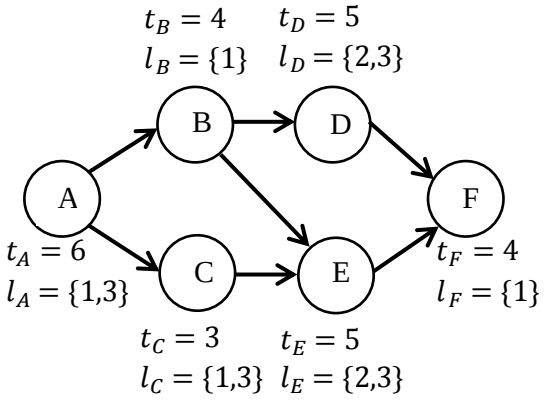

Figure 3.4 The precedence graph of the example problem

We set the cost of equipment type 1, 2 and 3 as 100, 200 and 300, respectively. Suppose cycle time, *CT* is 10 and the number of workstations, *K* is 3.

Initially, tasks A and F are eligible for the assignment. We calculate the lower bounds of the eligible tasks as follows.

$$\text{LB(A)} = \left\lceil \frac{t_\text{A} + t_\text{B} + t_\text{C} + t_\text{F}}{10} \right\rceil x100 + \left\lceil \frac{t_\text{D} + t_\text{E}}{10} \right\rceil x200 + \left\lceil \frac{t_\text{A} + t_\text{C} + t_\text{D} + t_\text{E}}{10} \right\rceil x300 = 1000$$

$$\text{LB(F)} = \left\lceil \frac{t_\text{A} + t_\text{B} + t_\text{C} + t_\text{F}}{10} \right\rceil x100 + \left\lceil \frac{t_\text{D} + t_\text{E}}{10} \right\rceil x200 + \left\lceil \frac{t_\text{A} + t_\text{C} + t_\text{D} + t_\text{E}}{10} \right\rceil x300 = 1000$$

Since LB(A)=LB(F), we start with the task having lower index, i.e., task 1.

After the assignment of task A, we have a partial solution where $S_1 = \{\text{A}\}$, $W_1 = 6$, $S = \{\text{B, C, D, E, F}\}$ and total cost is 400.

Now, tasks B and C become eligible in addition to task F. There are two options; (1) assigning one of the eligible tasks, (2) closing the current workstation. Close option is fathomed since all eligible tasks can be assigned to the current workstation without any additional equipment (See Property 2). Lower bounds of the eligible tasks are calculated as follows.

$$\text{LB(B)} = \left\lceil \frac{t_C + t_F - (10 - 10)}{10} \right\rceil \times 100 + \left\lceil \frac{(t_D + t_E)}{10} \right\rceil \times 200 + \left\lceil \frac{t_C + t_D + t_E - (10 - 10)}{10} \right\rceil \times 300 = 900$$

$$\text{LB(C)} = \left\lceil \frac{t_B + t_F - (10-9)}{10} \right\rceil \times 100 + \left\lceil \frac{(t_D + t_E)}{10} \right\rceil \times 200 + \left\lceil \frac{t_D + t_E - (10-10)}{10} \right\rceil \times 300 = 600$$

$$\text{LB(F)} = \left\lceil \frac{t_B + t_C - (10-10)}{10} \right\rceil \times 100 + \left\lceil \frac{(t_D + t_E)}{10} \right\rceil \times 200 + \left\lceil \frac{t_C + t_D + t_E - (10-10)}{10} \right\rceil \times 300 = 900$$

The current cost is 400. If task B or task F is assigned to the first workstation, the total equipment cost cannot be lower than 400+900. Similarly, if task C is assigned to the first workstation total equipment cost cannot be lower than 400+600. Note that, the upper bound is infinity. So, no elimination is possible using lower bounds. Hence, we assign task C having the minimum lower bound to the first workstation.

After the assignment of tasks A and C, we have a slack time of value 1. There is no task having processing time of 1 so the current workstation is closed and a new one is opened. At this point, we made a feasibility check since the number of workstations is given as 3.
S = {B, D, E, F}, k=1

$$k + \left\lceil \frac{(t_B + t_D + t_E + t_F)}{CT} \right\rceil = 1 + \left\lceil \frac{(4+5+5+4)}{10} \right\rceil = 3 \leq K = 3$$

The feasibility condition holds, so we continue with the next workstation.

There are two eligible tasks, task B and F with the same lower bound value of 600. Task B which has the lower index is assigned to the second workstation. After the assignment of task B, tasks D and E become eligible. Task F is already eligible and considered since it has a higher index than the last assigned task. There is also close option which is to be considered. The feasibility check is carried out in order to evaluate the close option as follows.

$$k + \left\lceil \frac{(t_D + t_E + t_F)}{CT} \right\rceil = 2 + \left\lceil \frac{(5+5+4)}{10} \right\rceil = 4 \not\leq K = 3$$

Close option is fathomed since, it violates the feasibility condition. Moreover, there is an eligible task that can be assigned to the current workstation without any additional equipment requirement

Lower bounds of the eligible tasks are calculated as shown below.

$$\text{LB(D)} = \left\lceil \frac{t_F - (10-9)}{10} \right\rceil \times 100 + \left\lceil \frac{t_E - (10-9)}{10} \right\rceil \times 200 + \left\lceil \frac{t_E - (10-9)}{10} \right\rceil \times 300 = 600$$

$$\text{LB(E)} = \left\lceil \frac{t_F - (10-9)}{10} \right\rceil \times 100 + \left\lceil \frac{t_D - (10-9)}{10} \right\rceil \times 200 + \left\lceil \frac{t_D - (10-9)}{10} \right\rceil \times 300 = 600$$

$$\text{LB(F)} = \left\lceil \frac{(t_D + t_E)}{10} \right\rceil \times 200 + \left\lceil \frac{t_D + t_E}{10} \right\rceil \times 300 = 500$$

Task F has a lower bound value of 500 while tasks D and E have a lower bound value of 600. Moreover, task D and task E requires extra equipments in order to assign to the second workstation. Thus, task F is assigned to the current workstation.

