5-1-2014

A Branch and Bound Method for Sum of Completion Permutation Flow Shop

Swapna Kodimala
University of Nevada, Las Vegas, swapnakodimala@gmail.com

A BRANCH AND BOUND METHOD FOR SUM OF COMPLETION PERMUTATION FLOW SHOP

By

Swapna Kodimala

Bachelor of Technology, Information Technology
Jawaharlal Nehru Technological University, India
2012

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Computer Science

School of Computer Science
Howard R. Hughes College of Engineering
The Graduate College

University of Nevada, Las Vegas
May 2014

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Swapna Kodimala

entitled

A Branch and Bound Method for Sum of Completion Permutation Flow Shop

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science
School of Computer Science

Wolfgang Bein, Ph.D., Committee Chair

Ajoy K. Datta, Ph.D., Committee Member

Laxmi Gewali, Ph.D., Committee Member

Zhiyong Wang, Ph.D., Graduate College Representative

Kathryn Hausbeck Korgan, Ph.D., Interim Dean of the Graduate College

May 2014

ABSTRACT

A Branch and Bound Method for Sum of Completion Permutation Flow Shop

By

Swapna Kodimala

Dr. Wolfgang Bein, Examination Committee Chair

Professor of Computer Science

University of Nevada, Las Vegas

We present a new branch and bound algorithm for solving three machine permutation flow shop problem where the optimization criterion is the minimization of sum of completion times of all the jobs. The permutation flow shop problem ($F||\sum C_i$) belongs to the class of NP-hard problems; finding the optimal solution is thus expected to be highly computational. For each solution our scheme gives an approximation ratio and finds near optimal solutions. Computational results for up to 20 jobs are given for 3 machine flow shop problem when the objective is minimizing the sum of completion times. The thesis also discusses a number of related but easier flow shop problems where polynomial optimization algorithms exist.

.

ACKNOWLEDGEMENTS

I wish to express my deep gratitude to my advisor and chair, Dr. Wolfgang Bein, for his excellent guidance, supervision and tremendous support. His guidance helped me all the time and his invaluable support during my TA work helped me to complete my thesis.

I would like to specially thank Dr. Laxmi P. Gewali, Dr. Ajoy K. Datta and Dr. Zhiyong Wang for their willingness to serve on my committee.

I am grateful to Dr. Ajoy K Datta for his moral support and encouragement. I am indebted to school of computer science, University of Nevada Las Vegas for providing the financial support.

Finally I would like to thank my parents, both elder sisters and my dear friends for their endless support and motivation. They were always willing to help me and encourage me with their best wishes.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION TO SCHEDULING

1.1 Scheduling

A scheduling problem can be described as follows. Given m identical machines M_j $(j=1, 2\ldots, m)$ and n jobs J_i $(i=1, 2,\ldots n)$ with processing times. A schedule is an optimal allocation of jobs to machines over time. The scheduling restrictions are a job cannot be processed by more than one machine at a time and a machine can process at most one job at a time.

Gantt charts are used to graphically represent a schedule. There are two types of Gantt charts, namely machine oriented Gantt charts and job-oriented Gantt charts. In machine oriented Gantt charts X-axis represents the time and Y-axis represents the machines. In job oriented Gantt charts X-axis represents the time and Y-axis represents the jobs. Figure 1.1 and Figure 1.2 represent the machine oriented and job oriented Gantt chart respectively for 3 machine and 4 jobs problem.

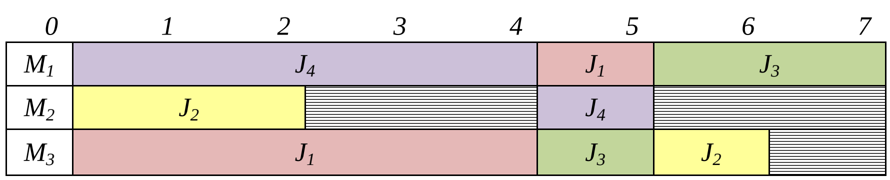


Figure 1.1 Machine Oriented Gantt Chart

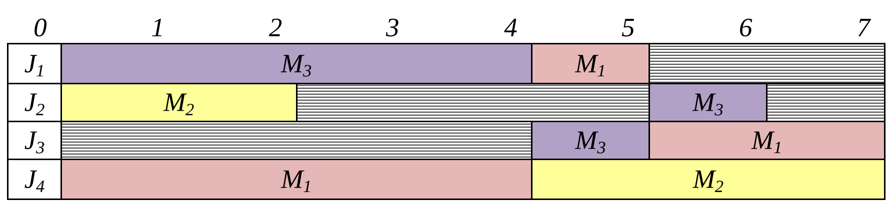


Figure 1.2 Job Oriented Gantt Chart

1.1.1 Notations

According to Peter Brucker[2], the following notations are used to describe a basic scheduling problem.

J_i represents the set of n jobs where $i = \{1, 2\ldots n\}$. M_j represents the set of m machines where

$j = \{1, 2 \ldots m\}$. Each job J_i has k number of operations and are denoted as $O_{i1}, O_{i2}, \ldots, O_{ik}$. Associated with each operation is a processing time denoted by p_{ij}. Completion time of operation of job i on machine j is denoted as c_{ij}. Completion time of job J_i is the time taken by the job to complete all its operations and is denoted by C_i. In addition each job has a weight w_i, deadline d_i and release time r_i. A schedule is said to be feasible if no two operations of a job are processed at the same time and a machine can process at most one job at a time. A schedule is said to be optimal if it minimizes the optimality criteria.

1.2 Classes of Scheduling

Scheduling problems are defined by a three field notation α|β|γ [2] where

α describes machine environment

β describes job characteristics and

γ specifies optimality criteria

1.2.1 Machine Environment (α)

The machine environment is described by the string $\alpha = \alpha_1\alpha_2$ where $\alpha_1 \in \{o, P, Q, R, PMPM, QMPM, G, J, O, F, X\}$ and α_2 specifies number of machines.

Case 1: If $\alpha_1 \in \{o, P, Q, R, PMPM, QMPM\}$ each job J_i consists of a single operation.

$\alpha_1 \in o$ **Single machine**

o represents the empty symbol. When $\alpha_1 = o$, $\alpha = \alpha_2$ and here only single machine is available for processing the jobs.

$\alpha_1 \in P$ **Identical parallel machines**

There are m parallel machines with identical speeds available for processing the jobs. The processing time p_{ij} of job J_i on machine M_j is, $p_{ij} = p_i$.

$\alpha_1 \in Q$ **Uniform parallel machines**

For processing the jobs there are m parallel machines available with each machine having an individual processing speed s_j. The processing time p_{ij} of job J_i on machine M_j is, $p_{ij} = p_i / s_j$.

$\alpha_1 \in R$ **Unrelated parallel machines**

For processing the jobs there are m parallel machines available with each machine having an individual processing speed s_{ij}. The processing time p_{ij} of job J_i on machine M_j is, $p_{ij} = p_i / s_{ij}$.

$\alpha_1 \in$ **PMPM or QMPM**

If α_1 = *PMPM* or *QMPM* then they are multi-purpose machines with identical speeds and uniform speeds respectively.

Case 2: If $\alpha_1 \in \{G, J, O, F, X\}$ then each job J_i is associated with a set of operations $\{O_{i1}, O_{i2},\ldots,O_{ik}\}$ and each operation must be processed on a dedicated machine.

$\alpha_1 \in G$ **General shop**

In general shop there is precedence relation between the operations.

$\alpha_1 \in J$ **Job shop**

Job shop is a special case of general shop. In job shop each job has a predetermined route and the precedence relation between the operations is of the form $O_{i1} \rightarrow O_{i2} \rightarrow \ldots\ldots O_{ik}$. Thus for the job shop problem, for each machine j we need to find a job order.

$\alpha_1 \in F$ **Flow shop**

In flow shop each job J_i consists of m operations $O_{i1},O_{i2},\ldots O_{im}$ and the j^{th} operation of job i has to be processed on machine j for p_{ij} time units. The precedence relation between the operations is, a job can start processing on machine j, only after completing its operation on machine (j-1). Here all the jobs follow the same machine order $M_1 \rightarrow M_2 \rightarrow\ldots\ldots M_m$. Thus for the flow shop problem we need to find the job order for each machine. If all the machines follow the same job order then is called permutation flow shop. For permutation flow shop we use the notation *F –perm*.

$\alpha_1 \in O$ **Open shop**

In open shop each job J_i consists of m operations $O_{i1}, O_{i2},\ldots, O_{im}$ and the j^{th} operation of job i has to be processed on machine j for p_{ij} time units. There are no precedence relations between the operations. Thus in case of open shop we need to find both the job as well as machine orders.

$\alpha_1 \in X$ **Mixed shop**

Mixed job is the combination of job shop and open shop.

Symbol	***Description***
1	*Single machine*
P	*Identical parallel machine*
Q	*Uniform parallel machine*
R	*Unrelated parallel machine*
PMPM	*Multi-purpose machine with identical speeds*
QMPM	*Multi-purpose machine with uniform speeds*
G	*General shop problem*
J	*Job shop*
O	*Open shop*
F	*Flow shop*
X	*Mixed shop*

Table 1.1 Notations for Machine Environment (α)

1.2.2 Job Characteristics (β)

Job characteristics are specified by the set $\beta \in \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6\}$[2].

