TYPE-II TRANSFER LINE BALANCING PROBLEM

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

BY

YASIN ERSIN TELEMECI

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

SEPTEMBER 2019

Approval of the thesis:

TYPE-II TRANSFER LINE BALANCING PROBLEM

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

Date: 09.09.2019

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, Surname: Yasin Ersin Telemeci

Signature:

ABSTRACT

TYPE-II TRANSFER LINE BALANCING PROBLEM

Telemeci, Yasin Ersin
Master of Science, Industrial Engineering
Supervisor: Prof. Dr. Meral Azizoğlu

September 2019, 45 pages

In this thesis, we consider Type II Transfer Line Balancing that assigns the operation to the blocks and blocks to the stations so as to maximize the production rate, i.e., to minimize the cycle time.

A mixed integer programming model is developed and a branch and bound algorithm is proposed for the exact solutions. The efficiency of the branch and bound algorithm is enhanced by lower and upper bounding procedures. The computational results have revealed the satisfactory behavior of the model and branch and bound algorithm on moderate size problem instances.

Keywords: Transfer Lines, Type-II Transfer Line Balancing, Mathematical Model, Branch and Bound Algorithm

ÖZ

TİP-2 TRANSFER HATTI DENGELEME PROBLEMİ

Telemeci, Yasin Ersin
Yüksek Lisans, Endüstri Mühendisliği
Tez Danışmanı: Prof. Dr. Meral Azizoğlu

Eylül 2019, 45 sayfa

Bu tezde, operasyonları bloklara, blokları istasyonlara atayarak üretim hızını ençoklamayı veya çevrim süresini enazlamayı hedefleyen Tip-II Transfer Hattı Dengeleme Problemini ele aldık.

Kesin çözümlere ulaşmak amacıyla karışık tamsayı programlama modeli ve dal-sınır algoritması geliştirdik. Dal-sınır algoritmasının verimliliği alt ve üst sınır belirleme yöntemleri kullanılarak arttırıldı. Deneysel sonuçlar, modelin ve dal-sınır algoritmasının orta ölçekli problem örneklerinde tatmin edici davranışlarını ortaya çıkarmıştır.

Anahtar Kelimeler: Transfer Hatları, Tip-II Transfer Hattı Dengeleme, Matematiksel Model, Dal-Sınır Algoritması

To my family for their endless
support and encouragement…

ACKNOWLEDGEMENTS

First, I wish to thank my supervisor Prof. Dr. Meral Azizoğlu for her valuable contributions and guidance throughout this study. I am also very thankful for the support and understanding.

I want to thank my company ASELSAN INC. and my managers for allowing me to spare time for this study. I am also grateful to my associates for handling my workload.

I would like to send my greetings to my colleagues Umutcan Ayasandır, Nail Karabay, Fatih Canbaz, Ercan Kablan and many others for their support and encouragement.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Production lines are characterized by a serially connected workstations that are linked by a material handling system. They are widely used in manufacturing, particularly in assembly, systems to produce high quantities of similar products. Transfer lines (tandem lines) are automated production lines that reside a series of multi-spindle machines. The multi-spindle machines do have several spindle heads. Each multi-spindle head has several spindles where each spindle can perform one operation at a time. The operations assigned to the spindles of any spindle head are integrated in such a way that their simultaneous processing is possible. This parallel processing is an important factor that increases the yield of the transfer lines.

Figure 1 is a pictorial representation of the transfer lines.

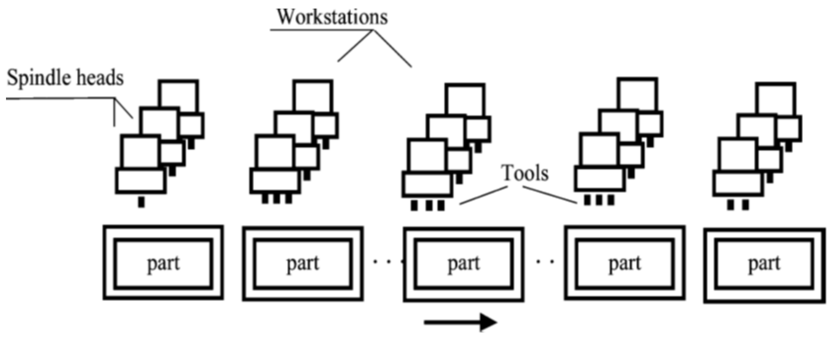

Figure 1 A transfer line configuration (excerpted from Dolgui et al. 2006).

On the above line, the operations are assigned to the blocks and processed in parallel. Once the operations of a block are performed, the part moves to the next block. Once all blocks of a station are performed, the part moves to the next station via automated handling mechanism.

Transfer lines are typically designed for very high levels of a single product type and they are inevitable to compete in today's markets that require high quality products, at high levels with low production times. These lines use high levels of automation and their efficient utilization is crucial to justify high investment costs of the multi spindle machines, spindle heads and material handling mechanism that link the machines and spindle heads.

The design problem for the transfer lines concerns with the determination of the number of machines (stations), number of spindle heads (blocks per station) and number of spindles (operations per block). The objective is usually to minimize the total investment cost.

Having an already installed transfer line, the operational problem is the allocation of the operations to the blocks and blocks to the stations so as to maximize the production rate, equivalently minimize the cycle time. The aim is to maximize total yield, therefore profit, so that the high line installation cost is justified. This operational problem is referred to as Transfer Line Balancing Problem.

Transfer Line Balancing Problem reduces to the Assembly Line Balancing Problem in the absence of its special constraints set and when there is a single operation per block and there is no limit on the number of blocks.

According to the Assembly Line Balancing terminology, the design problem is analogous to the Type I Assembly Line Balancing problem whereas the operational problem is analogous to the Type II Assembly Line Balancing problem. We refer to the operational problem in transfer lines as Type II Transfer Line Balancing Problem (Type II TLBP).

The research on the TLBP is of recent origin. The first study is due to in Dolgui et al. (2000) since then many studies have published on some extensions of Type I TLBP. To the best of our knowledge, there is no reported study on the Type II TLBP. Recognizing this gap in the literature, we study Type II TLBP and develop a mathematical model and propose a branch and bound algorithm along with powerful bounding mechanisms. We show the superiority of the algorithm over mixed integer linear programming model for small sized instances. On the other hand, for the medium sized instances, the mathematical model performs better. We also observe the satisfactory performance of the upper bounds, hence their promise for large sized problem instances.

We organize the rest of the study as follows. In Chapter 2, we review the Type II Assembly Line Balancing Problem and Type I Transfer Line Balancing Problem. In Chapter 3, we give the details of the problem environment and present the mathematical model and the branch and bound algorithm. In Chapter 4, we discuss the results of our experiment. Chapter 5 concludes the study by giving the main findings and addressing the future research directions.

CHAPTER 2

LITERATURE REVIEW

The classical flow line balancing problem concerns the assignment of a set of operations to workstations without exceeding the cycle time and obeying to the precedence relations. This problem is referred to as simple assembly line balancing problem (SALBP) in the literature. The SALBP was introduced in Salveson (1955) and has been the subject of many researches since then. For the extensive review of SALBP, one may refer to the studies by Battaia and Dolgui (2013).

The SALB studies are categorized into two types: Type I and Type II problem. Type I SALB problems aim to minimize the number of workstations for a given cycle time, whereas Type II SALB problems aim to minimize cycle time for a given number of workstations. Type I problems set the design of the assembly line by defining the number of resources used while Type II problems concern the efficient operation of the line for the already settled flow lines.

The transfer line balancing problem (TLBP) aims the assignment of operations to the blocks and the assignment of the blocks to the workstations while meeting more complex precedence relations. Type I TLBP concerns the cost of the configuration through the number of blocks and number of workstations for a specified cycle time. Type II TLBP concerns the minimization of the cycle time, thereby maximization of the production rate for a given number of workstations and the number of operations per block.