The current workstation is closed as there is no unassigned task left having higher index than task F and no tasks become eligible after the assignment of task F. Feasibility check holds, so we continue with the next workstation.

Task D is assigned to the new workstation since it has the same lower bound as task E and has lower index.

Now, only task E is left unassigned. We do not consider close option since it violates the feasibility condition and task E does not require any additional equipment when assigned to the current workstation.

As a result, a feasible solution is obtained with a total cost of 1000 and the following task and equipment assignments are obtained.

Workstation #	Task	Equipment Cost
1	A, C	400
2	B, F	100
3	D, E	500

We update the upper bound as 1000 and backtrack to the node with the execution number 15. At this point, there is one more alternative, i.e., task E needed to be evaluated. No eligible task is left once task E is assigned to the third workstation, so the workstation is to be closed. Thus, we make a feasibility check as follows.

$$k + \left\lceil \frac{t_D}{CT} \right\rceil = 3 + \left\lceil \frac{5}{10} \right\rceil = 4 \not\leq K = 3$$

Since the feasibility condition is violated we fathom this branch.

We continue backtracking and reach the node with execution number 9. There are two more alternatives, task D and task E. Lower bounds of these alternatives are calculated as 600. If one of these tasks is to be assigned total equipment cost incurred would be 1000. We carry out our elimination procedure for the partial schedules as follows.

$$\mathrm{LB(D)} = \mathrm{TC(S)} + \mathrm{LB(S)} = 1000 + 600 = 1600 \geq UB = 1000$$
$$\mathrm{LB(E)} = \mathrm{TC(S)} + \mathrm{LB(S)} = 1000 + 600 = 1600 \geq UB = 1000$$

Since the elimination condition holds, these branches are fathomed.

Once we continue in a similar manner, we reach to the first level with no eligible assignment. We stop and record the upper bound as the optimal objective function value. The partial branch and bound tree corresponding to our example is depicted by Figure 3.5. The execution number is given on the tree at the top of each node. For example, we first search tasks A and F and give numbers 1 and 2.

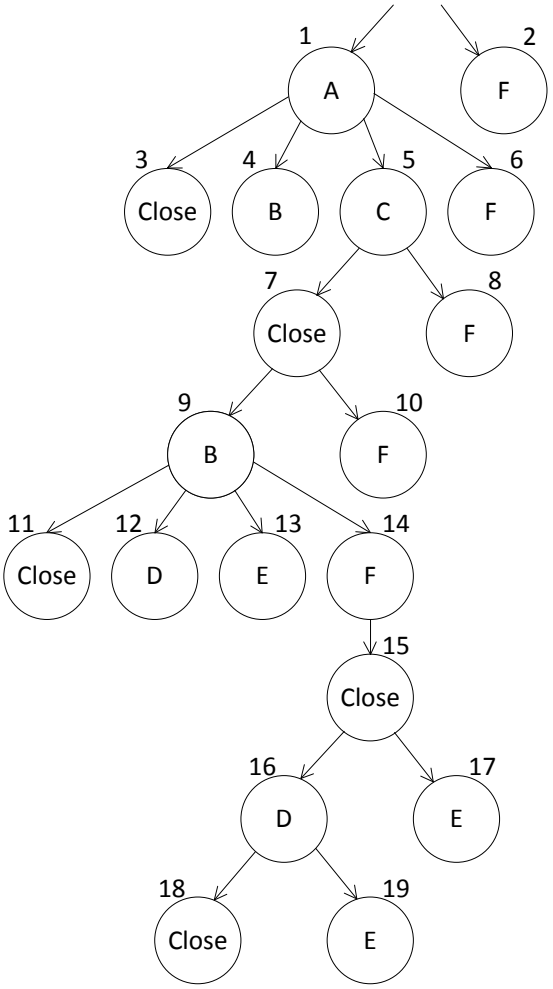

Figure 3.5 A partial branch and bound tree for the example problem

CHAPTER 4

COMPUTATIONAL EXPERIMENTS

In this chapter, we discuss the results of our experiment. The experiment is designed to test the performance of the branch and bound algorithm together with the reduction and bounding mechanisms.

We first give our data generation scheme and state our performance measures used to evaluate the branch and bound algorithm and integer model. Then, we report the results of our preliminary experiment. Based on the results of our preliminary runs, we design the main experiment and discuss its results.

4.1. DATA GENERATION

We select two precedence networks from Scholl that are available on the website http://www.assembly-line-balancing.de. These networks are used in some U-shaped studies like Urban (1998) and Aase et al. (2003). The selected networks reside 21 and 30 tasks and their respective precedence relations are depicted in Figures 4.1 and 4.2, respectively.

We generate task times from discrete uniform distribution U[1,10] and U[1,21] for the networks with 21 and 30 tasks, respectively.

We set the number of equipments, L, to 2, 3 and 5 and generate the following sets of equipment costs.

***L*=2**					
Set I	C_1=200	C_2=400			
Set II	C_1=300	C_2=400			
***L*=3**					
Set I	C_1=200	C_2=300	C_3=400		
Set II	C_1=300	C_2=350	C_3=400		
***L*=5**					
Set I	C_1=200	C_2=250	C_3=300	C_4=350	C_5=400

Note that, Set I resides smaller equipment costs with higher variability and Set II resides larger equipment costs with lower variability. We try a single set for L=5 as our experimentation on L=2 and L=3 have revealed that the costs do not have significant effect on the performance. As a total, 5 equipment cost sets are used.