$\beta_1 \in pmtn$ **Preemption**

Preemption means that the processing of the jobs can be interrupted and can be resumed later even on other machine. If $\beta_1 = pmtn$ then preemption is allowed, otherwise preemptions are not allowed.

$\beta_2 \in prec$ **Precedence constraints**

Job J_j cannot start processing until the job J_i has completed. This constraint on jobs is specified using precedence constraints. Precedence constraints are given by graph $G = (V, A)$ where each vertex corresponds to a job and each arc represents a precedence constraint. Chains, intree, outree, sp-graph gives restricted precedence constraint between the jobs. We set β_2 = chains if each node has atmost one predecessor and one successor. We set β_2 = intree if each node has atmost one successor and β_2 = outree if each node has atmost one predecessor.

According to Peter Brucker[2], A graph $G = (V, A)$ is called a series parallel graph if it consists of a single vertex or if it is formed by the parallel combination of two graphs $G1 = (V1, A1)$ and $G2 = (V2, A2)$ such that $G = (V1 \cup V2, A1 \cup A2)$ or by the series combination of two graphs

$G1 = (V1, A1)$ and $G2 = (V2, A2)$ such that $G = (V1 \cup V2, A1 \cup A2 \cup T1 \times S2)$. Here $T1$ is set of sinks in graph $G1$ and $S2$ is set of sources in graph $G2$. We set β_2 = sp-graph if the given graph is a series parallel graph.

$\beta_3 \in r_i$ **Release dates**

Release dates specifies the time when the first operation of the job J_i is available for processing. If each job is associated with a release time then it is specified by $\beta_3 = r_i$ [3].

$\beta_4 \in p_{ij}$ **Processing times**

If there are restrictions on the processing times of the jobs then we represent it using β_4. If

$\beta_4 = p_{ij} = 1$ then the processing times of all the jobs is 1. If $\beta_4 = p_{ij} = p$ then the processing times of all the jobs is equal to p.

$\beta_5 \in d_i$ **Deadlines**

Deadline is the time by which the job J_i has to complete its execution. If the jobs are subjected to deadline constraint then it is specified by $\beta_5 = d_i$.

$B_6 \in$ {*s*-batch, *p*-batch} **Batch processing**

In batch problems the jobs are grouped together and are scheduled. There are two types of batches namely *s*-batch and *p*-batch. The completion time of jobs in the batch is equal to finishing time of the batch. In *s*-batch the finishing time of the batch is the sum of processing times of all the jobs in the batch and in *p*-batch, the finishing time of the batch is maximum of processing times of jobs.

1.2.3 Optimality Criteria (γ)

According to Peter Brucker[2], the third field refers to optimality criteria. A schedule is said to be optimal if it minimizes the objective function. c_{ij} denotes the completion time of operation of job i on machine j.

C_i denotes completion time of job J_i. Completion time of a job is the time at which the job completes its processing and exits the system. The commonly used objectives are to minimize the makespan or the sum of the completion times of the jobs.

Makespan (Cmax)

Makespan is the maximum of the completion times of all the jobs. It is represented as C_{max}.

$$C_{max} = max\ \{C_i,\ i = 1,2,\ldots,n\}$$

Sum of Completion Time ($\sum C_i$)

Sum of completion time is the summation of the completion times of all the jobs.

$$\sum C_i = \sum_{i=1}^{n} C_i$$

Lateness (L_i)

Lateness is the difference between the completion time of a job and its due date. It is used to determine whether a job is completed before or after its due date. If lateness is positive implies a job is completed after the due date and is called tardiness. If lateness is negative, it is earliness and implies that the job is competed before the due date [3].

$$L_i = C_i - d_i$$

Tardiness (T_i)

Tardiness occurs if the job J_i is completed after its deadline. It is given as,

$$T_i = max\ \{0, C_i - d_i\}$$

Earliness (E_i)

Earliness occurs if the job J_i is completed before its deadline. It is given as,

$$E_i = max\ \{0, d_i - C_i\}$$

Unit Penalty (U_i)

If a job J_i is completed after the deadline, then a penalty of one unit is imposed on the job.

$$U_i = \begin{cases} 0 & C_i \le d_i \\ 1 & otherwise \end{cases}$$

Absolute Deviation (D_i)

$$D_i = |C_i - d_i|$$

Squared Deviation (S_i)

$$S_i = (C_i - d_i)^2$$

1.3 Disjunctive Graph Model

Disjunctive graph depicts all the feasible solutions of the shop problems. The feasible solution set always contains the optimal solution. Therefore disjunctive graph model can be used to find the optimal solution. According to Peter Brucker[2], for a disjunctive graph G (V, C, D)

V **Set of vertices**

V is the set of vertices containing the operations of all jobs. In addition to these vertices, it also contains a source (0) and a sink (*) vertex. Weight of source and sink are zero while the weights of all the other nodes are there corresponding processing times.

C **Set of conjunctive arcs**

C is the conjunctive arc set representing the precedence constraint between the operations. Additionally conjunctive arcs are drawn between source and all operations without a predecessor and between sink and all operations without a successor.

D **Set of disjunctive arcs**

Disjunctive arcs are drawn between pair of operations belonging to the same job which are not connected by conjunctive arcs and between pair of operations which are to be processed on the same machine and which are not connected by conjunctive arcs.

CHAPTER 2

FLOW SHOP SCHEDULING

2.1 Flow Shop Problem

A flow shop problem is defined as follows. There are n jobs J_i (J_1, J_2, J_3,….,J_n) and m machines M_j (M_1,M_2,M_3,….M_m). Each job J_i consists of m operations O_{i1},O_{i2},…O_{im} and the j^{th} operation of job i has to be processed on machine j for p_{ij} time units.

The precedence relation between the operations is, a job can start processing on machine j, only after completing its operation on machine (j-1) [5]. No two operations of a job are processed at the same time and a machine can process at most one job at a time. In flow shop all the jobs follow the same machine order $M_1 \rightarrow M_2 \rightarrow \ldots\ldots M_m$ but the job order for each machine differs. The common objectives are to minimize the makespan or the sum of the completion times of the jobs. Thus for the flow shop problem, for each machine j we need to find a job order. In case of n-job m-machine flow shop problem there exists $(n!)^m$ schedules and finding an optimal schedule in that case is likely hard. Therefore we restrict our attention to permutation schedules.

Example for Flow Shop Problem

Job i	M_1	M_2	M_3
J_1	1	2	3
J_2	2	3	4
J_3	2	3	5

Table 2.1 Example to Illustrate Flow Shop Problem

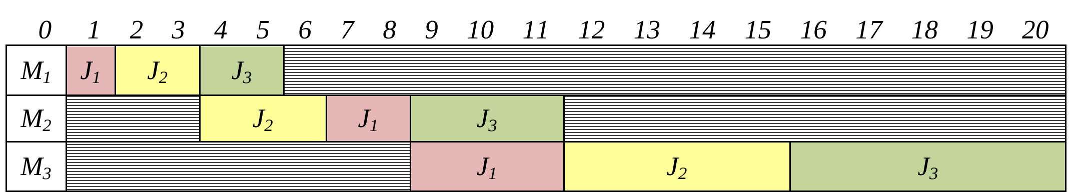


Figure 2.1 Gantt Chart for Flow Shop problem

2.2 Permutation Flow Shop

Permutation flow shop is a special case of flow shop problem with an additional constraint that the job sequence is same on all the machines. With this constraint the number of sequences reduces to *(n!)*.

Example for Permutation Flow Shop Scheduling Problem

Job i	p_{i1}	p_{i2}
J_1	5	2
J_2	1	6

Table 2.2 Example to Illustrate Permutation Flow Shop Problem

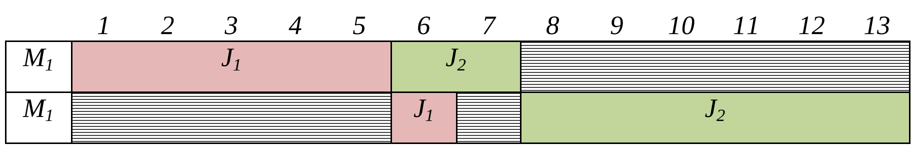


Figure 2.2 Gantt Chart for Permutation Flow Shop Problem

2.2.1 $F2||C_{max}$ and $F2||\sum C_i$

According to Peter Brucker[2], "*For the* $F2||C_{max}$ and $F2||\sum C_i$ *problem there exists an optimal schedule in which both the machines process the jobs in the same order".*

Proof: Assume an optimal schedule with the same order for first k jobs on both machines and $k<n$. Let i be the k^{th} job, and let j be the job immediately after job i on machine 2.