In this study we consider Type II TLBP and focus our literature review on two very closely studied problem environments: Type II SALBP and Type I TLBP. To the best of our knowledge, there is no reported study on Type II TLBP.

2.1. Literature on Type-II Simple Assembly Line Balancing Problem

In the literature, the SALB-II studies generally benefited from the corresponding solutions of the SALB-I problems.

Scholl (1994) developed a branch and bound method (B&B) for Type-II SALBP that use tabu search algorithm for finding initial upper bounds and several lower bounds. Their branch and bound employs a task-oriented depth-first search strategy.

Klein and Scholl (1996) propose a B&B procedure for Type-II SALBP. That repeatedly solves the related Type I SALBP by B&B method. Their B&B employs efficient bounding and dominance rules. They show that their algorithm outperformed the one proposed by Scholl (1994).

2.2. Literature on Transfer Line Balancing Problem

All studies in TLBP considers Type I problem, i.e., minimizing some function of the number of stations and blocks.

Dolgui et al. (2000) is the first study that introduced the TLBP. Cost minimization of opening blocks and stations is considered while satisfying the operational and technological constraints. Mixed integer program (MIP) and shortest path approach are suggested as solution methods. MIP model’s precedence relations constraint is formulated in four different ways. In addition, the initial problem is assimilated to constrained shortest path problem that gives exact solution.

Dolgui et al. (2005a) suggested two heuristic algorithms based on the COMSOAL technique that is introduced by Arcus (1996). The first algorithm is named as Recursive Assignment of Predecessors (RAP) and the second algorithm is named as First Satisfy Inclusion Constraints (FSIC). (See Guschinskaya et al., 2007) for detailed information about FSIC.) In both methods, operations are assigned to the blocks in stations randomly and best assignment is kept. According the RAP algorithm, a feasible operation is chosen randomly, then, its predecessors and inclusion constraint pairs are assigned. If this operation cannot be assigned to a block, a new block is formed and if cannot be assigned to a station, a new station is created. The algorithm controls the station exclusion constraints after the station is filled. Therefore, some undesirable solutions are generated. The FSIC algorithm focus to eliminate these infeasible solutions by processing an operation’s inclusion constraint pairs and unassigned predecessors. The results of their experiment showed that FSIC algorithm is more effective than the RAP algorithm.

Dolgui et al. (2005b) developed a MIP model and propose mechanisms to reduce its size. They also introduced decomposition heuristic procedure for large size problems. The mechanisms for the reduction of the model size is similar to the SALBP study of Patterson and Albracht (1975) but more complex due to the inclusion and exclusion constraints. In the decomposition heuristic procedure, the initial set of operations are partitioned into smaller sets and then these small size sets are separately solved by using MIP model. Guschinskaya et al. (2005c) provided a hybrid method which is based on the decompositions of the initial set of operations. FSIC is used for finding a feasible solution and an assignment path. Then, decomposition is based on the feasible solution’s assignment path. Each subproblem is solved by the shortest path approach. Lastly, feasible solution assignment path is updated according to subproblem’s solutions’ assignment path.

Dolgui et al. (2006) compared the solution methods for the TLBP. These approaches are MIP, shortest path approach, FSIC, decompositions methods, hybrid method and aggregate solving. Dolgui et al. (2009) extended the solution methods comparison by investigating deterministic decomposition based on precedence graphs and heuristic multi-start decomposition. Heuristic multi-start decomposition method is an improvement of the hybrid method by applying FSIC many times. (See Guschinskaya et al. (2008) for detailed information.) In both studies, the approaches are compared by the problem size (small, medium, large). For the small size instances, the shortest path method returns better solutions both in terms of computational time and quality of solutions. For medium and large size instances, hybrid and FSIC methods are better than their competitors.

Dolgui et al. (2009) developed a lower bound (LB) based set partitioning problem and applied branch and bound algorithm. B&B procedure is proposed whose efficiency is enhanced by LB and dominance rule. It is close to our B&B solution approach. In fact, Scholl and Klein (1996) provided the main concepts of B&B for SALBP. Determination of LB and Upper Bound (UB) is clearly introduced and possible improvements of these bounds remarked properly. B&B as an implicit enumeration method detailed by examining fathoming rules, backtracking and efficient selections. We mostly benefited from the paper in our work's B&B solution method part.

Dolgui (2010) discussed the new trends and challenges in the TLBP. For fixed and known demand Dedicated Transfer Lines (DTL) can be used. For diverse and unknown demand Flexible Transfer Lines (FTL) can be used. Transfer lines are evolved due to advanced technology and highly competitive markets. Mentioned transfer lines can become obsolete because of facing large variations and demand uncertainty. Reconfigurable Transfer Lines (RTL) have been proposed to adapt the changes in market variations. Some authors imply RTL is more flexible than the FTL. Reconfiguration and rebalancing of transfer lines foreseen as attractive topic about this subject.

Dolgui (2013) introduced a procedure on a locally feasible solution so as to reduce TLBP to the set partitioning problem. Many suggestions are made to reduce the number of variables and the size of the feasible domain. Their computational results reveal the superiority of the algorithm to the earlier similar models.

Battaia et al. (2017) considered a joint formulation of process planning and transfer line design for mixed model production. A mathematical model is proposed tested on real time industrial problem. Their algorithm could solve the instances with up to six different parts.

CHAPTER 3

PROBLEM DEFINITION & SOLUTION APPROACHES

In this chapter we first define the problem and then give the details of the solution approaches.

3.1. Problem Statement

We assume n operations should be assigned to one of prespecified stations. There are m stations that are equally equipped and can reside any operation, without exception. The time that is required to process operation i is ti time units. The operations are first assigned to the blocks and then the blocks are assigned to the workstations. The operations are not divisible so that each operation should be assigned to exactly one block.

Due to the restrictions on the number of spindle heads, we assume each block resides at most no operations and each station resides at most mo blocks.

There are several types of precedence relations that are grouped as follows:

Immediate Predecessors: Operation j cannot be started if operation i is not complete. Hence operation i should be assigned to a block that is processed earlier than that of operation j's block. We let IPi be the set of immediate predecessors of operation i.

Exclusion Constraints:

Station Exclusion Constraints: Two operations i and j cannot be processed in the same station. This might be due to the technological capability of stations. The resource that are required by those operations cannot be placed properly in one station. We let ES be the set of operation pairs with station exclusion constraints.

Block Exclusion Constraints: Two operations i and j cannot be placed to the same block. Those operations might be so different that they cannot share the same setup. We let EB be the set of operation pairs with block exclusion constraints.

We further assume that the parameters that define our problems are known with certainty and not subject to any change. That is the system is deterministic and static.

The parameters and decision variables are denoted as follows.