We select three cycle time (*CT*) values for each *N* values. For each task *i*, we generate l_i from U[1,L]. For the small network (*N*=21), we select the cycle time values as 14, 15 and 21 and for the large network (*N*=30), we select the cycle time values as 45, 60 and 75.

For each cycle *CT* and *N* value, we use three values for the number of workstations, *K*. We set *K* to K_{min}, K_{min+1} and K_{min+2} where K_{min} is the minimum value for a feasible solution.

Two values of *N*, five sets of *L*, three values of *CT* and three values of *K* form 2x5x3x3=90 combinations. For each combination, we generate 10 instances. Hence, our experiments reside 900 problem instances.

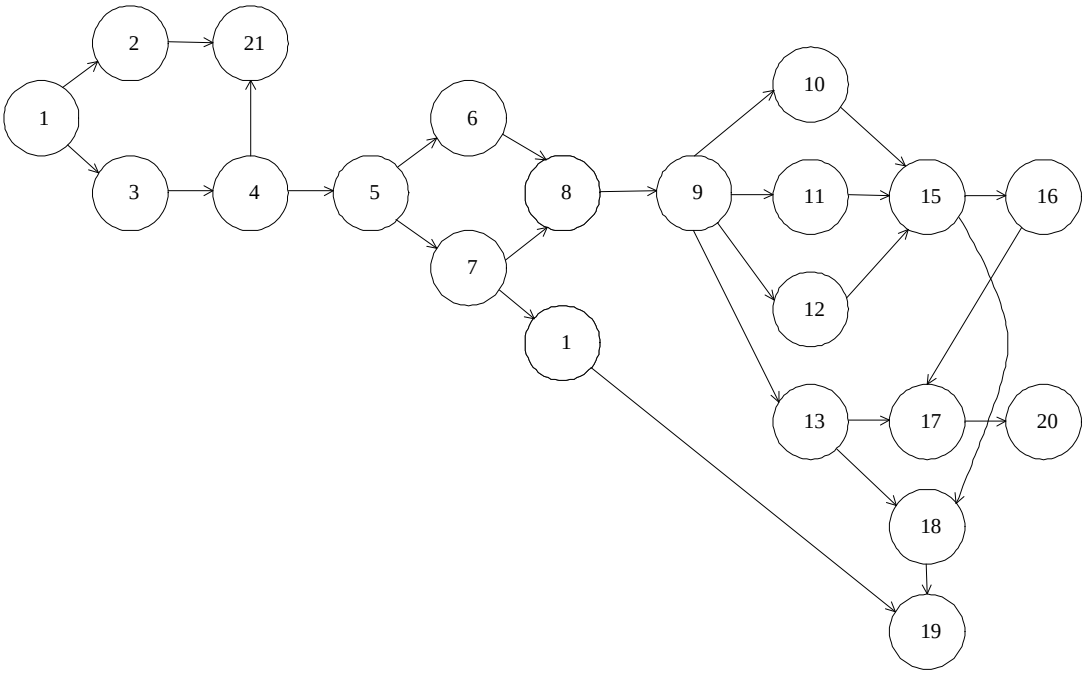

Figure 4.1 Precedence network with 21 tasks

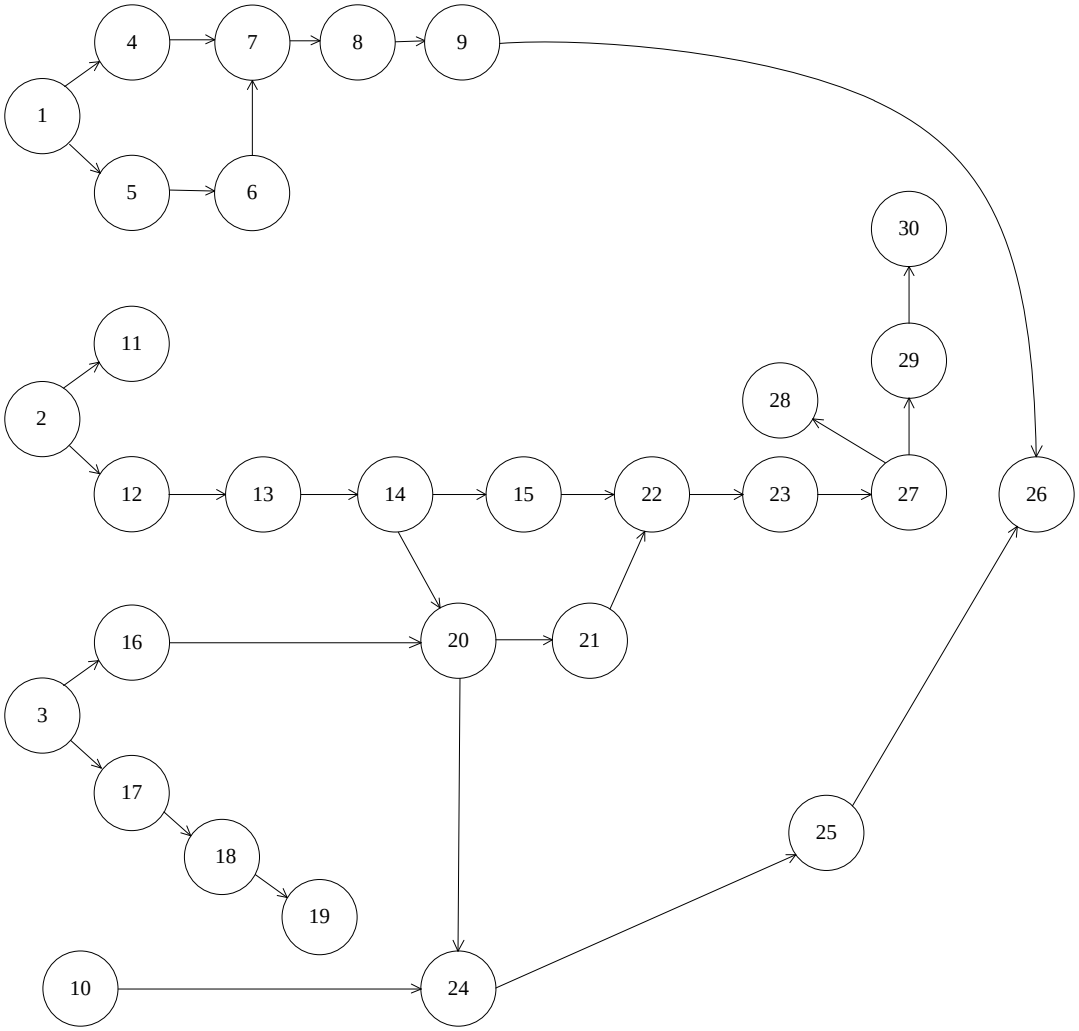

Figure 4.2 Precedence network with 30 tasks

We code our branch and bound algorithm in C++ programming language. The model is built using GAMS 23.5.1 with CPLEX 12 solver. The experiment is run on Intel Core i5, 1.80 GHz PC with 4 GB RAM.

4.2. PERFORMANCE MEASURES

We use the following performance measures to test the efficiency of our branch and bound algorithm.

- Average Central Processing Unit (CPU) Time (in seconds)
- Maximum CPU Time
- Average number of nodes
- Maximum number of nodes

We evaluate the mathematical model by the average and maximum CPU times. We set a termination limit of 2 hours for the execution of each instances. If the branch and bound algorithm or the model cannot return a solution in 2 hours, we terminate their execution. We also include the number of instances that could not be solved in 2 hours.

4.3. PRELIMINARY RUNS

The aim of the preliminary experiment is to test the power of the reduction and bounding mechanisms. We first investigate the performance of the precedence relations. We run our branch and bound algorithm with and without precedence relations and report the results in Table 4.1. The table gives the average and maximum solution times, and number of partial solutions for 21 tasks, 6 and 8 workstations and cycle time value of 21 for Set I.

Table 4.1 BAB Performances with and without Precedence Relations

L	*K*	**BAB with precedence relations**					**BAB without precedence relations**				
		CPU Time		**Number of nodes**		**# of unsolved instances**	**CPU Time**		**Number of nodes**		**# of unsolved instances**
		Average	**Maximum**	**Average**	**Maximum**		**Average**	**Maximum**	**Average**	**Maximum**	
2	6	7.3	30.0	72,728	309,043	0	9.8	39.7	96,587	403,403	0
	8	14.4	44.6	152,103	474,784	0	20.1	65.7	211,360	693,505	0
3	6	11.7	53.1	90,033	412,689	0	18.1	76.2	139,923	615,020	0
	8	22.0	57.6	178,435	458,668	0	38.4	113.0	303,914	826,642	0