Then we have the optimal schedule as follows:

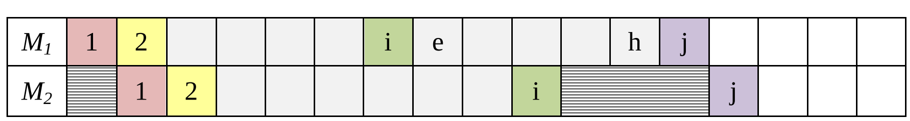


Figure 2.3 Schedule Representing Same Job Order for First *k* Jobs

If we reschedule job j to the position immediately after job i on machine 1 and move all jobs scheduled between job i and job j by p_{j1} time units to the right, (we can do this without increasing the completion time of any job on machine 2) we get another optimal schedule[1]. We can continue this pairwise switching of jobs on the machine 1 until the job order of machine 1 matches with machine 2 [6]. Thus for the $F2||C_{max}$ and $F2||\sum C_i$ problem there exists an optimal schedule in which both the machines process the jobs in the same order.

2.2.2 $Fm||C_{max}$

According to Lemma 6.8 [2], "*For problem $Fm||C_{max}$ optimal schedule exists with the following properties:*

(i) The job sequence on the first two machines is the same.

(ii) The job sequence on the last two machines is the same.

For two or three machines, the optimal solution of the flow shop problem is not better than that of the corresponding permutation flow shop. This is not the case if there are more than three machines".

Proof: The proof of (i) is similar to $F2||C_{max}$.

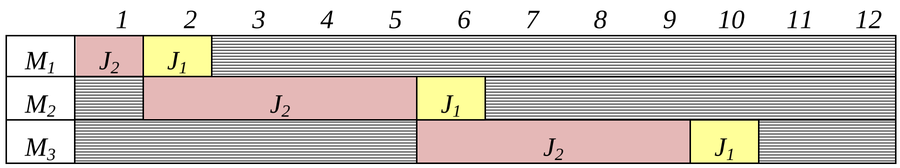
In case of (ii), from [1], if the job order differs on last two machines; reschedule the jobs on machine m so that it matches with the order on machine $m-1$. We continue this pairwise switching of jobs on the machine m until the job order of machine m and machine $m-1$ is identical. Therefore when $m \geq 3$ the number of sequences reduces from $(n!)^m$ to $(n!)$.

But when $m \geq 3$ the above property is not true for the sum of completion time of all jobs, $Fm||\sum C_i$

Example:

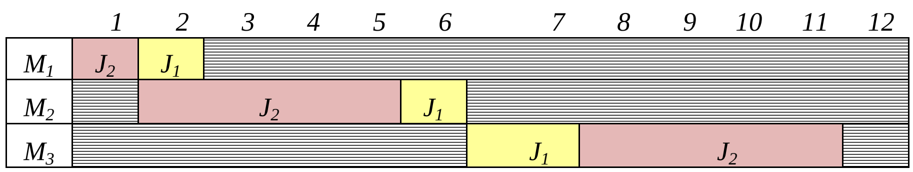
Job i	p_{i1}	p_{i1}	p_{i1}
J_1	4	1	1
J_2	1	4	1

Table 2.3 Example to show same job order for $Fm||\sum C_i$ does not hold for $m \geq 3$



For same job order J_2-J_1 $\sum C_i = C_1+C_2 = 9 + 10 = 19$

Figure 2.4 Schedule with same job order on last 2-machines



For different job order $\sum C_i = C_1+C_2 = 7 + 11 = 18$

Figure 2.5 Schedule with different job order on last 2-machines

From Figure 2.4 and Figure 2.5 we see that, if we follow a different job order on the machine 3, we get another schedule where $\sum C_i = C_1 + C_2 = 7 + 11 = 18$. Therefore the above example makes the point that, when $m \geq 3$ property (ii) does not hold for the total completion time, $Fm||\sum C_i$.

2.3 Johnson's Algorithm for $F2||C_{max}$ Problem

According to Peter Brucker [2], Johnson's algorithm finds the optimal schedule for $F2||C_{max}$ problem. The algorithm uses the same job order on both the machines. It constructs two lists L and R, where list L contains jobs such that $p_{i1} < p_{i2}$ and list R contains jobs such that $p_{i1} > p_{i2}$. The optimal schedule is constructed by concatenating $T = L$ and R [6].

From the list of unscheduled jobs identify the job with the smallest processing time. If the job with smallest processing time involves machine *1*, then concatenate the job at the end of the list L.
If the job with the smallest processing time involves machine *2* concatenate the job at the beginning of the list R. Then delete the job from the list. This process continues on until all jobs have been scheduled. Final schedule is obtained by combining the lists L and R.

Algorithm 2.1 Johnson's Algorithm

1. Let $S = \{1, 2 \dots n\}$ be the list of unscheduled jobs. Let L, R denote two other lists
2. Find the job i with minimum processing time i.e p_{ij}
3. If $j = 1$, concatenate job i at the end of list L
4. Else concatenate job i at the beginning of list R
5. Remove the job i from the list S.
6. If there is an unscheduled job GO TO step 1
7. Else concatenate L and R

Example for $F2||C_{max}$:

To explain Johnson's algorithm the following 5 jobs and 2 machines problem has been used.

Job i	p_{i1}	p_{i2}
J_1	4	5
J_2	1	6
J_3	9	1
J_4	8	1
J_5	5	6

Table 2.4 Example to Demonstrate Johnson's $F2||C_{max}$ Problem

Let $S = \{1, 2 \dots n\}$, $L = \{\ \}$, $R = \{\ \}$

Min p_{ij}	*Machine j*	*List L*	*List R*	*Set of job's S*
p_{21}	$j=1$	{2}	{ }	{1,3,4,5}
P_{32}	$j=2$	{2}	{3}	{1,4,5}
P_{42}	$j=2$	{2}	{4,3}	{1,5}
P_{11}	$j=1$	{2,1}	{4,3}	{5}
P_{51}	$j=1$	{2,1,5}	{4,3}	{ }

Table 2.5 Execution Steps for Johnson's $F2||C_{max}$ problem

Disjunctive Graph for *F2||C_{max}*

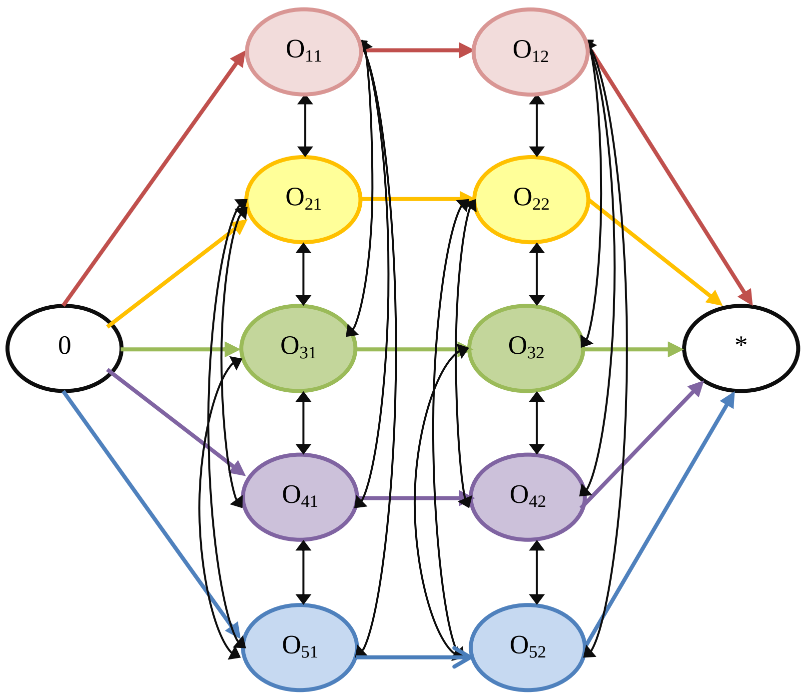
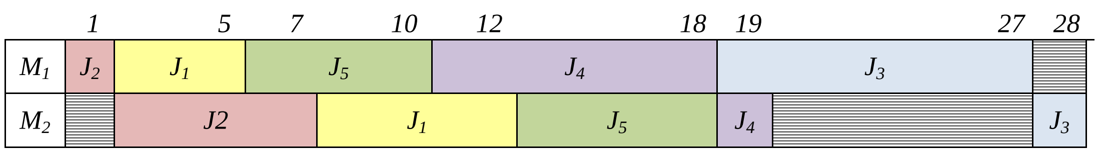


Figure 2.6 Disjunctive Graph for $F2||C_{max}$

Therefore using Johnson's algorithm the optimal sequence is, T = {J_2, J_1, J_5, J_4, J_3}