Parameters:

j operations 1,…,n
q blocks 1,…, n_o
k stations 1,…,m
m fixed number of stations
$\boldsymbol{n_o}$ max number of operations per block
t$_{\mathbf{j}}$ operation time of operation j
$\boldsymbol{IP_j}$ set of direct predecessors of operation j
ES a family of subset representing operations to “not” same station
EB a family of subset representing operations to “not” same block

Decision variables:

$\boldsymbol{F_{qk}} = Time\ of\ block\ q\ in\ station\ k$

$$\boldsymbol{X_{jqk}} = \begin{cases} 1 & \\ & if\ \ operation\ j\ is\ assigned\ to\ block\ q\ in\ station\ k \\ 0 & \end{cases} \quad \begin{matrix} j = 1, \dots, n \\ q = 1, \dots, mo \\ k = 1, \dots, m \end{matrix}$$

$\boldsymbol{CT} = Cycle\ time$

We aim to minimize cycle time, CT. CT defines our objective function as

$\boldsymbol{Min\ CT}$

The constraint set has assignment type constraints, precedence constraints, cycle time constraints and limits for operations and blocks, each of them are explained below:

Assignment Constraints

$\sum_k \sum_q x_{jqk} = 1 \qquad \forall j \qquad (1)$

Precedence Constraints

$\sum_k \sum_q (nk + q) * x_{jqk} + 1 \leq \sum_k \sum_q (nk + q) * x_{iqk} \qquad \forall i \epsilon IP_j, \forall j \qquad (2)$

"Not" Same Station Constraints

$\sum_q x_{jqk} + \sum_q x_{iqk} \leq 1 \qquad i, j \epsilon ES, \forall k \qquad (3)$

"Not" Same Block Constraints

$x_{jqk} + x_{iqk} \leq 1 \qquad \forall i, j \epsilon EB, \forall q, k \qquad (4)$

Operation per Block Constraints

$\sum_j x_{jqk} \leq n_o \qquad \forall q, k \qquad (5)$

Cycle time Constraints

$F_{qk} \geq x_{jqk} * t_j \qquad \forall j, q, k \qquad (6)$

$\sum_q F_{qk} \leq CT \qquad \forall k \qquad (7)$

Integrality and Sign Constraint

$X_{jqk} = \{0, 1\} \qquad \forall j, q, k \qquad (8)$

$F_{qk} \geq 0 \qquad \forall q, k \qquad (9)$

Constraint set (1) ensures that each operation is assigned to one and only one block and one and only one station. Constraint set (2) ensures that each operation can be assigned only after all preceding operations are completed in earlier block. Constraint set (3) ensures that operation pairs that have station exclusion limitation are not assigned to the same station. Constraint set (4) ensures that operation pairs which have block exclusion limitation are not assigned to the same block. Constraint set (5) ensures that the number of operations assigned to a block cannot exceed its determined limit. Constraint sets (6) and (7) ensure that block times are identified as the maximum operation time in the block, station times are calculated with summation of block times in the station and cycle time is equal to maximum station time. Constraint sets (8) and (9) represents the binary and sign requirements on decision variables.

3.2. Branch and Bound Algorithm

Attributing to the practical importance of our problem in increasing the production rate, thereby production time and cost, we aim to find optimal solutions. As an optimization approach, we select branch and bound algorithm that implicitly enumerates all feasible solutions by keeping more reasonable memory compared to our explicit enumeration techniques.

Our B&B algorithm assigns the operations to the blocks and blocks to the workstations starting from the first block of the first workstation

3.2.1. Branching Scheme

We index the operations according to their nonincreasing order of operation times, $P_1 \geq P_2 \geq ... \geq P_n$

For a partial solution with the first k stations and t^th block of station k are opened, not-yet-assigned operation i is said to be eligible if it satisfies all the following conditions:

i. All its predecessors are assigned to earlier blocks, but not to current block
ii. Operation j is not assigned to current block, if $(i,j) \epsilon$ EB or operation j is not assigned to current workstation if $(i,j) \epsilon$ ES
iii. To avoid duplication of the solutions, $i<j$ if j is already assigned to the current block
iv. If addition of operation i do not increase the cycle time beyond the upper bound

If no eligible operation exists, then we close the current block. Moreover, when the block has n_o number of operations already assigned, we close the block.

According to our indexing, i.e. i>j implies Pi≤Pj, and assignment strategy in a block, i.e., always to a higher indexed operation, the block cycle time is defined by the first assigned operation. After the first assignment the block cycle time does not change, hence, to favor our objective function we never close a block if there exists a fittable operation.

Once we close a block due to the absence of any eligible operation, we consider two types of decisions; leading the following types of nodes

- Type I - Open a new block, i.e., (t+1)st block of station k
- Type II – Open a new station, i.e., first block of station k+1

We do not consider Type II nodes if there exists an eligible not-yet-assigned operation such that

i. The addition does not violate station exclusion constraint

ii. The addition does not increase the station cycle above the cycle time of the partial solution, i.e., maximum workload among the first k-1 stations, or the lower bound on the cycle time value. Formally, we let W_{kt} = workload of the tth block of station k if $\sum_{r=1}^{t} W_{kr} + t_j \leq LB_{CT}$ then we do not close the workstation, i.e., Type II node is not considered.

We illustrate the branch and bound tree via the following simple example whose precedence network, operation times and exclusion constraints are given below.

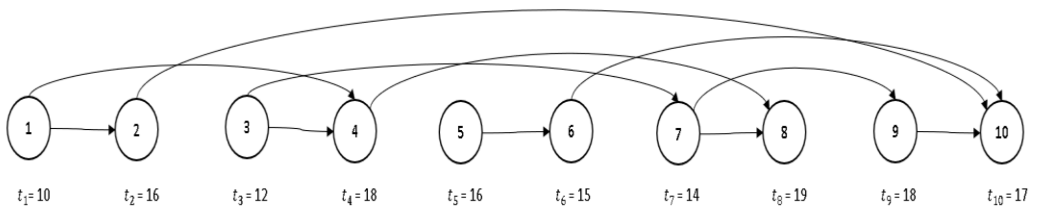

Figure 2 A precedence diagram

Note that the operations are already indexed according to the nonincreasing order.

Table 1 Operation times, block exclusion and station exclusion relations

Operation	**Process Time**	**NOT Same Station**	**NOT Same Block**
1	10	4	
2	16	6	
3	12		6
4	18	1,5,8	
5	16	4	
6	15	2	3
7	14		
8	19	4	
9	18		10
10	17		9

Assume n = 10, m=2, n_o=3, m_o=3

Initial levels of branch and bound tree are as follows:

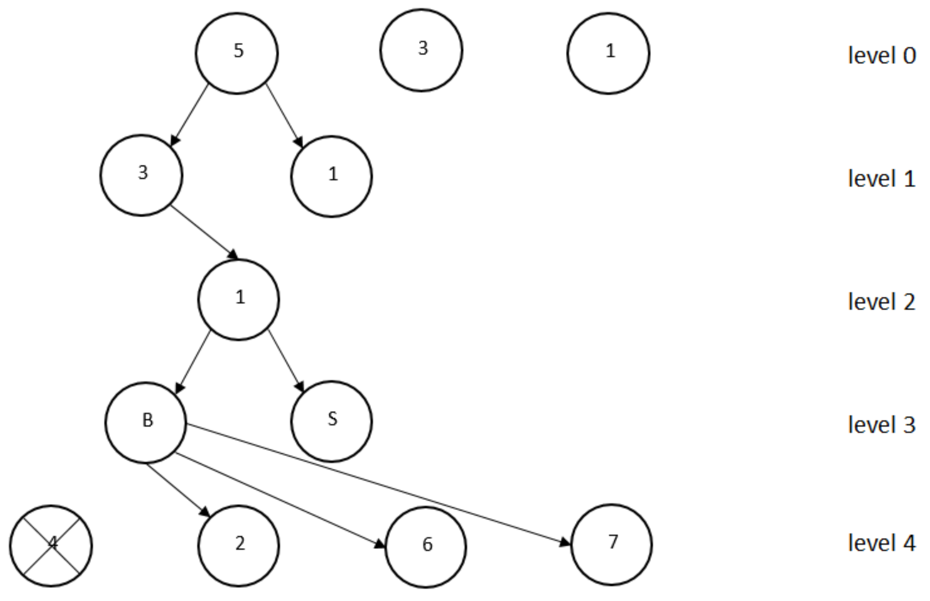

Figure 3 A representation of the B&B Tree

We start with an upper bound, i.e., a feasible solution. We update the upper bound once a feasible solution with a smaller cycle time is found. Moreover, while closing a workstation we try to find a feasible assignment of the remaining operations with the hope of reducing the upper bound.

We also find lower bounds while closing the workstations. We fathom the node whenever the lower bound is no less than the best known upper bound. Always higher levels are searched and once all nodes are fathomed, we backtrack. This is so called depth first strategy and is favored due to its low memory requirements.