As can be observed from the table, incorporating the precedence relations improves the performance of the branch and bound algorithm significantly. For example, when K=8 and L=2 the average CPU time is 14 seconds when the precedence relations are used and 20 seconds when the precedence relations are not used.

The effects of the precedence relations become more significant as L increases. Note from the table that when L=3 and K=8, the average CPU time is reduced about factor 2 (from 38.4 to 22.0) by the power of the precedence relations. For this combination, the precedence relations reduce the number of nodes from about 300,000 to 175,000.

We also observe that the precedence relations eliminate more as L increases. This is due to the fact our lower bounds become ineffective as L increases. Our lower bound is the collection of the lower bounds in terms of each equipment type. As the type of equipment increases number of estimations also increases leading to an ineffective lower bound.

In order to see the effect of our bounding mechanism we run our branch and bound algorithm with and without lower bound. We use the network with 30 tasks and set the number of workstations to 6 and 8 for Set II. L is taken as 2 and 3. The average and maximum values of CPU times and number of nodes searched are presented in Table 4.2.

Table 4.2 BAB Performances with and without Lower Bound

L	*K*	**BAB with Lower Bound**					**BAB without Lower Bound**				
		CPU Time		**Number of nodes**		**# of unsolved instances**	**CPU Time**		**Number of nodes**		**# of unsolved instances**
		Average	**Maximum**	**Average**	**Maximum**		**Average**	**Maximum**	**Average**	**Maximum**	
2	6	113.3	821.3	840,340	6,092,860	0	35.2	91.6	196,963	503,398	0
	8	544.7	3600.0	3,977,542	25,988,549	1	831.6	3600.0	4,724,101	21,867,576	2
3	6	965.8	2742.5	5,434,213	15,285,201	0	1971.4	3600.0	8,259,139	15,544,320	2
	8	1822.1	3600.0	10,637,842	21,166,897	2	2861.1	3600.0	11,622,245	15,664,850	6

Note form the table that bounding mechanisms decreases the CPU time and number of nodes drastically except for the combination where the smallest L and K combination is used. This is an expected situation since calculation of lower bounds may require more than the solution time for small problems solutions.

The effects of the bounding mechanisms become more effective as K increases. For instance, when L=3, K=6 and the bounding mechanisms are not used, there only 2 instances that cannot be solved with in the termination limit. However, when K=8, the number of unsolved instances increases to 6. The problem becomes harder as K increases and the elimination of the nonpromising alternatives at the earlier step gain importance and lower bound is an effective mechanism for the determination of such situations.