T=J_2, J_1, J_5, J_4, J_3

C_{max}=28

Figure 2.7 Optimal Schedule for Johnson's $F2||C_{max}$ Problem

Disjunctive Graph of Optimal Solution for $F2||C_{max}$

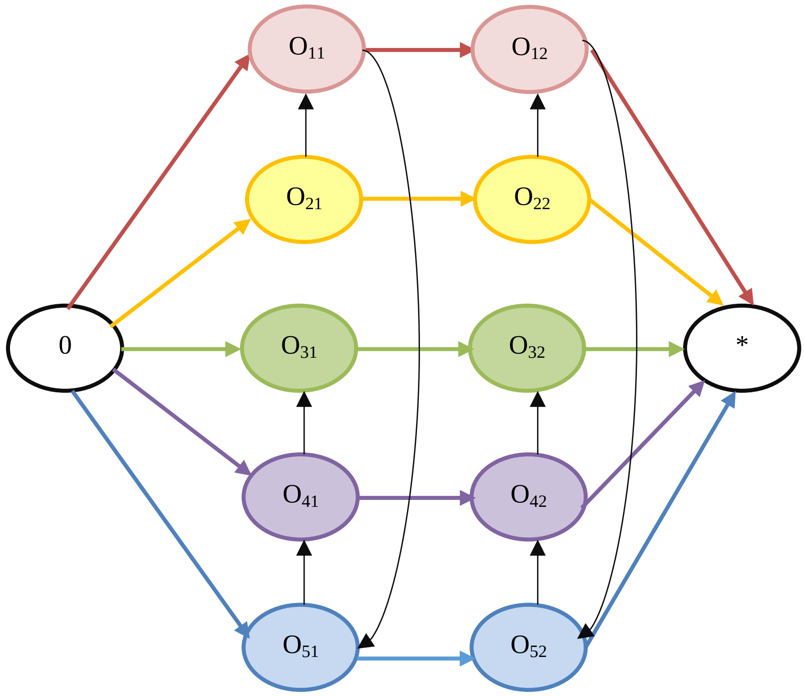


Figure 2.8 Disjunctive Graph for Optimal Solution for $F2||C_{max}$

Lemma 2.3.1

According to Lemma 6.9 [2], to solve $F2||C_{max}$ problem Johnson proposed a rule called Johnson's rule. If T is the list constructed by the algorithm then, for any two jobs J_i and J_j if

min $\{a_i, b_j\}$ < min$\{a_j, b_i\}$ then job J_i is scheduled earlier than job J_j in the list T.

Proof:

Case 1:

If a_i is min, $a_i <$ min$\{a_j,b_i\}$ then $a_i < b_i$ implies Job J_i belongs to list L. If job J_j is added to list R we are done. Otherwise if Job J_j goes into L, it appears after J_i because $a_i < a_j$.

Case 2:

If b_j is min, $b_j <$min$\{a_j,b_i\}$ then $b_j < b_i$ implies Job J_j belongs to list R. If job J_i is added to list L we are done. Otherwise if Job J_i goes into R, it appears before J_j because $b_i > b_j$.

Lemma 2.3.2

According to Lemma 6.10 [2], *Consider a schedule in which job j is scheduled immediately after job i, then*

$$min\{ p_{j1}, p_{i2}\} \leq min\{p_{i1}, p_{j2}\}$$

implies that i and j can be swapped without increasing the C_{max} *value.*

Proof:

If j is scheduled immediately after i, then we have three possible cases as shown in figure. Let w_{ij} be the length of the time period from the start of job i to the finishing time of job j. Then,

Case 1:

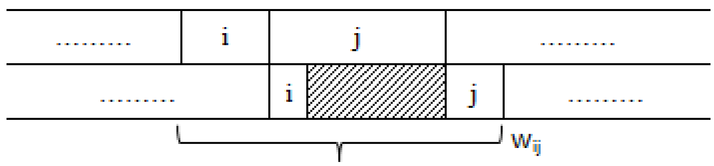


Figure 2.9 Case (a) if j is scheduled immediately after i

For case 1, $w_{ij} = max\{ p_{i1} + p_{j1} + p_{j2}\}$

Case 2:

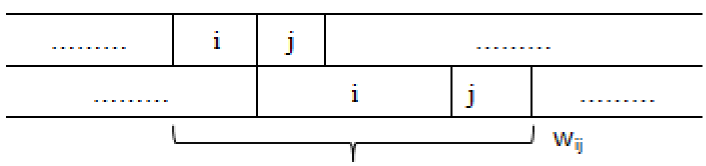


Figure 2.10 Case (b) if j is scheduled immediately after i

For case 2, $w_{ij} = max\{ p_{i1} + p_{i2} + p_{j2}\}$

Case 3: $w_{ij} = max\{x + p_{i2} + p_{j2}\}$

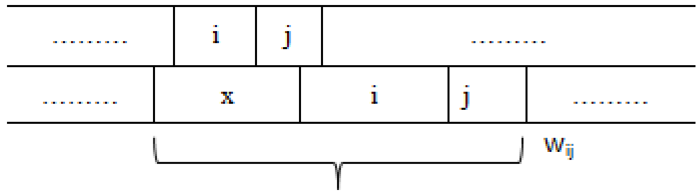


Figure 2.11 Case(c) if j is scheduled immediately after i

For case 3, $w_{ij} = max\{x + p_{i2} + p_{j2}\}$

From case 1, case 2 and case 3, the possible w_{ij} is,

$$w_{ij} = max\{ p_{i1} + p_{j1} + p_{j2}, p_{i1} + p_{i2} + p_{j2}, x + p_{i2} + p_{j2}\}$$

$$= max\ \{p_{j1} + p_{i2} + max\ \{p_{i1}, p_{j2}\}, x + p_{i2} + p_{j2}\}$$

Similarly $w_{ji} = max\ \{p_{i1} + p_{j2} + max\ \{p_{j1}, p_{i2}\}, x + p_{j2} + p_{i2}\}$, if i is scheduled immediately after j

According to Lemma 6.10 [2], we see that

$min\{ p_{j1}, p_{i2}\} \leq min\{p_{i1}, p_{j2}\}$ can be written as,

$$max\{-p_{i1},-p_{j2}\} \leq max\{-p_{j1},-p_{i2}\}$$

Adding p_{i1}, p_{i2}, p_{j1}, p_{j2} to both sides of the above inequality we get,

$$p_{i1} + p_{j2} + max\{-p_{i1},-p_{j2}\} + p_{i2} + p_{j1} \leq p_{j1} + p_{i2} + max\{-p_{j1},-p_{i2}\} + p_{i1} + p_{j2}$$

$$= max\{ p_{i1}, p_{j2}\} + p_{i2} + p_{j1} \leq max\{ p_{j1}, p_{i2}\} + p_{i1} + p_{j2}$$

$$= w_{ji} \leq w_{ij}$$

As, $w_{ji} \leq w_{ij}$ implies that we can swap i and j without increasing C_{max} value.

Theorem 2.3.3

According to Theorem 6.11 [2], the list *L: L(1), L(2)…,L(n)* constructed by the Johnson's algorithm for $F2||C_{max}$ problem is optimal

Proof:

To prove the above theorem we use the Lemma 2.3.1 and Lemma 2.3.2

Assume that the list L constructed by the Johnson's algorithm was not optimal. Let us consider then, an optimal solution S such that, S matches with L as much as possible in the following way [2]:

$$L(v) = S(v) \text{ for } v = 1, 2, 3, \ldots, (s-1).$$

Let $L(s) = i$ and $S(s) = j$. In S, i is not an immediate successor of j. Let the job k be schedule between job j and job i. Thus we have,

$$L: L(1), L(2), \ldots, L(s-1), i, k, j \text{ and } S: S(1), S(2), \ldots, S(s-1), j, k, i$$

Here all we have to show is that we can swap k and i without increasing C_{max} value of S. We need to continue swapping until S matches with L, then we can say that the list L constructed by Johnson's algorithm is optimal.

Since k is not before i in L, using the Lemma 2.3.1 we say that,

$$\min\{ p_{k1}, p_{i2}\} \geq \min\{p_{i1}, p_{k2}\}$$

Now applying the lemma 2.3.2 to S, we can swap k and i without increasing the C_{max} value. We continue swapping in S until, S matches with L. Thus the list L constructed by Johnson's algorithm is optimal.

2.4 Johnson's Algorithm for $F2||\sum C_i$ Problem

Johnson's algorithm gives arbitrarily bad solution for $F2||\sum C_i$ problem. From [5], for example let us consider a two machine flow shop problem with n jobs. The value ϵ is considered very small and value k is very large.

Job i	p_{i1}	p_{i2}
J_1	ϵ	ϵ
J_2	ϵ	ϵ
J_3	ϵ	ϵ
$\vdots$		
J_n	$\epsilon/2$	k

Table 2.6 Example to Show Johnson's Algorithm is bad for $F2||\sum C_i$ Problem

Johnson's algorithm schedules the n^{th} job first, followed by jobs $J_1, J_2 \ldots, Jn$.

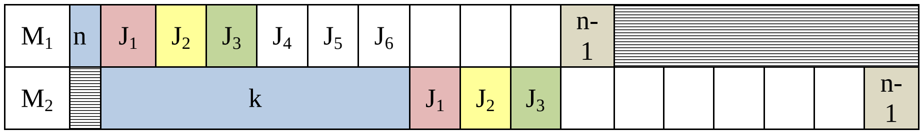


Figure 2.12 Schedule for F2||∑C_i based on Johnsons Algorithm

$\sum C_i = C_1+C_2+C_3+\ldots\ldots C_{n-1}+C_n$

$= (\epsilon/2+k)+(\epsilon/2+k+\epsilon)+(\epsilon/2+k+2\epsilon)+\ldots\ldots\ldots\ldots\ldots+(\epsilon/2+k+(n-1)\epsilon)$

$= n(\epsilon/2)+nk+(n(n-1)\epsilon)/2$

$= nk+\epsilon/2(n+n(n-1))$

Therefore the solution constructed by this algorithm is arbitrarily bad as *n* grows.