We now give the detailed explanation of our lower and upper bounding procedures.

3.2.2. Lower Bounds

Given the assignments to the blocks and in the absence of the number of blocks per station requirement, TALB problem reduces to SALB problem. Hence, any lower bound derived for Type II SALB problem is valid for the TALB problem for a given block contents. Unfortunately, the block contents are not known, they are decisions. However, once they are formed by ignoring the constraints of the problem in a proper way, they can lead to the lower bounds. We now describe the formation of the groups. Assume the operations are ordered by the Longest Processing Time (LPT) rule, i.e., $t1 \geq t2 \geq \ldots \geq tn$

Note that $\left\lceil \frac{n}{n_0} \right\rceil$ is a lower bound on the number of blocks that any feasible solution may reside. Assume the optimal solution has O_b blocks and the blocks are ordered by $t^*_{b_1} \geq t^*_{b_2} \geq \ldots \geq t^*_{O_b}$ $where\ O_b \geq \left\lceil \frac{n}{n_0} \right\rceil$

Now assume we form blocks by ignoring all precedence relations, but considering the number of operations per block condition. If the operations of LPT order are put to the blocks, there will be $\left\lceil \frac{n}{n_0} \right\rceil$ blocks with the following block times:

$$t_{b_1} = Max\left\{t_1, t_2, \ldots, t_{n_0}\right\} = t_1$$

$$t_{b_2} = Max\left\{t_{n_0+1}, t_{n_0+2}, \ldots, t_{2n_0}\right\} = t_{n_0+1}$$

$$t_{b_r} = Max\left\{t_{(r-1)n_0+1}, t_{(r-1)n_0+2}, \ldots, t_{rn_0}\right\} = t_{(r-1)n_0+1}$$

$$t_{b_{\left\lceil \frac{n}{n_0} \right\rceil}} = Max\left\{t_{(\left\lceil \frac{n}{n_0} \right\rceil-1)n_0+1}, t_{(\left\lceil \frac{n}{n_0} \right\rceil-1)n_0+2}, \ldots, t_n\right\} = t_{(\left\lceil \frac{n}{n_0} \right\rceil-1)n_0+1}$$

The relation between $t^*_{b_r}$ and t_{b_r} is set as follows:

$$t^*_{b_1} = t_1 = t_{b_1}$$

$t^*_{b_2} \geq t_{b_2}$

$t^*_{b_r} \geq t_{b_r}$

$t^*_{b_{\left\lceil \frac{n}{n_0} \right\rceil}} \geq t_{b_{\left\lceil \frac{n}{n_0} \right\rceil}}$

This follows if $t_{b_r}{}'s$ are used as block processing times, an optimal solution to the SALB problem would lead to a lower bound on the optimal solution of TALB problem. As the SALB Type II is an NP-hard problem, finding such a lower bound requires an exponential effort. Recognizing this fact, we use two lower bounds proposed for the SALBP by Klein and Scholl (1996).

The lower bounds are stated below:

1- $max\left\{\left\lceil\frac{\sum t_i}{m}\right\rceil, max_i\{t_i\}\right\}$
2- $max\left\{\sum_{i=0}^{k} t_{km+1-i} \mid k = 1, \dots, \left\lfloor\frac{n-1}{m}\right\rfloor\right\}$ $such\ that\ t_{i+1} \geq\ t_i\ \ i = 1, \dots, n-1$

We used two lower bounds to define the lower bounds to our TALB problem, using t_{b_i} in place of t_i.

Formally,

$$LB_1 = max\left\{\left\lceil\frac{\sum_{r=1}^{m} t_{b_i}}{m}\right\rceil, t_{b_1}\right\}$$

$$LB_2 = max\left\{\sum_{i=0}^{k} t_{b_{km+1-i}} \mid k = 1, \dots, \left\lfloor\frac{\left\lceil\frac{n}{n_0}\right\rceil - 1}{m}\right\rfloor\right\}$$

Example.

- Consider an instance of a TLBP with the following parameters.

n=25, m=3, n_o=3

Table 2 Operation times, precedence relations, block exclusion constraint

Operation	Process Time	Block Exclusion	Predecessors Set	Successors Set
21	20		3,5,6,7,9,11,12,13,14,15,16	22,24,25
8	19		1,3,4,7	
12	19	6,7	11	13,14,15,16,17,18,19,20,21,22,24,25
9	18	10	3,7	10,13,14,15,16,17,18,20,21,22,23,24,25
4	18		1,3	8,18
10	17	9	1,2,3,5,6,7,9	23,24,25
13	16	2,22	3,7,9,11,12	14,15,16,17,18,20,21,22,24,25
5	16			6,10,21,22,23,24,25
2	16	13	1	10,23,24,25
14	15		3,7,9,11,12,13	15,16,17,18,20,21,22,24,25
6	15	3,12,22	5	10,21,22,23,24,25
22	14	6,13	3,5,7,9,11,12,13,14,15,16,21	
18	14	19	1,3,4,5,7,9,11,12,13,14,17	
7	14	12	3	8,9,10,13,14,15,16,17,18,20,21,22,23,24,25
15	13		3,7,9,11,12,13,14	16,17,18,21,22,24,25
3	12	6		4,7,8,9,10,13,14,15,16,17,18,20,21,22,23,24,25
11	12			12,13,14,15,16,17,18,19,20,21,22,24,25
20	12	25	3,7,9,11,12,13,14,19	
19	11	18	11,12	20
23	11	1,24	1,2,3,5,6,7,9,10	24,25
17	11		3,5,7,9,11,12,13,14,15	18
16	10		3,7,9,11,12,13,14,15	21,22,24,25
1	10	23		2,4,8,10,18,23,24,25

Table 2 (continued).

24	10	23,25	1,2,3,5,6,7,9,10,11,12,13,14,15,16,21,23	25
25	10	20,24	1,2,3,5,6,7,9,10,11,12,13,14,15,16,21,23,24	

- Assign operations to the groups in non-ascending order of t_i's.

Table 3 Operation assignments to the blocks

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

Total block time is calculated as 126 units. Time per station is rounded up to 42 units as m=3. Maximum group time, 20 units, is not greater than time per station, therefore initial LB1 is equal to 42 units.

LB2 is equal to maximum of (t_{b_3}+ t_{b_4}; t_{b_7}+t_{b_6}+ t_{b_5}) which is (31; 37) 37.

Finally, LB1>LB2, then the global LB is 42.

We improved LB1 and LB2 by incorporating block exclusion constraint to further increase t'_{b_r}values. In doing so, for any formed block we checked for the presence of the block exclusion pairs, starting from the first block. If there is a pair in block 1, we move the shorter operation to block 2. If there are more than one pair, we shift final set of the shorter operations of all pairs and shift longest operation in the set to block 2. If any operation is shifted to block 2, we proceed to block 3 and repeat the process in block 1. If no operation is shifted to block 2 then we repeat the process in block 1 to block 2. In general, if any operation is shifted from block r to r+1 then we proceed to block r+2. If no operation is shifted from block r to r+1 then we look for possibility of shifting from block r+1 to r+2.

Example (cont.)

- Search for improvements

In group 2, operation 9 and 10 is a block exclusion pair. Operation 10 has smaller operation time therefore it is shifted to group 3.

Table 4 Improved operation assignments to the blocks (step-1)

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

In group 4, operation 6 and 22 is a block exclusion pair. Operation 22 has smaller operation time therefore it is shifted to group 5.

Table 5 Improved operation assignments to the blocks (step-2)

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

There is no other overlap and the improved LB is calculated as 43 units. (LB1=43, LB2=37)

We let t''_{b_r} be the processing time of block r after including block exclusion constraint and state the following relation $t''_{b_r} \geq t'_{b_r}$ r $= 1, \dots, \bar{B}$ where $\bar{B}$ is the number of blocks formed by excluding block exclusion relation. Using t''_{b_r}as block processing times, we employ the lower bounding procedures of Klein and Scholl (1996) and let the resulting bounds be LB_3 if the first lower bound of Klein and Scholl (1996) is used and LB_4 if the second bound of Klein and Scholl (1996) is used.