4.4. MAIN EXPERIMENT

The results of the preliminary runs reveal that, the precedence relations and lower bounds are very powerful. The effort spent to compute them is worth since they lead to significant reduction in the number nodes searched and the solution times. Based on these facts, we carry our main runs with the precedence relations and lower bounds.

In our main runs, we look for the effect of the parameters (number of tasks, number of equipments and number of workstations) on the performance of the branch and bound algorithm. Moreover, we compare the performance of the algorithm with the model solutions using GAMS, CPLEX solver.

We set a termination limit of 2 hours. We include the instances that do not return optimal solutions in 2 hours, while computing the average number of nodes and solution times.

We report the performance of the branch and bound algorithm in Tables 4.3 through 4.11. Tables 4.3 through 4.7 report on the small N, i.e., N=21 whereas Tables 4.8 through 4.11 repot on the instances with big N.

Table 4.3 BAB and Model Performances for Set I, L=2, N=21

CT	*K*	BAB					Model		
		CPU Time		Number of nodes		# of unsolved instances	CPU Time		# of unsolved instances
		Average	Maximum	Average	Maximum		Average	Maximum	
14	**8**	19.6	51.6	148,321	395,686	0	495.7	1298.7	0
	9	113.2	443.8	957,564	4,110,762	0	1415.3	2836.8	0
	10	1254.2	5327.6	11,960,447	50,472,787	0	4150.3	7200.0	4
15	**8**	15.0	67.2	147,048	702,898	0	313.9	565.7	0
	9	18.7	68.3	192,040	733,559	0	2843.2	7200.0	1
	10	23.8	83.8	250,652	873,140	0	4335.6	7200.0	5
21	**6**	7.3	30.0	72,728	309,043	0	94.1	344.2	0
	7	14.1	43.9	147,302	463,230	0	317.0	590.7	0
	8	14.5	44.6	152,103	474,784	0	1050.5	1732.7	0

Table 4.4 BAB and Model Performances for Set II, L=2, N=21

CT	*K*	BAB					Model		
		CPU Time		Number of nodes		# of unsolved instances	CPU Time		# of unsolved instances
		Average	Maximum	Average	Maximum		Average	Maximum	
14	**8**	19.5	51.7	148,426	395,686	0	362.5	1020.1	0
	9	113.4	434.0	969,726	4,110,762	0	1936.9	7200.0	1
	10	1852.3	7200.0	17,698,624	67,116,686	1	4289.1	7200.0	4
15	**8**	15.1	67.1	147,048	702,898	0	456.3	781.9	0
	9	19.1	69.6	192,309	733,559	0	2034.2	7200.0	1
	10	24.2	84.8	250,944	873,140	0	4679.7	7200.0	4
21	**6**	7.6	30.3	74,126	309,054	0	131.7	641.9	0
	7	14.4	44.1	148,926	463,278	0	398.0	903.5	0
	8	14.7	44.9	153,728	474,835	0	1776.2	5062.9	0

Table 4.5 BAB and Model Performances for Set I, L=3, N=21

CT	*K*	BAB					Model		
		CPU Time		Number of nodes		# of unsolved instances	CPU Time		# of unsolved instances
		Average	Maximum	Average	Maximum		Average	Maximum	
14	**8**	19.7	47.6	140,945	320,763	0	866.4	1766.2	0
	9	135.5	349.2	979,042	2,566,607	0	3830.6	7200.0	3
	10	1548.5	7200.0	12,444,791	59,826,746	1	4770.0	7200.0	6
15	**8**	25.0	63.2	199,490	526,380	0	1429.0	3717.0	0
	9	108.0	697.0	891,471	5,760,871	0	3523.8	7200.0	3
	10	189.7	872.5	1,592,683	7,259,671	0	5788.7	7200.0	7
21	**6**	11.7	53.1	90,033	412,689	0	323.4	770.8	0
	7	20.0	55.4	160,633	456,946	0	1993.2	4740.3	0
	8	22.3	57.6	180,147	458,668	0	5443.3	7200.0	5

Table 4.6 BAB and Model Performances for Set II, L=3, N=21

CT	*K*	BAB					Model		
		CPU Time		Number of nodes		# of unsolved instances	CPU Time		# of unsolved instances
		Average	Maximum	Average	Maximum		Average	Maximum	
14	**8**	21.2	48.6	146,901	320,763	0	802.4	1860.7	0
	9	146.7	383.0	1,025,468	2,721,610	0	3172.5	7200.0	2
	10	1583.1	7200.0	12,545,279	59,544,377	1	4470.7	7200.0	4
15	**8**	25.6	64.5	200,417	526,380	0	1153.9	2557.0	0
	9	112.2	710.1	908,428	5,771,693	0	4132.6	7200.0	3
	10	199.9	879.6	1,656,551	7,279,621	0	5659.3	7200.0	7
21	**6**	12.9	51.4	100,926	412,690	0	451.3	1165.9	0
	7	19.7	55.3	160,987	456,947	0	2556.6	5874.4	0
	8	21.6	55.8	179,462	465,223	0	5470.3	7200.0	4

Table 4.7 BAB and Model Performances for Set I, *L*=5, *N*=21

CT	*K*	BAB					Model		
		CPU Time		Number of nodes		# of unsolved instances	CPU Time		# of unsolved instances
		Average	Maximum	Average	Maximum		Average	Maximum	
14	**8**	356.7	895.1	1,998,431	5,178,657	0	1560.4	3218.3	0
	9	980.7	2785.3	5,564,119	15,381,393	0	2712.0	7200.0	1
	10	3110.9	7200.0	17,503,778	37,156,171	1	5218.6	7200.0	5
15	**8**	1145.6	7200.0	5,768,519	35,732,519	1	2591.2	7200.0	1
	9	968.3	4419.3	5,334,528	24,240,089	0	4866.5	7200.0	4
	10	2404.6	7200.0	13,957,032	40,881,837	1	6654.4	7200.0	9
21	**6**	30.9	108.5	173,234	629,043	0	600.5	933.7	0
	7	145.2	456.7	841,658	2,658,035	0	3970.4	7200.0	1
	8	229.8	553.6	1,367,819	3,359,613	0	6461.8	7200.0	8