The optimal solution for $F2||\sum C_i$ problem would schedule the n^{th} job last

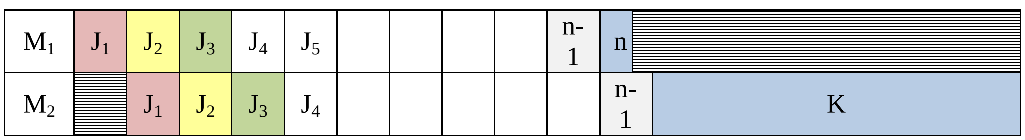


Figure 2.13 Gantt Chart for Optimal Schedule F2||∑C_i

$\sum C_i = C_1+C_2+C_3+\ldots\ldots C_{n-1}+C_n$

$= (\epsilon+\epsilon)+(\epsilon+\epsilon+\epsilon)+\ldots\ldots\ldots\ldots\ldots\ldots n\epsilon+(n\epsilon+\epsilon/2+k)$

$= (n(n+3)-1)(\epsilon/2)+k$

CHAPTER 3

BRANCH AND BOUND ALGORITHM FOR PERMUTATION FLOW SHOP

From this chapter we consider only permutation flow shop *Fm-perm*. This chapter is organized as follows: In the next section we define the problem statement. In section 3.2 we present the branch and bound algorithm; then in section 3.3 we derive the three possible lower bounds. In section 3.4 we introduce the notations used for branch and bound algorithm. In section 3.5 we illustrate the branch and bound algorithm with an example. In section 3.6 we generalize the branch and bound approach when $m \geq 3$.

3.1 Problem Statement

Given a three machine permutation flow shop scheduling problem *F3-perm*, and the objective is to find a permutation schedule that minimizes the sum of the completion time $\sum C_i$ of all the jobs. The three machine flow shop problem *F3* is defined as follows:

There are n Jobs J_i ($J_1, J_2, J_3 \ldots J_n$) and 3 machines M_1, M_2, M_3. Each job must be processed on the three machines, first on machine M_1, then on M_2 and then on M_3. The processing times of job i on machine j is denoted as p_{ij}. The completion time of job i on machine j is denoted as c_{ij}. The completion time of job J_i ; C_i, is the time when its last operation has completed on the last machine
$C_i = C_{i3}$.

The problem $F3\text{-}perm||\sum C_i$ belongs to the class of NP-hard and thus finding the optimal solution is likely hard. We construct a new branch and bound algorithm for solving it. Branch and bound intelligently enumerates permutations of the schedule. This algorithm is obviously an exponential algorithm, but it performs much better in practice than the complete enumeration.

3.2 Using Branch and Bound Algorithm

Given a Problem P and all feasible solutions of the problem P are defined by the set S, which is called the solution space for that problem. The problem P is divided into sub problems S_i such that
$S_i \subseteq S$ [2]. These sub problems are again divided into smaller sub problems. Thus branching is a recursive process and entire solution space is organized as a tree.

The basic components needed for branch and bound algorithm are:

Branching Strategy: Branching strategy divides the solution space *S* in to smaller and smaller sub problems S_i (i =1, 2, 3…r) such that $S = \bigcup_{i=1}^{r} S_i$.

Lower Bounding: Then we apply an algorithm to calculate the lower bound for each sub problem generated in the branching tree.

Pruning Strategy: If the lower bound of the sub problem is greater than or equal to upper bound, then this sub problem cannot yield a better solution and we stop branching from the corresponding node and all other nodes that emerge from it in the branching tree.

The principle of branch and bound algorithm is to make an implicit search through all feasible solutions. Branch and bound tree starts with an initial root node where no jobs have been scheduled. Then we try to branch in this tree by trying to fix each of the jobs as the first job in the sequence. The possible branches are *n* since there are *n* jobs. Each of these *n* nodes emanate in to *(n-1)* branches as there are *(n-1)* possible jobs that can occupy the second place in the sequence. Thus this is a recursive process.

In branch and bound algorithm, each node represents a partial schedule where *k* jobs are scheduled in fixed order. Branching from a node consists of taking each of the unallocated jobs in turn and placing it next to the partial schedule. Each of these new partial schedules is then represented by a new node. The lower bound values for each node are then calculated.

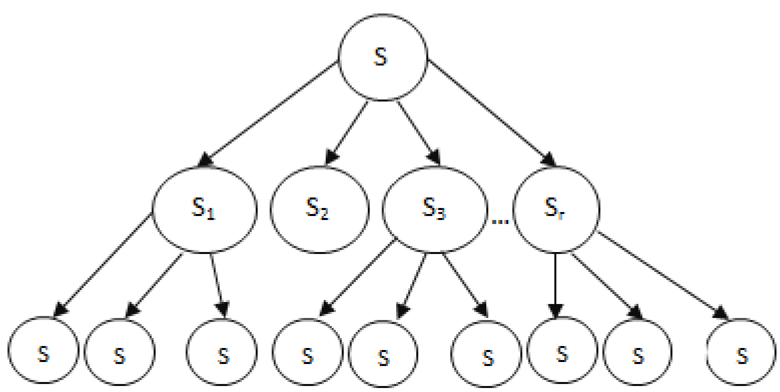


Figure 3.1 General Branch and Bound Search Tree

3.3 Lower Bound Calculation:

Each node in the search tree contains a set of jobs k that are already scheduled and set of jobs that need to be scheduled. Let J_i (J_1, J_2, $J_3, \ldots, J_n$) represents the set of jobs. Suppose we are at a node at which the jobs in the set $M \subseteq \{1, 2 \ldots k\}$ are already scheduled in that order; $|M| = r$. Let

$U \subseteq \{r+1, r+2, \ldots, n\}$ represents the set of unscheduled jobs. Sum of the completion times for this schedule can be divided into,

$$S = \sum_{i \in M} C_i + \sum_{i \notin M} C_i \tag{3.1}$$

Computing the second sum is very difficult, therefore we estimate its lower bound based on the following assumptions:

3.3.1 Calculation of LB_1:

1. Every job $i \notin M$ starts processing on machine 1 without any delay time. That is, after the first job finishes its processing on machine 1, the following job starts immediately without any waiting time.

$$LB_1 = \sum_{k=r+1}^{n} [\sum_{i \in M} p_{i1} + (n-k+1) p_{i_k 1} + p_{i_k 2} + p_{i_k 3}]$$

Consider the jobs $r+1$, $r+2 \ldots n$ completes its processing without any delay on machine 1.

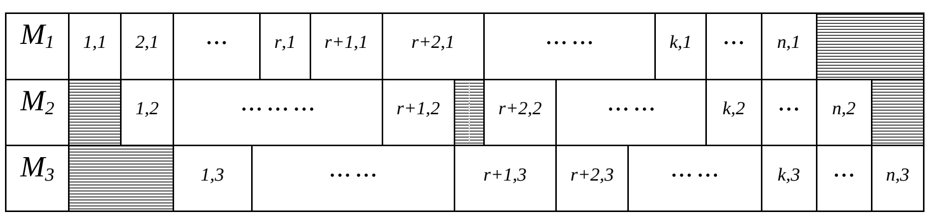


Figure 3.2 Calculation of LB_1

$$LB_1 = C_{r+1} + C_{r+2} + C_{r+3} + \ldots\ldots\ldots\ldots + C_n$$

$$C_{r+1} = \sum_{i \in M} p_{i1} + p_{i_{r+1}1} + p_{i_{r+1}2} + p_{i_{r+1}3}$$

$$C_{r+2} = \sum_{i \in M} p_{i1} + p_{i_{r+1}1} + p_{i_{r+2}1} + p_{i_{r+2}2} + p_{i_{r+2}3}$$

$$\vdots$$

$$C_k = \sum_{i \in M} p_{i1} + p_{i_{r+1}1} + p_{i_{r+2}1} + \ldots\ldots + p_{i_k 1} + p_{i_k 2} + p_{i_k 3}$$

$$\vdots$$

$$C_n = \sum_{i \in M} p_{i1} + p_{i_{r+1}1} + p_{i_{r+2}1} + \ldots\ldots + p_{i_k 1} + \ldots\ldots + p_{i_n 1} + p_{i_n 2} + p_{i_n 3}$$

$$\text{Therefore, } LB_1 = \sum_{k=r+1}^{n} [\sum_{i \in M} p_{i1} + (n-k+1) p_{i_k 1} + p_{i_k 2} + p_{i_k 3}]$$

For LB_1 schedule the jobs in U in increasing order of p_{i1} values.

3.3.2 Calculation of LB_2 :

2. Every job $i \notin M$ starts processing on machine 2 without any delay time. That is, after the first job finishes its processing on machine 2, the following job starts immediately without any waiting time. The expression $\max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\}$ is a lower bound on the start of first job $i \notin M$ on machine 2.

$$LB_2 = \sum_{k=r+1}^{n} [\max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + (n-k+1) p_{i_k 2} + p_{i_k 3}]$$

Consider the jobs $r+1, r+2 \ldots n$ completes its processing without any delay on machine 2.