Note that, $LB_3 \geq LB_1$ and $LB_4 \geq LB_2$ as $t''_{b_r} \geq t_{b_r}$for all r and the same lower bound is applied on given blocks, for 'LB_1 and LB_3' and 'LB_2 and LB_4'

In our branch and bound algorithm, we apply LB_1 and LB_2 first with hope of eliminating the nodes without a need for more powerful bounds. Hence LB_1 first, LB_2 second are used as filtering mechanisms. If $LB_1 < UB$ and $LB_2 < UB$ where UB is the best-known cycle time value, we find LB_3. If $LB_3 < UB$ then we find LB_4. If $LB_i \geq UB$ for any i then we discard the node from further considerations.

3.2.3. Upper Bounds

Our B&B algorithm starts with an initial upper bound that is found through a heuristic procedure. The heuristic procedure has two steps

Step 1. Construction Step

Step 2. Improvement Step

In construction step, we first form the Longest Processing Time (LPT) list and using the list we form feasible blocks. A feasible block is the one that has at most n_o operations, no block exclusion and no precedence relation pair exists. An operation is said to be ineligible to a block if its addition does not lead to a feasible group.

We start from blocks starting from the first eligible operation of the list and close the block, whenever no eligible operation exists. We continue with forming blocks till all operations in the list are assigned to one block.

Let NR denote the number of resulting blocks. We try to assign these NR blocks to the workstations.

While assigning we aim to balance the workload by even distribution of the blocks to the workstation. The average number of blocks per workstation is $\left\lceil \frac{NR}{m} \right\rceil$ and this average is the target value of our assignments.

In workstation assignments, we treat the blocks as operations. Starting from the first workstation, we assign the first eligible block to the closest the workstation when there are $\left\lceil \frac{NR}{m} \right\rceil$ block assignments. A block is eligible for the current workstation if all predecessors have appeared in one of the assigned blocks and no violation of station exclusion limitations.

The construction step ends whenever all blocks are assigned.

The even distribution of the blocks is somewhat favored by equal number of block assignment per workstation. However equal number may not mean fair distribution of the workload among the workstations. Recognizing this fact, we perform an improvement step, in which first block assignment among the workstations are exchanged and then the operation assignment among the blocks are exchanged.

The exchanges are realized between the block of the most loaded workstation and any other workstation, if the resulting exchange does not violate feasibility and improves the cycle time. Any exchange between two blocks is realized at most once. Once no block pair exchange improves the cycle time, we proceed to non-improving feasible exchanges till all block pairs are considered. The best cycle time solution is used for operation exchanges. The operation exchanges among the blocks are performed in the same way with those block exchanges among stations.

Below is the stepwise description of or initial upper bound procedure.

Procedure. Initial feasible solution

Step 1.

```
• For each block b
    • For each operation i (in descending order of operation times)
        ▪ If $\sum_b X_{ib} = 0$ and operation i has no block exclusion pair on block b
          and all predecessors of operation i are assigned and number of
          operations in block b < $n_o$
                ❖ Assign operation i to block b ($X_{ib} = 1$)
        ▪ End If
    • End For
• End For
```

Step 2.

```
• For each station k
    • For each block b (in descending order of block times)
        ▪ If $\sum_k Y_{bk} = 0$ and block b has no station exclusion pair on station k
          and all predecessors of block b are assigned and number of blocks in
          station k < $m_o$ and number of blocks in station k < $\left\lceil \frac{NR}{m} \right\rceil$
                ❖ Assign block b to station k ($Y_{bk} = 1$)
        ▪ End If
    • End For
• End For
```

Step 3.

```
For each iteration
    ❖ Determine the most loaded station ($k_L$)
    • For each station k
        • For each block b
            ▪ If $Y_{bk} = 1$ and $Y_{pkL} = 1$ and time of station k + time block p – time
              of block b < station time of most loaded station and no direct or
              indirect precedence relation between blocks p & b and changing
              the places of blocks p & b do not violate the precedence & station
              exclusion constraints and blocks b & p are not changed before
                ➢ Assign block b to the most loaded station
                ➢ Assign block p to station k
            ▪ End If
        • End For
    • End For
    ❖ Control that any change is done or not
        ▪ If no change between blocks is made
            ➢ Make a change among blocks that worsen the objective value
              without violating the precedence and station exclusion constraint.
            ➢ Store the changed blocks
        ▪ End If
End For when no more exchanges are available
```

Step 4.

```
• For each iteration
    ❖ Determine the most loaded station ($k_L$)
    • For each station k
        • For each block b
            • For each operation i
            ▪ **If** $Y_{bk} = 1$ **and** $Y_{pkL} = 1$ **and** $X_{ib} = 1$ **and** $X_{jp} = 1$ **and**
              operation j has highest operation time in block p **and** tj>ti
              **and { (**time of station k + tj – ti < station time of most loaded
              station **and** time of block b= $t_i$**) or (**time of station k + tj –
              time of block b  < station time of most loaded station **and**
              time of block b = tj**) or** (time of block b > ti **and** time of
              block b > tj) **} and** no direct or indirect precedence relation
              between operations i & j **and** changing the places of
              operations i & j do not violate the precedence & station
              exclusion & block exclusion constraints **and** operations i &
              j are not changed before
                ➢ Assign operation i to block p assigned in the most
                  loaded station
                ➢ Assign operation j to block b assigned in station k
            ▪ **End If**
          • **End For**
      • **End For**
  • **End For**
❖ Control that any change is done or not
    ▪ **If** no change between operations is made
        ➢ Make a change among operations that worsen the objective value
          without violating the precedence, block and station exclusion
          constraint.
        ➢ Store the changed operations.
    ▪ **End If**

**End For** when no more exchanges are available
```

We employ the above procedure to start the branch and bound algorithm. We update the upper bound, UB, whenever a complete solution with smaller cycle time is reached. We make those updates whenever closing a workstation as follows:

We find an upper bound through construction steps of initial upper bound for the unassigned operations, and the upper bound for the partial solution, UB_p, is $Max\ \{Cycle\ time\ of\ unassigned\ tasks, Lower\ bound\ of\ the\ partial\ solution = LB_p\}$

If $UB_p < UB$ then we set $UB = UB_p$

If $UB_p = LB_p$ then we fathom the node as the best solution from this node has an objective function value of LB_p , i.e., one of the optimal solutions emanating from the node is already available through its upper bound.

CHAPTER 4

COMPUTATIONAL EXPERIMENT

In this chapter, we discuss the performance of the branch and bound algorithm and the mathematical model. In Section 4.1, the data generation scheme is introduced. The performance measures used for evaluating the performance of the B&B algorithm and mathematical model are reported. In Section 4.2, we report on the preliminary experiment used to select the parameter values. Section 4.3 discusses the results of the main experiment.

The models are modeled using GAMS (version 23.9.5) and solved by software CPLEX 12. The B&B algorithm is coded with C++ programming language. All experiments are conducted on a computer with INTEL (R) CORE (TM) i7-4770S CPU @ 3.10 GHz 16.00 GB RAM.

4.1. Data Generation and Performance Measures

Guschinskaya and Dolgui (2006) provide a set of benchmark problems for 25, 50 and 100 operations problems with 4, 5, 10, 15 stations. We modify their networks according to the selected numbers of immediate predecessors, block and station exclusion relations. They use operation times between U[10,20]. We try another distribution U[1,20] and see that the generated instances are harder to solve ones. We use the following sets of the number of operations per block (n_o) and the number of blocks per station (m_o)

For preliminary experiment we set n=15, m=5, n_o=3, m_o=2 and in main experiment we set n=15, 20, 25, 30 and m=4, 6.