Table 4.8 BAB and Model Performances for Set I, L=2, N=30

CT	*K*	BAB					Model		
		CPU Time		Number of nodes		# of unsolved instances	CPU Time		# of unsolved instances
		Average	Maximum	Average	Maximum		Average	Maximum	
45	**7**	2907.1	7200.0	21,595,138	53,295,505	1	2589.3	5052.5	0
	8	3991.7	7200.0	29,577,082	54,365,522	3	5950.1	7200.0	5
	9	4241.1	7200.0	31,658,330	56,121,938	4	7171.7	7200.0	9
60	**6**	137.2	818.5	1,028,169	6,092,860	0	280.4	801.6	0
	7	902.2	7200.0	6,968,110	56,074,976	1	1071.2	2932.1	0
	8	903.4	7200.0	6,902,235	55,226,296	1	3266.5	7200.0	2
75	**5**	399.7	2317.6	2,957,320	17,045,067	0	115.3	371.0	0
	6	425.2	2409.5	3,177,618	17,936,891	0	169.7	424.4	0
	7	425.2	2411.4	3,178,719	17,946,358	0	441.8	1145.9	0

Table 4.9 BAB and Model Performances for Set II, L=2, N=30

CT	*K*	BAB					Model		
		CPU Time		Number of nodes		# of unsolved instances	CPU Time		# of unsolved instances
		Average	Maximum	Average	Maximum		Average	Maximum	
45	**7**	2945.8	7200.0	21,362,305	52,131,340	1	1675.1	2257.2	0
	8	4006.2	7200.0	29,504,271	54,317,292	3	5589.4	7200.0	4
	9	4235.5	7200.0	31,442,326	55,035,328	4	7200.0	7200.0	10
60	**6**	138.3	821.3	1,028,638	6,092,860	0	440.1	2038.0	0
	7	903.0	7200.0	6,952,161	55,903,493	1	1100.3	2265.3	0
	8	904.9	7200.0	6,986,385	56,055,774	1	3454.8	7200.0	1
75	**5**	394.7	2284.7	2,957,320	17,045,067	0	106.3	567.4	0
	6	420.7	2375.6	3,177,618	17,936,891	0	278.9	1278.5	0
	7	422.6	2387.8	3,178,719	17,946,358	0	656.6	2551.1	0

Table 4.10 BAB and Model Performances for Set I, L=3, N=30

CT	*K*	BAB					Model		
		CPU Time		Number of nodes		# of unsolved instances	CPU Time		# of unsolved instances
		Average	Maximum	Average	Maximum		Average	Maximum	
45	**7**	3609.9	7200.0	20,018,097	39,677,436	1	6434.8	7200.0	3
	8	3960.7	7200.0	22,236,329	42,146,727	2	7200.0	7200.0	10
	9	4050.8	7200.0	23,362,392	42,505,686	2	7200.0	7200.0	10
60	**6**	2033.3	7200.0	11,354,383	40,351,081	1	1311.8	4385.3	0
	7	3617.2	7200.0	20,377,296	42,371,958	3	4226.3	7200.0	4
	8	3835.6	7200.0	21,930,592	42,614,738	4	6655.0	7200.0	7
75	**5**	1332.0	3625.8	7,555,611	20,603,608	0	113.5	174.2	0
	6	3043.7	7200.0	17,623,516	41,851,255	2	628.3	1212.5	0
	7	3124.4	7200.0	18,244,150	42,342,744	2	2234.3	7200.0	1

Table 4.11 BAB and Model Performances for Set II, L=3, N=30

CT	*K*	BAB					Model		
		CPU Time		Number of nodes		# of unsolved instances	CPU Time		# of unsolved instances
		Average	Maximum	Average	Maximum		Average	Maximum	
45	**7**	3846.3	7200.0	21,456,888	40,610,375	1	6035.0	7200.0	2
	8	4051.8	7200.0	22,605,721	40,925,976	2	7200.0	7200.0	10
	9	4440.6	7200.0	25,639,153	42,753,978	4	7200.0	7200.0	10
60	**6**	2023.4	7200.0	11,340,572	40,191,025	1	1738.1	7200.0	1
	7	3598.0	7200.0	20,370,871	41,335,961	3	4687.5	7200.0	4
	8	3823.5	7200.0	21,823,238	41,756,306	4	7091.9	7200.0	9
75	**5**	1428.1	3699.4	8,067,846	21,049,568	0	180.4	413.3	0
	6	3155.4	7200.0	18,208,051	41,654,819	2	544.0	1383.7	0
	7	3241.5	7200.0	19,097,855	42,738,748	2	1695.2	2845.9	0

Tables 4.3 and 4.4 reside the low equipment cost and high equipment cost instances, respectively, for two equipment types. Tables 4.5 and 4.6 report the respective results for three equipment types. In all tables, we report on the performances based on three different cycle time values and three different values for the number of the workstations.

Using all tables, we discuss the effect of the problem size parameters (N, L and K) and the parameters related task and equipment, on the performance of the branch and bound algorithm. We also include a discussion that compares the branch and bound and model performances.

We observe that the problem size parameters N and K are very effective on the complexity of the solutions. The effect of N is more dominant than that of K. For N=21, we could solve instances with up to 10 workstations when CT=14 and 15, and 8 workstations when CT=21. On the other hand when N=30, we could not solve any instance when K=10. The maximum numbers of workstations that we find solutions are 10 and 9 for small and big N, respectively.