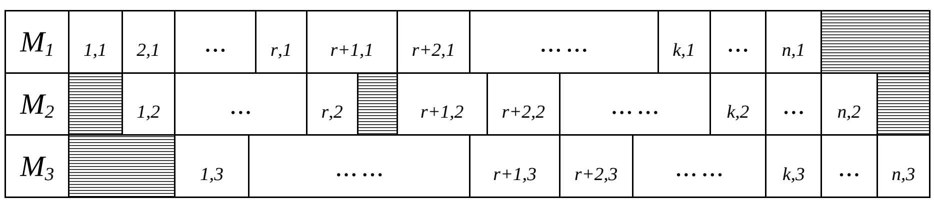


Figure 3.2 Calculation of LB_2

$$LB_2 = C_{r+1} + C_{r+2} + C_{r+3} + \ldots\ldots\ldots\ldots + C_n$$

$$C_{r+1} = \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + p_{i_{r+1}2} + p_{i_{r+1}3}$$

$$C_{r+2} = \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + p_{i_{r+1}2} + p_{i_{r+2}2} + p_{i_{r+2}3}$$

$$\vdots$$

$$C_k = \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + p_{i_{r+1}2} + p_{i_{r+2}2} + \ldots\ldots + p_{i_k 2} + p_{i_k 3}$$

$$\vdots$$

$$C_n = \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + p_{i_{r+1}2} + p_{i_{r+2}2} + \ldots\ldots. + p_{i_k 2} + \ldots. + p_{i_n 2} + p_{i_k 3}$$

Therefore, $LB_2 = \sum_{k=r+1}^{n} [\max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + (n-k+1) p_{i_k 2} + p_{i_k 3}]$

For LB_2 schedule the jobs in U in increasing order of p_{i2} values.

3.3.3 Calculation of LB_3 :

3. Every job $i \notin M$ starts processing on machine 3 without any waiting time. That is, after the first job finishes its processing on machine 3, the following job starts immediately without any waiting time. The expression $\max\{C_{i_r 3}, \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + \min_{i \notin M} p_{i2}\}$ is a lower bound on the start of first job $i \notin M$ on machine 3.

$$LB_3 = \sum_{k=r+1}^{n} [\max\{C_{i_r 3}, \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + \min_{i \notin M} p_{i2}\} + (n-k+1) p_{i_k 3}]$$

Consider the jobs *r+1, r+2…n* completes its processing without any delay on machine 3.

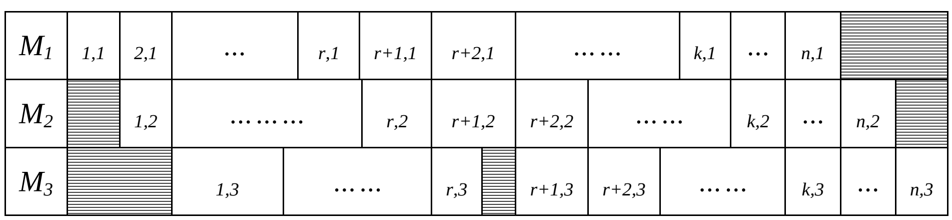


Figure 3.3 Calculation of LB_3

$$LB_3 = C_{r+1} + C_{r+2} + C_{r+3} + \ldots\ldots\ldots\ldots. + C_n$$

$$C_{r+1} = \max\{C_{i_r 3}, \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + \min_{i \notin M} p_{i2}\} + p_{i_{r+1}3}$$

$$C_{r+2} = \max\{C_{i_r 3}, \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + \min_{i \notin M} p_{i2}\} + p_{i_{r+1}3} + p_{i_{r+2}3}$$

$$\vdots$$

$$C_k = \max\{C_{i_r 3}, \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + \min_{i \notin M} p_{i2}\} + p_{i_{r+1}3} + p_{i_{r+2}3} + \ldots.. + p_{i_k 3}$$

$$\vdots$$

$$C_n = \max\{C_{i_r 3}, \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + \min_{i \notin M} p_{i2}\} + p_{i_{r+1}3} + p_{i_{r+2}3} + \ldots.. + p_{i_k 3} + \ldots. + p_{i_n 3}$$

Therefore, $LB_3 = \sum_{k=r+1}^{n} [\max\{C_{i_r 3}, \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + \min_{i \notin M} p_{i2}\} + (n-k+1) p_{i_k 3}]$

For LB_3 schedule the jobs in U in increasing order of p_{i3} values.

Therefore the lower bound is max (LB_1, LB_2, LB_3)

From (3.1), we obtain,

$$S = \sum_{i \in M} C_i + \sum_{i \notin M} C_i$$

$$S = \sum_{i \in M} C_i + \max(LB_1, LB_2, LB_3) \text{ is the cost of the schedule.}$$

3.4 Parameters of the Algorithm

Input

The input to the algorithm is given in a file, where the first parameter indicates the number of jobs n, second parameter indicates the number of machines m. From the third parameter the processing times of jobs follows. The number in row i and column j is the processing time of job i on machine j.

Notations

Following notations are used to implement the algorithm

Notations	
N	*Number of jobs*
M	*Number of machines*
$Job\_arr = \{J_1, J_2, J_3 \ldots J_n\}$	*Set of jobs*
$M \subseteq \{1, 2 \ldots r\}$	*Ordered set of scheduled jobs*
$U \subseteq \{r+1, r+2, \ldots n\}$	*Set of unscheduled jobs*
$p_{ij} \geq 0$	*Processing time of job i on machine j*
c_{ij}	*Completion time of operation of job of i on machine j*
C_i	*Completion time of job* J_i
S	*Sum of the completion time of the schedule*
LB_1	*Lower bound based on machine 1*
LB_2	*Lower Bound based on machine 2*
LB_3	*Lower bound based on machine 3*

Table 3.4 Basic Notations for Branch and Bound Algorithm

Algorithm 3.1 Branch and Bound

1. Initialize *Job_arr* = {J_1, J_2, J_3...J_n}
2. Calculate *initial upperbound* = sum of completion times of initial feasible schedule

   cb_order = initial feasible schedule
3. *sorting_Jobs()*
4. *generate_node(fixed_Jobarr, level)*
   a. IF *level* = *n* (i.e. leaf) then current solution = completion time of the schedule.

      If current solution < upper bound, update upper bound
   b. ELSE
      i. CALCULATE the *lowerBound*
      ii. IF *lowerbound* >= *upperbound* THEN prune the node

          ELSE

              CALL *generate_node(fixed_Jobarr, level+1)*

          END IF

      END IF
5. Stop

3.5 The Algorithm Illustration

To evaluate the branch and bound algorithm the following 5 jobs and 3 machines problem has been used.

Job i	p_{i1}	p_{i2}	p_{i3}
1	4	1	1
2	2	3	2
3	6	5	1
4	5	1	3

Table 3.5 Example to Demonstrate Branch and Bound

In the above table, each row represents the job *i* and each column represents the machine *j*. The processing time of an operation of job *i* on machine *j* is mentioned in each cell and is denoted as p_{ij}. Our objective is to obtain a permutation schedule that minimizes the sum of completion times of all the jobs.

Step 1: Find initial feasible schedule by arranging the jobs in the increasing order of their sum of processing times. The initial feasible schedule is [1, 2, 4, 3]

Job i	*Sum of* p_{ij}
1	4+1+1 = 6
2	2+3+2 = 7
3	6+5+1 = 12
4	5+1+3 = 9

Table 3.6 Calculating Initial Feasible Schedule

Step 2: Calculate initial upper bound which is sum of completion times of initial feasible schedule.

For order [1, 2, 4, 3] upper bound (UB) = $\sum_{i=1}^{n} C_i$ = 6+11+15+23 = 55

Job i	c_{i1}	c_{i2}	c_{i3}
1	*4*	*5*	*6*
2	*6*	*9*	*11*
4	*11*	*12*	*15*
3	*17*	*22*	*23*

Table 3.6 Calculating Initial Upper Bound

Step 3: We now compute the lower bound for each node in the tree. In tree each node represents a partial sequence S_k where jobs in the first *k* positions are fixed. $C_1(k)$, $C_2(k)$, $C_3(k)$, be the completion times on machine 1, machine 2, machine 3 respectively for the partial sequence.

Calculating Lower Bound for Partial Sequence [1 * * *]

Set of scheduled jobs $M = \{1\}$ and $|M| = r = 1$

Set of unscheduled jobs $U = \{2, 4, 3\}$.

Cost of the schedule $S = \sum_{i \in M} C_i + \max$ (LB_1, LB_2, LB_3)

Job i	C_1	C_2	C_3
1	*4*	*5*	*6*

Table 3.8 Calculation of Completion Times for Partial Sequence [1 * * *]

For LB_1 from the list of unscheduled jobs U, schedule the jobs in the increasing order of their processing times on machine 1. Sequence of jobs with increasing p_{i1} values is [2, 4, 3]. Let i_k, k = 1, 2,…,n be the index of these jobs.

	$\sum_{i \in M} p_{i1}$	$(n-k+1)p_{i_k 1}$	$p_{i_k 2}$	$p_{i_k 3}$	$\sum_{i \in M} p_{i1} + (n-k+1)p_{i_k 1} + p_{i_k 2} + p_{i_k 3}$
k=2	*4*	$3.p_{21} = 6$	$p_{21} = 3$	$p_{23} = 2$	*15*
k=3	*4*	$2.p_{41} = 10$	$p_{41} = 1$	$p_{43} = 3$	*18*
k=4	*4*	$1.p_{31} = 6$	$P_{31} = 5$	$p_{33} = 1$	*16*

Table 3.9 Calculation of LB_1 for Partial sequence [1 * * *]

$LB_1 = \sum_{k=r+1}^{n} [\sum_{i \in M} p_{i1} + (n-k+1)p_{i_k 1} + p_{i_k 2} + p_{i_k 3}]$ = 15+18+16 = 49.