For each value of n, m, n_o, m_o, we generate and solve 10 problem instances. We use the following measures to evaluate the performance:

- Average and maximum (worst case) operation times expressed in Central Processing Unit (CPU) seconds (for mathematical model and branch and bound algorithm)
- Average and maximum number of nodes (for branch and bound algorithm)
- Average and maximum relative deviation from the optimal solution (for initial upper bounds)

We set a termination limit of one hour for the execution of both mathematical model and branch and bound algorithm.

4.2. Preliminary Experiment

We perform a preliminary experiment to see the effects of the parameters (operation times, precedence structure) and bounding mechanisms (initial upper bound at root node, upper bound and lower bounds for the partial solutions (nodes)). We base our main experiment on the results of the preliminary experiment.

We select 15 operations and 5 workstations instances with 2 blocks per station and 3 operations per block.

We test two uniform distributions, four sets of precedence structures and four versions of B&B algorithm in our preliminary experiment. The abbreviations used to define the combinations are as stated below:

Set I – Number of Immediate Predecessors (IP) =16
Number of Block Exclusion Relations (BE) = 5
Number of Station Exclusion Relations (SE) = 7
Set II – Number of IPs =18
Number of BEs = 5

Number of SEs = 7

Set III – Number of IPs =16

Number of BEs = 5

Number of SEs = 9

Set IV – Number of IPs =16

Number of BEs = 7

Number of SEs = 7

B&B I – B&B algorithm using all bounding mechanisms

B&B II – B&B algorithm using all bounding mechanisms but not upper bounds at nodes

B&B III – B&B algorithm using all bounding mechanisms but not initial upper bound

B&B IV – B&B algorithm using all bounding mechanisms but not lower bounds

First the effect of operation time distribution is tested on two discrete uniform distributions: U[1,20] and U[10,20]. U[10,20] is the distribution used in Guschinskaya and Dolgui (2006) and U[1,20] is considered to see the effect of higher variance.

Table 6 reports the average and worst-case performance results (number of nodes and CPU time) for B&B I and B&B II, and precedence sets Set I and Set II.

Table 6 The effect of operation time distributions, n=15, m=5, no=3, mo=2

Operation Time Distribution			**Number of Nodes**		**CPU Time**	
			Avg	**Max**	**Avg**	**Max**
U[1,20]	B&B I	Set I	999,447	2,534,569	46.22	121.06
		Set II	249,137	607,351	12.13	28.25
	B&B II	Set I	999,447	2,534,569	30.65	79.66
		Set II	249,137	607,351	8.02	18.81
U[10,20]	B&B I	Set I	382,284	1,088,376	16.38	48.83
		Set II	221,367	343,263	10.32	15.74
	B&B II	Set I	382,284	1,088,376	11.20	31.62
		Set II	221,367	343,263	7.00	10.43

As can be observed from Table 6 for both versions of the B&B algorithm, and for selected precedence structures, U[1,20] yields to harder-to-solve instances than U[10,20]. Note that B&B I solves the instances with U[1,20] in 46.22 and 12.13 seconds on average for Set I and Set II respectively. When U[10,20] is used, the same combination is used, the respective CPU times reduce to 16.38 and 10.32. With B&B II, the CPU times are 30.65 seconds and 8.02 seconds when U[1,20] is used, for Set I and Set II, respectively. The respective CPU times significantly reduce to 11.20 seconds and 7.00 seconds.

This reduction is due to the fact, when the variance is lower that is the operation do have more similar times, the lower bounds and upper bounds behave better as their values are expectedly closer to the optimal cycle time value.
In the rest of computational experiments and main experiment, we continue with harder combination, i.e., U[1,20].
Next, the effect of the precedence relations is analyzed through four sets. Each set is formed by varying one precedence type while keeping the two others at the same level. We report the results in Table 7 for all sets through B&B I and B&B II.

Table 7 The effect of precedence structure on the performance, U[1,20], n=15, m=5, no=3, mo=2

Set	**Precedence Relation**			**Algorithm**	**Number of Nodes**		**CPU Time**	
	IP	**BE**	**SE**		**Avg**	**Max**	**Avg**	**Max**
I	16	5	7	B&B I	999,447	2,534,569	46.22	121.06
				B&B II	999,447	2,534,569	30.65	79.66
II	18	5	7	B&B I	249,137	607,351	12.13	28.25
				B&B II	249,137	607,351	8.02	18.81
III	16	5	9	B&B I	941,687	2,389,513	42.82	112.01
				B&B II	941,687	2,389,513	28.47	74.01
IV	16	7	7	B&B I	964,196	2,465,089	44.07	114.60
				B&B II	964,196	2,465,089	29.18	75.52

As can be observed form the table, a decrease in one of precedence relations, while keeping the other relations at the same level, increases the difficulty in attaining optimal solutions. Note that when number of BEs and SEs are fixed at 5 and 7, respectively; an increase in number of IPs from 16 to 18, decreases the average CPU times from 46.22 seconds to 12.13 seconds for B&B I and from 30.65 seconds to 8.02 seconds for B&B II, respectively.

When the number of IPs and number of SEs are set at 16 and 7, respectively, an increase in number of BEs from 5 to 7, decreases the average CPU times form 46.22 seconds to 44.07 seconds for B&B I. For B&B II, the reduction is from 30.65 seconds to 29.18 seconds.

When the number of IPs and number of BEs are set at 16 and 5, respectively, an increase of SEs from 7 to 9, decreases the B&B I CPU times from 46.22 seconds to 42.82 seconds and B&B II CPU times from 30.65 seconds to 28.47 seconds. Therefore, Table 7 also shows that the effects of number of BEs and SEs are similar, and the most effective reductions are observed by increasing the number of IPs.

Based on those results, we continue our experiments with two levels of precedence relations: low and high, in our main runs.

We finally analyze the effects of bounding mechanisms through the defined B&B algorithms (B&B I, B&B II, B&B III, B&B IV), and report the results in Table 8.

Table 8 The effect of bounding mechanisms on the performance, U[1,20], n=15, m=5, no=3, mo=2

Algorithm Used		**Number of Nodes**		**CPU Time**	
		Avg	**Max**	**Avg**	**Max**
B&B I	Set I	999,447	2,534,569	46.22	121.06
	Set II	249,137	607,351	12.13	28.25
B&B II	Set I	999,447	2,534,569	30.65	79.66
	Set II	249,137	607,351	8.02	18.81
B&B III	Set I	1,021,473	2,560,931	46.86	121.22
	Set II	260,641	607,351	12.64	28.63

Table 8 (continued).

B&B IV	Set I	33,808,835	59,724,213	645.83	1,101.67
	Set II	8,525,927	20,117,919	156.40	340.29

As can be observed form Table 8, the most significant reductions are due to the lower bounds. For Set I, the average CPU times are reduced to 46.22 seconds from 645.83 seconds, hence several folds, by incorporations the lower bounds. Hence there are found very effectively in controlling the size of the search. The number of nodes reduces to 249,137 from 8,525,927 similar results hold for Set II.

The comparison of B&B I and B&B III, shows that the initial upper bound is also effective in enhancing the efficiency of the search. The number of nodes increases to 1,021,473 from 999,447 and to 260,641 from 249,137 for Set I and Set II respectively once the initial upper bound is removed. The associated effects on CPU times are from 46.86 seconds to 46.22 seconds and from 12.64 seconds to 12.13 seconds, for Set I and Set II, respectively. Hence, the effort spent to find the initial upper bounds is less than the reduction it brings.

The analysis of B&B I and B&B II reveals that using upper bounds at intermediate nodes has adverse effect on the efficiency. The CPU times increase to 46.22 seconds from 30.65 seconds for Set I, to 12.13 seconds from 8.02 seconds for Set II. Hence the extra effort spends in calculating the intermediate upper bounds could not be justified through the reduction it brings to the nodes.