When N increases, the number of decisions hence the depth of the branch and bound tree increases significantly. This increase directly affects the number of nodes and CPU times. Moreover, as the number of tasks increases, the precedence relations and lower bounds become weaker.

The tables indicate that when N=21, K=8 and L=2 all 60 instances over two sets could be solved optimally. When N becomes 30, 8 out of 40 instances with CT=45 and 60 values could not be solved.

For fixed N, when K increases the number of nodes and CPU time increase significantly. This is due to the fact that the feasible region enlarges, thereby leaving many partial solutions, i.e, nodes, for evaluation. Moreover, the evaluations by reduction and bounding mechanisms are more effective for small values of K. For N=21, note from Tables 4.3 through 4.7 that when CT=15, as K increases from 8 to 10, the CPU times increase from 15.0 to 23.8, 15.1 to 24.2, 25.0 to 189.7, 25.6 to 199.9 and 1145.6 to 2404.6 for L=2 and Set I, L=2 and Set II, L=3 and Set I, L=3 and Set II, and L=5 and Set I, respectively.

The effect of K is more pronounced for N=30 Tables 4.8 through 4.11 reveal that as K increases from 6 to 8 for CT=60, the CPU times increase from 137.2 to 903.4, 138.3 to 904.9, 2033.3 to 3835.6 and 2023.4 to 3823.5 for L=2 and Set I, L=2 and Set II, L=3 and Set I, and L=3 and Set II, respectively.

We also observe that the effect of K becomes more prominent when CT decreases. This is due to the fact that for small values of cycle time, the number of workstations becomes very tight, and many partial solutions are eliminated due to this tight constraint. Note from the tables that when CT is low, i.e, 14 for N=21, when K increases from 8 to 10, the CPU times increase from 19.6 to 1254.2, 19.5 to 1852.3, 19.7 to 1548.5 and 21.2 to 1583.1, for L=2 and

Set I, L=2 and Set II, L=3 and Set I, and L=3 and Set II, respectively. For N=30, we observe that for CT=45, when K increases from 7 to 9, the CPU times increase from 2907.1 to 4241.1 for L=2 and Set I, from 2945.8 to 4235.5 for L=2 and Set II, from 3609.9 to 4050.8 for L=3 and Set I, and from 3846.3 to 4440.6 for L=3 and Set II.

Those tables also point that when the number of equipments increases, the problem becomes harder to solve. This is due to the fact that reduction and bounding mechanisms perform poorer for higher values of L. For N=21, note from tables that for CT=15 and K=9, when the number of equipments increases from 2 to 3 and 3 to 5, the CPU times increases from 18.7 to 108.0 and from 108.0 to 968.3 for Set I, respectively. For N=30, when CT=60 and K=6, as L increases from 2 to 3, the CPU times increases from 137.2 to 2033.3 and 138.3 to 2023.4 for Set I and Set II respectively.

On the other hand, we could not observe any significant effect of the cost figures on the performance. The CPU times are very close for Set I (low variability case) and Set II (high variability case).

We also observe a negative correlation between CT and the CPU times. As CT increases the objective function values become relatively small. This follows the associated lower bounds are closer to the optimal objective function values, hence they are more powerful. Note from the tables that, for N=21, K=10 as CT increases from 14 to 15, the CPU times decreases from 1254.2 to 23.8, 1852.3 to 24.2, 1548.5 to 189.7 and 1583.1 to 199.9 for L=2 and Set I, L=2 and Set II, L=3 and Set I, and L=3 and Set II, respectively. For N=30, K=8 as CT increases from 45 to 60, the CPU times decreases from 3991.7 to 903.4, 4006.2 to 904.9, 3960.7 to 3835.6 and 4051.8 to 3823.5 for L=2 and Set I, L=2 and Set II, L=3 and Set I, and L=3 and Set II, respectively

We observe that all problem combinations, our branch and bound algorithm shows superior performance than the integer model solutions by GAMS. The only exception is for N=30 and CT=75. Note that for N=21, the number of unsolved instances in two hours (out of 360 instances) is 3 for BAB whereas the model cannot return optimal solution for 64 instances.

In almost all problem combinations, the BAB runs significantly faster. For example, when N=21, L=2, CT=15, K=10 the average CPU time used by the BAB is 24 seconds whereas the model cannot solve 9 instances in two hours. For another notable example for small N when L=3, CT=15, K=10. BAB use about 200 seconds for one instance, whereas GAMS cannot solve 7 out of 10 instances in two hours for both sets. For N=30, when CT=45, K=9, the CPU times of BAB are 4241.1, 4235.5, 4050.8 and 4440.6 while the CPU times of the model are 7171.7, 7200, 7200 and 7200 for L=2 and Set I, L=2 and Set II, L=3 and Set I, and L=3 and Set II, respectively. Notice that, the model could solve only 1 instance optimally (out of 40) in two hours.

The performance of GAMS deteriorates significantly as N increases. This is due to the increase in the number of binary decision variables. We report for the smallest values of the

trial cycle times (*CT*=14, *N*=21 and *CT*=45, *N*=30). Note that when *N*=21 and *L*=2, 4 and 5 out of 30 instances for Set I and Set II remain unsolved in two hours, respectively. When *N* becomes 30, about half of the instances (14 out of 30 for each set) could not be solved. For the same combinations but with *L*=3, when *N*=21, 6 instances could not be solved for Set I and Set II in two hours. When *N* becomes 30, the number of unsolved instances increases to 22 and 23 for Set I and Set II, respectively.