For LB_2 from the list of unscheduled jobs U, schedule the jobs in the increasing order of their processing times on machine 2. Sequence of jobs with increasing p_{i2} values is [4, 2, 3]. The expression, $\max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} = max\ \{5, 4 + p_{21}\} = max\ \{5, 4 + 2\} = 6$

	$(n-k+1)p_{i_k 2}$	$p_{i_k 3}$	$\max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + (n-k+1)p_{i_k 2} + p_{i_k 3}$
k=2	$3.p_{42} = 3$	$P_{43} = 3$	*12*
k=3	$2.p_{22} = 6$	$P_{23} = 2$	*14*
k=4	$1.p_{32} = 5$	$p_{33} = 1$	*12*

Table 3.10 Calculation of LB_2 for Partial sequence [1 * * *]

$$LB_2 = \sum_{k=r+1}^{n} [\max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + (n-k+1) p_{i_k 2} + p_{i_k 3}] = 12+14+12 = 38$$

For LB_3 from the list of unscheduled jobs U, schedule the jobs in the increasing order of processing times on machine 3. Sequence of jobs with increasing p_{i3} values is [3, 2, 4]. The expression,

$$\max\{C_{i_r 3},\ \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + \min_{i \notin M} p_{i2}\} = \max\{6, \max\{5, 4 + p_{21}\} + p_{42}\}$$

$$= \max\{6, \max\{5, 4 + 2\} + 1\} = \max\{6, 6+1\} = 7$$

	$(n-k+1) p_{i_k 3}$	$\max\{C_{i_r 3},\ \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + \min_{i \notin M} p_{i2}\} + (n-k+1) p_{i_k 3}$
k=2	$3.p_{33} = 3$	*10*
k=3	$2.p_{23} = 4$	*11*
k=4	$1.p_{43} = 3$	*10*

Table 3.11 Calculation of LB_3 for Partial sequence [1 * * *]

$$LB_3 = \sum_{k=r+1}^{n} [\max\{C_{i_r 3},\ \max\{C_{i_r 2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + \min_{i \notin M} p_{i2}\} + (n-k+1) p_{i_k 3}] = 10+11+10 = 31$$

$$\text{LB}(1 * * *) = \sum_{i \in M} C_i + \max(LB_1, LB_2, LB_3) = C_1 + \max(49, 38, 31) = 6 + 49 = 55$$

Since lower bound of partial sequence (1 * * *) = 55 ≥ upper bound, prune the node [1***] and all the branches that emerge from it.

Calculating Lower Bound for Partial Sequence [2 * * *]

Similarly we calculate the lower bound for partial sequence (2 * * *)

Set of scheduled jobs $M = \{2\}$ and $|M| = r = 1$

Set of unscheduled jobs $U = \{1, 4, 3\}$.

Job i	C_1	C_2	C_3
2	*2*	*5*	*7*

Table 3.12 Calculation of Completion Times for Partial sequence [2 * * *]

For LB_1 from the unscheduled jobs U, schedule the jobs in the increasing order of their processing times on machine 1. Sequence of jobs with increasing p_{i1} values is [1, 4, 3].

	$\sum_{i\in M} p_{i1}$	$(n-k+1)p_{i_k 1}$	$p_{i_k 2}$	$p_{i_k 3}$	$\sum_{i\in M} p_{i1} + (n-k+1)p_{i_k 1} + p_{i_k 2} + p_{i_k 3}$
k=2	*2*	$3.p_{11} = 12$	$P_{11} = 1$	$p_{23} = 1$	*16*
k=3	*2*	$2.p_{41} = 10$	$p_{41} = 1$	$p_{43} = 3$	*16*
k=4	*2*	$1.p_{31} = 6$	$P_{31} = 5$	$p_{33} = 1$	*14*

Table 3.13 Calculation of LB_1 for Partial sequence [2 * * *]

$LB_1 = \sum_{k=r+1}^{n} [\sum_{i\in M} p_{i1} + (n-k+1)p_{i_k 1} + p_{i_k 2} + p_{i_k 3}]$ = 16+16+14 = 46.

For LB_2 from the list of unscheduled jobs U, schedule the jobs in the increasing order of their processing times on machine 2. Sequence of jobs with increasing p_{i2} values is [1, 4, 3]. The expression, $\max\{C_{i_r 2}, \sum_{i\in M} p_{i1} + \min_{i\notin M} p_{i1}\}$ = max {5, 2 + p_{11}} = max {5, 2 + 4} =6

	$(n-k+1)p_{i_k 2}$	$p_{i_k 3}$	$\max\{C_{i_r 2}, \sum_{i\in M} p_{i1} + \min_{i\notin M} p_{i1}\} + (n-k+1)p_{i_k 2} + p_{i_k 3}$
k=2	$3.p_{12} = 3$	$P_{43} = 1$	*10*
k=3	$2.p_{42} = 2$	$P_{23} = 3$	*11*
k=4	$1.p_{32} = 5$	$p_{33} = 1$	*12*

Table 3.14 Calculation of LB_2 for Partial sequence [2 * * *]

$LB_2 = \sum_{k=r+1}^{n} [\max\{C_{i_r 2}, \sum_{i\in M} p_{i1} + \min_{i\notin M} p_{i1}\} + (n-k+1)p_{i_k 2} + p_{i_k 3}]$ = 10+11+12 = 33

For LB_3 from the list of unscheduled jobs U, schedule the jobs in the increasing order of processing times on machine 3. Sequence of jobs with increasing p_{i3} values is [1, 3, 4]. The expression,

$\max\{C_{i_r 3}, \max\{C_{i_r 2}, \sum_{i\in M} p_{i1} + \min_{i\notin M} p_{i1}\} + \min_{i\notin M} p_{i2}\}$ = max {7, max {5, 2 + p_{11}} + p_{12}}

= max {7, max {5, 2 + 4} + 1} = max {7, 6+1} = 7

	$(n-k+1)p_{i_k3}$	$\max\{C_{i_r3},\ \max\{C_{i_r2}, \sum_{i\in M} p_{i1} + \min_{i\notin M} p_{i1}\} + \min_{i\notin M} p_{i2}\} + (n-k+1)p_{i_k3}$
k=2	$3.p_{13} = 3$	*10*
k=3	$2.p_{33} = 2$	*9*
k=4	$1.p_{43} = 3$	*10*

Table 3.15 Calculation of LB_3 for Partial sequence [2 * * *]

$$LB_3 = \sum_{k=r+1}^{n}[\ \max\{C_{i_r3},\ \max\{C_{i_r2}, \sum_{i\in M} p_{i1} + \min_{i\notin M} p_{i1}\} + \min_{i\notin M} p_{i2}\} + (n-k+1)p_{i_k3}] = 10+9+10 = 29$$

$$\text{LB}(2 * * *) = \sum_{i\in M} C_i + \max(LB_1, LB_2, LB_3) = C_2 + \max(46, 33, 29) = 7 + 46 = 53$$

Since lower bound of partial sequence (2 * * *) = 53< upper bound, we branch to lower level nodes from partial sequence (2 * * *). Branching from a node consists of taking each of the unallocated jobs in turn and placing it next to the partial schedule. Each of these new partial schedules is then represented by a new node.

Calculating Lower Bound for Partial Sequence [2 1 * *]

Lower bound for the partial sequence [2 1 * *] is calculated as follows:

Set of scheduled jobs M = {2, 1} and $|M| = r = 2$

Set of unscheduled jobs U = {4, 3}.

Job i	C_1	C_2	C_3
2	*2*	*5*	*7*
3	*6*	*7*	*8*

Table 3.16 Calculation of Completion Times for Partial sequence [2 1 * *]

For LB_1 from the list of unscheduled jobs U, schedule the jobs in the increasing order of their processing times on machine 1. Sequence of jobs with increasing p_{i1} values is [4, 3].