Based on those results, we decided to use B&B II (B&B with lower bounds and initial upper bound) in our main experiment.

4.3. Main Experiment

Based on the results of our preliminary experiments, we design an experiment to test the performance of our mathematical model and branch and bound algorithm. We generate the operation times from U[1,20] and, hence tackle with harder instances.

We set the number of operations to 15, 20, 25. The number of stations is set to 4 and 6. The number of operations per block is set to 2 and 4. The number of blocks per station is set to 6.

We generate the following two precedence structures.

Set Loose (L) – Number of Immediate Predecessors (IP) = 0.25* $\frac{n*(n-1)}{2}$

Number of Block Exclusion Relations (BE) = 0.15* $\frac{n*(n-1)}{2}$

Number of Station Exclusion Relations (SE) = 0.05* $\frac{n*(n-1)}{2}$

Set Tight (T) – Number of IPs = 0.30* $\frac{n*(n-1)}{2}$

Number of BEs = 0.20* $\frac{n*(n-1)}{2}$

Number of SEs = 0.05* $\frac{n*(n-1)}{2}$

We put a termination limit of 1 hour to the execution of both mathematical model and branch and bound algorithm.

We first analyze the results of the model and B&B algorithm and report the results in Tables 9, 10 and 11, for n=15, 20 and 25 respectively.

We include the unsolved instances CPU time (3600 sec) to the average CPU time. Those instances are considered to evaluate the number of nodes.

Table 9 The Computational Results, n=15

m	no	Precedence Structure	Mathematical Model		Branch and Bound Algorithm			
			CPU Time		CPU Time		Number of Nodes	
			Avg	Max	Avg	Max	Avg	Max
4	4	L	10.594	33.34	4.02	12.35	120,638	370,915
6	4	L	24.502	62.28	5.90	17.49	147,119	442,044
4	4	T	7.144	29.55	3.94	9.98	123,100	321,900
6	4	T	29.683	53.77	6.22	15.68	156,063	391,243

Table 10 The Computational Results, n=20

m	**no**	**Precedence Structure**	**Mathematical Model**		**Branch and Bound Algorithm**			
			CPU Time		**CPU Time**		**Number of Nodes**	
			Avg	**Max**	**Avg**	**Max**	**Avg**	**Max**
4	4	L	5.81	9.10	89.56	258.91	1,534,560	4,514,995
6	4	L	285.19	1,502.67	177.23	797.66	2,301,440	9,777,423
4	4	T	4.90	10.85	304.36	1,003.52	3,492,058	12,365,867
6	4	T	62.78	273.20	589.52	3,600.00	4,928,949	29,842,113
4	2	L	7.22	15.24	33.15	135.03	470,547	2,043,520
6	2	L	525.35	3,600.00	151.63	557.36	1,936,901	6,572,800
4	2	T	5.12	12.49	75.23	283.18	1,082,096	3,971,212
6	2	T	70.49	131.59	253.41	1,597.76	3,221,780	20,597,666

Table 11 The Computational Results, n=25

m	**no**	**Precedence Structure**	**Mathematical Model**		**Branch and Bound Algorithm**			
			CPU Time		**CPU Time**		**Number of Nodes**	
			Avg	**Max**	**Avg**	**Max**	**Avg**	**Max**
4	4	L	9.54	27.54	2,200.78	3,600.00	19,540,242	33,166,435
6	4	L	709.48	2,225.11	2,863.78	3,600.00	22,270,762	32,589,921
4	4	T	7.99	24.58	1,552.22	3,600.00	17,082,934	40,060,230
6	4	T	324.78	689.54	2,156.72	3,600.00	19,634,759	34,927,475

The results show that the performance of the mathematical model highly depends on the increases in the number of operations, and number of stations. Note that when there are 15 operations, n_o=4 and m_o=6, the increase in number of stations from 4 to 6, increases the average CPU time of the model from 7.14 seconds to 29.68 seconds i.e., more than 4 times. For this combination, the B&B gives average CPU times of 3.94 seconds and 6.22 seconds for 4 and 6 stations, respectively. That is the increase is less than 2 times. This shows that the model is more sensitive to the increase in the problem size parameters, as those parameters directly affect the number of decision variables.

It can be observed from the tables that the number of operations is a significant parameter that affects the CPU times. This is due to the fact that the depth and width of the search tree are both defined by the number of operations. Note that, for the B&B algorithm when m=6, n_o=4, m_o=6, the CPU times and the number of nodes increase from 5.9 seconds to 177.23 seconds and from 147,119 to 2,301,440 respectively, when N increases from 15 to 20. Those increases are more significant for higher N, i.e., from 20 to 25.

We also observe the significant effect of the number of operations per block on the performance of the B&B algorithm. The performance improves as the number of operations per block decreases. This is due to the fact that the bounds that rely on relaxed grouping idea consider less operations, produce groups (blocks) that are closer their actual ones. Note that when N=20, m=4, the average CPU time is 304.36 seconds and 75.23 seconds, for 4 operations per block and 2 operations per block cases, respectively.

We observe that the B&B outperforms the mathematical model when there are 15 operations. This superiority holds for all problem combinations.

When there are 20 and 25 operations, the performance of the mathematical model is better, in most of the problem combinations. For 20 operations we also generate instances when the operation times from U[10,20] and report the results in Table 12.

Table 12 The Computational Results, n=20, U[10,20]

m	**no**	**Precedence Structure**	**Mathematical Model**		**Branch and Bound Algorithm**			
			CPU Time		**CPU Time**		**Number of Nodes**	
			Avg	**Max**	**Avg**	**Max**	**Avg**	**Max**
4	4	L	7.42	16.55	74.64	113.34	1,260,935	1,950,137
6	4	L	380.79	1188.76	91.59	142.26	1,328,559	2,087,455
4	4	T	8.37	39.00	149.69	252.87	2,517,094	4,185,771
6	4	T	206.29	396.41	210.50	337.22	3,130,819	5,042,125
4	2	T	5.78	9.70	56.48	83.48	843,558	1,347,912
6	2	T	194.16	463.77	81.67	135.01	985,754	1,617,522

Table 12 (continued).

4	2	L	10.56	28.45	22.30	36.88	310,928	554,869
6	2	L	526.96	1118.64	35.46	74.56	430,279	930,613

We observe from Table 12 that when the operation times are from U[10,20], the instances could be solved easily with the B&B. We observe the superiority of the B&B algorithm over the mathematical model for the hardest combinations when there are 6 workstations. Note that when there are 6 workstations the CPU times by the B&B algorithm are 91.59, 210.5, 81.67 and 35.46 seconds and the respective by the mathematical model are 380.79, 206.29, 194.16 and 526.96 seconds, respectively.
When N=30 and the operation times are from U[1,20], both model and B&B could hardly return any solution in one hour.

Our suggestions for instances with more than 25 operations are in three ways:

1. Initial Upper Bounding Procedure
2. Decomposition Based Approaches
3. Truncated Branch and Bound Algorithm

We now investigate the details of each suggestion.

1. Initial Upper Bounding Procedure

We report the average and worst-case relative deviations of the heuristic solution form the optimal solution in Table 13.

Table 13 The Performance of the Initial Upper Bound

n	**m**	**no**	**Precedence Structure**	**(UB-OPT)/OPT**	
				Avg	**Max**
15	4	4	T	26.30%	64.71%
15	6	4	T	20.07%	38.89%
15	4	4	L	31.84%	57.58%

Table 13 (continued).

15	6	4	L	37.25%	75.00%
20	4	4	T	10.21%	20.00%
20	6	4	T	29.01%	64.71%
20	4	2	T	10.21%	20.00%
20	6	2	T	29.01%	64.71%
20	4	4	L	19.73%	37.21%
20	6	4	L	43.47%	88.00%
20	4	2	L	5.78%	27.50%
20	6	2	L	41.30%	86.67%
25	6	4	T	26.15%	48.57%
25	6	4	L	25.05%	56.67%

As can be observed from Table 13 the relative deviations are mostly below or around 30%. The performances do not deteriorate with increases in problem size like number of operations, number of stations. These satisfactory performances are achieved in negligible time. Hence one may improve those solutions and get even higher quality balances for large sized problem instances.