The number of the binary decision variables is also affected by the number of workstations and number of equipments. Hence, the performance of the model deteriorates as *K* and *L* increases. Note from the Tables 4.3 and 4.4 that when *N*=21, *K*=8 and *L*=2, all instances could be solved in two hours. When *K* becomes 10 for the same *N* and *L* values, only 23 out of 40 instances could be solved in two hours. Tables 4.8 and 4.9 reveal that when *N* becomes 30, for *K*=7 and *L*=2 all 20 instances over two sets could be solved. For the same combination but with *K*=9, only 1 out of 20 instances could be solved in two hours. This verifies the significant effect of *K* values on the complexity of the model.

We also observe from the tables that the *L* values have significant effect on the model complexity. The CPU times are much higher for *L*=5 compared to 3 and *L*=3 compared to *L*=2. Note that, when *N*=21 and *L*=2, 20 out of 180 instances could not be solved in two hours. When *L* becomes 3, 44 out of 180 instances remain unsolved in two hours. When *L*=5, 29 out of 90 instances could not be solved in two hours. This effect is even more significant for big *N*, *N*=30. Note from Tables 4.8 through 4.11 that when *L*=2, 31 out of 180 instances could not be solved in two hours and when *L*=3, 71 out 180 instances remain unsolved in two hours. Those results verify the significant effect of *L* values on the performance of the integer model.

Another remarkable observation is related to *CT*. As *CT* increases many tasks fit to the same workstation, hence assignment decisions are taken easier. For *N*=30, *K*=7 as *CT* increases from 45 to 60, the CPU times decreases from 2589.3 to 1071.2, 1675.1 to 1100.3, 6434.8 to 4226.3 and 6035.0 to 4687.5 for *L*=2 and Set I, *L*=2 and Set II, *L*=3 and Set I, and *L*=3 and Set II, respectively.

CHAPTER 5

CONCLUSIONS

In this study, we consider a U-Shaped Assembly Line Design Problem with fixed number of workstations. We assume that the tasks require specified equipments and the cycle time is known. Each equipment has a specific cost. Our problem is to assign the tasks to the workstations so as to minimize the total cost of equipment.

We formulate the problem as a mixed integer linear program and observe that the model fails to solve large-sized problem instances.

We develop some properties that explain the structure of the optimal solution. Based on the properties, we develop an efficient way of defining partial solutions of our branch and bound algorithm. We improve the performance of the branch and bound algorithm via some efficient lower bounding procedures.

The results of our extensive experiment have revealed that the number of tasks and the number of workstations are very effective on the performance of the branch and bound algorithm. However, we find that the number of equipments is not as effective. We find that the problems with up to 21 jobs and 10 workstations, and 30 jobs and 8 workstations are solved in reasonable times. We also observe that cycle time is an important parameter that affects the performance, and find that the solution times increase as cycle time decreases.

To the best of our knowledge, our study is the first attempt to solve the flexible U-shaped assembly design problem.

Future research may also point out the development of heuristic procedures to solve large-sized problem instances. The heuristic procedures may benefit from our branch and bound algorithm, like truncated branch and bound algorithms, α-approximation algorithms and beam search techniques.

We can extend our findings for different problem environments where there are equipment alternatives and the task times depend on the equipment selected. Another extension might be to consider the number of workstations as a decision.

REFERENCES

Aase, G.R., Schniederjans, M.J., Olson, J.R., 2003. 'U-Opt: An Analysis of Exact U-shaped Line Balancing Procedures', International Journal of Production Research, 41, 4185-4210.

Barutçuoğlu, Ş., Azizoğlu, M., 2011. 'Flexible Assembly Line Design Problem with Fixed Number of Workstations', International Journal of Production Research, 49, 3691-3714.

Baybars, İ., 1986. 'A Survey of Exact Algorithms for the Simple Assembly Line Balancing Problem', Management Science, 32, 909-932.

Becker, C., Scholl, A., 2006. 'A Survey on Problems and Methods in Generalized Assembly Line Balancing', European Journal of Operational Research, 168, 694-715.

Buckhin, J., Tzur, M., 2000. 'Design of Flexible Assembly Line to Minimize Equipment Cost', IEE Transactions, 32, 585-598.

Ege, Y., Azizoğlu, M., Özdemirel, N.E., 2009. 'Assembly Line Balancing with Station Paralelling', Computers & Industrial Engineering, 57, 1218-1225.

Erel, E., Sabuncuoğlu I., Aksu, B.A., 2001. 'Balancing of U-type Assembly Systems Using Simulated Annealing', International Journal of Production Research, 39, 3003-3015.

Gökçen, H., Ağpak, K., 2006. 'A Goal Programming Approach to Simple U-line Balancing Problem', European Journal of Operational Research, 171, 577-585.

Kara, Y., Özgüven, C., Yalçin, N., Atasagun, Y., 2011. 'Balancing Straight and U-shaped Assembly Lines with Resource Dependent Task Times', International Journal of Production Research, 49, 6387-6405.

Scholl, A., 2007, 'Data of U-line Assembly Line Balancing', http://www.assembly-line-balancing.de/

Shewchuk, J.P, 2008. 'Worker Allocation in Lean U-shaped Production Lines', International Journal of Production Research, 46, 3485-3502.

Simaria, A.S., Zanella de Sa, M., Vilarinho, P.M., 2009. 'Meeting Demand Variation Using Flexible U-shaped Assembly Lines', International Journal of Production Research, 47, 3937-3955.

Sirovetnukul, R., 2009. 'Worker Allocation in U-shaped Assembly Lines with Multiple Objectives', Industrial Engineering and Engineering Management, vol. 105-109.

Urban, T.L., 1998. ‘Optimal Balancing of U-shaped Lines’, Management Science, 44, 738-741.