2. Decomposition Based Approaches

We may decompose the problem into small subproblems and solve each subproblem to optimality by our B&B algorithm. The precedence network can be used for defining a subset of operations that would be processed in the first defined number of stations, to form the first subproblem. Another, subset of operations from the downstream parts of the network with new set of stations could be another subproblem, and so on.

After solving the subproblems to optimality, the resulting solutions are merged to get a feasible for the overall line. There may be rooms to improve this solution enroute to minimizing the cycle time.

3. Truncated Branch and Bound Algorithm

Recall that we run the B&B algorithm for one hour and terminate its execution. The best solution recorded at the termination limit of one hour is at least good as our initial upper bound.

To evaluate the performance of the truncated B&B algorithm, we record the node at which the optimal solution is reached for the problems that could be solved in one hour. After the node optimality, the nodes are evaluated to verify the optimality. The results are reported in Table 14.

Table 14 Optimality Node Analysis

n	**m**	**no**	**Precedence Structure**	**Number of nodes**		**Optimality Node**		**Opt. Node/Number of nodes**	
				Avg	**Max**	**Avg**	**Max**	**Avg**	**Max**
15	4	4	T	123,100	321,900	29,263	107,237	25.89 %	73.69 %
15	6	4	T	156,063	391,243	44,011	139,260	31.37 %	59.52 %
15	4	4	L	120,638	370,915	56,130	237,584	40.56 %	92.07 %
15	6	4	L	147,119	442,044	62,529	396,743	41.42 %	89.75 %
20	4	4	T	3,492,058	12,365,867	489,354	2,175,790	16.67 %	47.31 %
20	6	4	T	4,928,949	29,842,113	1,074,122	3,549,750	42.81 %	92.71 %
20	4	2	T	1,082,096	3,971,212	213,726	1,166,022	22.36 %	57.32 %
20	6	2	T	3,221,780	20,597,666	579,728	1,613,857	43.09 %	91.18 %
20	4	4	L	1,534,560	4,514,995	417,366	2,704,147	24.78 %	61.09 %
20	6	4	L	2,301,440	9,777,423	954,438	2,290,663	56.55 %	85.61 %
20	4	2	L	470,547	2,043,520	263,816	1,609,440	40.79 %	91.20 %
20	6	2	L	1,936,901	6,572,800	1,058,058	2,392,474	72.66 %	95.74 %
25	4	4	T	17,082,934	40,060,230	8,229,275	24,684,752	50.28 %	81.47 %
25	6	4	T	19,634,759	34,927,475	8,668,262	17,236,805	46.74 %	87.79 %
25	4	4	L	19,540,242	33,166,435	9,087,365	21,315,681	56.87 %	87.05 %

Table 14 (continued).

25	6	4	L	22,270,762	32,589,921	7,829,705	17,983,034	41.29 %	79.33 %

The table shows that in worst case the B&B algorithm finds the optimal solutions just before the terminations. There are some combinations when the maximum optimality node over total node ratios are even more than 90%, implying that only 10% of the nodes are used to verify optimality. Despite these high worst case percentages, almost all average case percentages are lower than 50%. This implies that in majority of the instances, the optimal solutions are found at very early stages of the B&B algorithm.

Note that when n=20, the average percentage is 40.79%, below 50%, even the worst-case percentage is 91.2, above 90%.

Similar observations hold for all problem combinations, the optimal solutions are found in the first half of the search for majority of the instances and close to termination for some few instances.

Hence, we can use our B&B algorithm with a proper termination limit to get approximate solutions.

CHAPTER 5

CONCLUSIONS

In this thesis, we consider Type II Transfer Line Balancing Problem that assigns the operations to the blocks and blocks to the stations so as to minimize the cycle time.

We formulate the problem as a mixed integer linear program. We propose a branch and bound algorithm along with efficient lower and upper bounding mechanisms. We see that the lower bounding mechanisms are very effective in enhancing the efficiency of the branch and bound algorithm. We find that the heuristic that we propose to get an initial upper bound for the branch and bound algorithm, can be used as a solution approach for large sized instances. Our computational experiments have revealed that both the mathematical model and branch and bound algorithm can solve instances with up to 25 operations and 6 workstations. To solve the problems of larger sizes, we propose to branch and bound based solution algorithms: decomposition algorithms and truncated branch and bound algorithms.

We observe that the number of operations, number of workstations and number of operations per block are important problem size parameters that affect the difficulty of attaining optimal solutions. Moreover, the precedence structure and spread of the operation times are also significant factors that influence the efficiency of the branch and bound algorithm.

To the best of our knowledge, we present the first study for the Type II TLBP. We hope our study motivates the future studies for the extensions of our model like parallel stations, U-shaped stations, mixed models. The rebalancing for the TLBP is an unexplored area that may attract future researchers.

REFERENCES

Arcus, A.L., 1966, COMSOAL: A computer method of sequencing operations for assembly lines. *International Journal of Production Research*, 4, 259–277.

Battaia, O., Delorme, X., Dolgui, A., Frederic, G. and Finel, B., 2015, Flow line balancing problem: A survey. *2015 International Conference on Industrial Engineering and Systems Management (IESM)*.

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

Battaïa, O., Dolgui, A. and Guschinsky, N., 2016, Heuristics for batch machining at reconfigurable rotary transfer machines. *IFAC-PapersOnLine*, *49*, 491–496.

Battaïa, O., Dolgui, A. and Guschinsky, N., 2016, Integrated process planning and system configuration for mixed-model machining on rotary transfer machine. *International Journal of Computer Integrated Manufacturing*, *30*, 910–925.

Battaïa, O., Gurevsky, E., Makssoud, F. and Dolgui, A., 2012, Equipment location in machining transfer lines with multi-spindle heads. *Journal of Mathematical Modelling and Algorithms in Operations Research*, *12*, 117–133.

Dolgui, A., 2009, An approach to transfer line balancing via a special set partitioning problem. *13th IFAC Symposium on Information Control Problems in Manufacturing*.

Dolgui A., Fine B., Guschinsky N., Levin G., and Vernadat F., 2005a, A heuristic approach for transfer line balancing. *Journal of Intelligent Manufacturing*, 16, 159–171.

Dolgui, A., Guschinsky, N. and Levin, G., 2000, Approaches to balancing of transfer line with block of parallel stations, Preprint No. 8, 42 pages, Institute of Engineering Cybernetics/University of Technology of Troyes, Minsk.

Dolgui, A., Guschinsky, N. and Levin, G., 2005, Transfer line balancing by a combined approach. *IFAC Proceedings Volumes*, *38*, 277–282.

Dolgui, A., Kovalev, S., Kovalyov, M. Y., Malyutin, S. and Soukhal, A., 2018, Optimal workforce assignment to operations of a paced assembly line. *European Journal of Operational Research*, *264*, 200–211.

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

Guschinskaya, O. and Dolgui, A., 2006, A comparative evaluation of exact and heuristic methods for transfer line balancing problem. *IFAC Proceedings Volumes, 39*, 413–418.

Guschinskaya, O. and Dolgui, A., 2007, Balancing transfer lines with multi-spindle machines using grasp. *IFAC Proceedings Volumes*, *40*, 511–516.

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

Patterson, J. H. and Albracht, J. J., 1975, Technical Note: Assembly-line balancing: zero-one programming with fibonacci search. *Operations Research, 23*, 166–172.

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

Sancı, E. and Azizoğlu, M., 2017, Rebalancing the assembly lines: exact solution approaches. *International Journal of Production Research*, *55*, 5991–6010.

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

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