	$\sum_{i \in M} p_{i1}$	$(n-k+1)p_{i_k1}$	p_{i_k2}	p_{i_k3}	$\sum_{i \in M} p_{i1} + (n-k+1)p_{i_k1} + p_{i_k2} + p_{i_k3}$
k=3	*6*	*2.p_{41} = 10*	*p_{41} = 1*	*p_{43} = 3*	*20*
k=4	*6*	*1.p_{31} = 6*	*P_{31} = 5*	*p_{33} = 1*	*18*

Table 3.17 Calculation of LB_1 for Partial sequence [2 1 * *]

$$LB_1 = \sum_{k=r+1}^{n} [\sum_{i \in M} p_{i1} + (n-k+1)p_{i_k1} + p_{i_k2} + p_{i_k3}] = 20+18 = 38$$

For LB_2 from the list of unscheduled jobs U, schedule the jobs in the increasing order of their processing times on machine 2. Sequence of jobs with increasing p_{i2} values is [4, 3]. The expression,

$$\max\{C_{i_r2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} = \max\{7, 6 + p_{41}\} = \max\{5, 6 + 5\} = 11$$

	$(n-k+1)p_{i_k2}$	p_{i_k3}	$\max\{C_{i_r2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + (n-k+1)p_{i_k2} + p_{i_k3}$
k=3	*2.p_{42} = 2*	*P_{43} = 3*	*16*
k=4	*1.p_{32} = 5*	*p_{33} = 1*	*17*

Table 3.18 Calculation of LB_2 for Partial sequence [2 1 * *]

$$LB_2 = \sum_{k=r+1}^{n} [\max\{C_{i_r2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + (n-k+1)p_{i_k2} + p_{i_k3}] = 16+17 = 33$$

For LB_3 from the list of unscheduled jobs U, schedule the jobs in the increasing order of processing times on machine 3. Sequence of jobs with increasing p_{i3} values is [3, 4]. The expression,

$$\max\{C_{i_r3}, \max\{C_{i_r2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + \min_{i \notin M} p_{i2}\} = \max\{8, \max\{7, 6 + p_{41}\} + p_{42}\}$$

$$= \max\{8, \max\{7, 6 + 5\} + 1\} = \max\{8, 12\} = 12$$

	$(n-k+1)p_{i_k3}$	$\max\{C_{i_r3}, \max\{C_{i_r2}, \sum_{i \in M} p_{i1} + \min_{i \notin M} p_{i1}\} + \min_{i \notin M} p_{i2}\} + (n-k+1)p_{i_k3}$
k=3	*2.p_{33} = 2*	*14*
k=4	*1.p_{43} = 3*	*15*

Table 3.19 Calculation of LB_3 for Partial sequence [2 1 * *]

$$LB_3 = \sum_{k=r+1}^{n}[\ \max\{C_{i_r 3},\ \max\{C_{i_r 2}, \sum_{i\in M} p_{i1} + \min_{i\notin M} p_{i1}\} + \min_{i\notin M} p_{i2}\} + (n-k+1)p_{i_k 3}] = 14+15 = 29$$

LB(2 1 * *) = $\sum_{i\in M} C_i + \max(LB_1, LB_2, LB_3) = C_2 + C_1 + \max(38, 33, 29) = 15 + 38 = 53$

Since lower bound of partial sequence (2 1 * *) = 53< upper bound, we branch to lower level nodes from partial sequence (2 1* *). Branching from a node consists of taking each of the unallocated jobs in turn and placing it next to the partial schedule. Each of these new partial schedules is then represented by a new node.

Calculating Lower Bound for Partial Sequence [2 1 4 *]

Lower bound for the partial sequence [2 1 4 *] is calculated as follows:

Set of scheduled jobs M = {2, 1, 4} and $|M| = r = 3$

Set of unscheduled jobs U = {3}.

Job i	C_1	C_2	C_3
2	*2*	*5*	*7*
1	*6*	*7*	*8*
4	*11*	*12*	*15*

Table 3.20 Calculation of Completion Times for Partial sequence [2 1 4 *]

For LB_1 from the list of unscheduled the jobs arrange jobs in the increasing order of processing times on machine 1. Sequence of jobs with increasing p_{i1} values is [3].

	$\sum_{i\in M} p_{i1}$	$(n-k+1)p_{i_k 1}$	$p_{i_k 2}$	$p_{i_k 3}$	$\sum_{i\in M} p_{i1} + (n-k+1)p_{i_k 1} + p_{i_k 2} + p_{i_k 3}$
k=4	*11*	$1.p_{31} = 6$	$P_{31} = 5$	$p_{33} = 1$	*23*

Table 3.21 Calculation of LB_1 for Partial sequence [2 1 4 *]

$$LB_1 = \sum_{k=r+1}^{n}[\sum_{i\in M} p_{i1} + (n-k+1)p_{i_k 1} + p_{i_k 2} + p_{i_k 3}] = 23$$

For LB_2 from the list of unscheduled jobs U, schedule the jobs in the increasing order of their processing times on machine 2. Sequence of jobs with increasing p_{i2} values is {3}. The expression,

$\max\{C_{i_r 2}, \sum_{i\in M} p_{i1} + \min_{i\notin M} p_{i1}\}$ = max {12, 11 + p_{31}} = max {12, 11 + 6} = 17

	$(n-k+1)p_{i_k 2}$	$p_{i_k 3}$	$\max\{C_{i_r 2}, \sum_{i\in M} p_{i1} + \min_{i\notin M} p_{i1}\} + (n-k+1)p_{i_k 2} + p_{i_k 3}$
k=4	*1.*p_{32} *= 5*	p_{33} *= 1*	*23*

Table 3.22 Calculation of *LB_2* for Partial sequence [2 1 4 *]

$LB_2 = \sum_{k=r+1}^{n} [\max\{C_{i_r 2}, \sum_{i\in M} p_{i1} + \min_{i\notin M} p_{i1}\} + (n-k+1)p_{i_k 2} + p_{i_k 3}]$ = 23

For LB_3 from the list of unscheduled jobs U, schedule the jobs in the increasing order of processing times on machine 3. Sequence of jobs with increasing p_{i3} values is {3}. The expression,

$\max\{C_{i_r 3}, \max\{C_{i_r 2}, \sum_{i\in M} p_{i1} + \min_{i\notin M} p_{i1}\} + \min_{i\notin M} p_{i2}\}$ = max {15, max {12, 11 + p_{31}} + p_{32}}

= max {15, max {12, 11 + 6} + 5} = max {15, 22} = 22

	$(n-k+1)p_{i_k 3}$	$\max\{C_{i_r 3}, \max\{C_{i_r 2}, \sum_{i\in M} p_{i1} + \min_{i\notin M} p_{i1}\} + \min_{i\notin M} p_{i2}\} + (n-k+1)p_{i_k 3}$
k=4	*1.*p_{33} *= 1*	*23*

Table 3.23 Calculation of *LB_3* for Partial sequence [2 1 4 *]

$LB_3 = \sum_{k=r+1}^{n} [\max\{C_{i_r 3}, \max\{C_{i_r 2}, \sum_{i\in M} p_{i1} + \min_{i\notin M} p_{i1}\} + \min_{i\notin M} p_{i2}\} + (n-k+1)p_{i_k 3}] = 23$

LB(2 1 4 *) = $\sum_{i\in M} C_i$ + max (LB_1, LB_2, LB_3) = C_2 + C_1 + C_4 + max(23, 23, 23) = 7 + 8+ 15 +23 = 53

Since lower bound of partial sequence (2 1 4 *) = 53< upper bound, we branch to lower level nodes from partial sequence (2 1 4 *). Here when we branch to the lower level, we find that the node with partial sequence (2 1 4 3) is a leaf node, so we calculate the completion time of the schedule.

Job i	C_1	C_2	C_3
2	*2*	*5*	*7*
1	*6*	*7*	*8*
4	*11*	*12*	*15*
3	*17*	*22*	*23*

Table 3.24 Calculation of $\sum C_i$ for Schedule [2 1 4 3]

Completion time for the schedule [2 1 4 3] =53 < upper bound. Therefore, now we update the upper bound and the current best order. We now explore other nodes in the search tree with the updated upper bound.

Similarly the Lower bound for the partial sequence [2 1 3 *] = 54, [2 4 * *] = 54, [2 3 * *] = 56, [3 * * *] = 60 and [4 * * *] = 57. Since all lower bounds ≥ upper bound (53), we prune all these nodes.

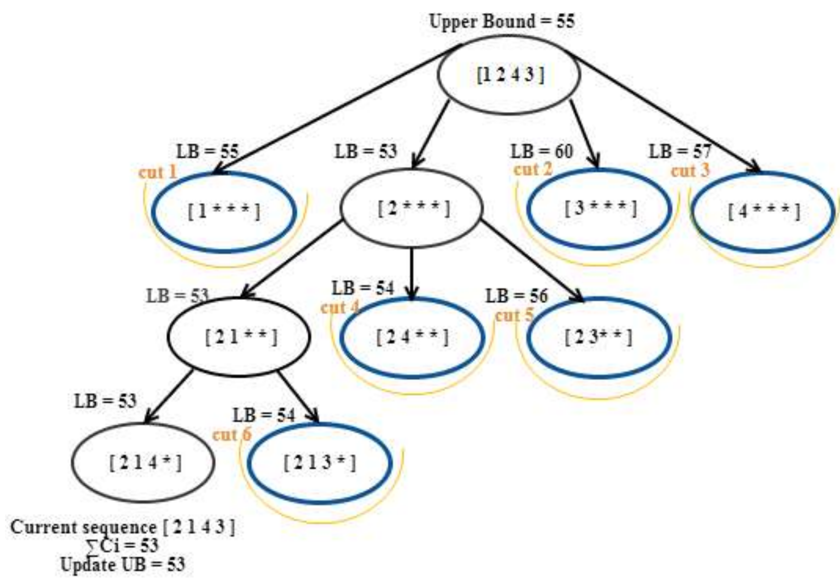


Figure 3.2 Enumeration tree for 4 jobs- 3 machine using branch and bound algorithm

Partial sequence	*Lower bound*	*Current best UB and order*	*Operation*
*[1 * * *]*	*55*	*55, [1, 2, 4, 3]*	*Cut node*
*[2 * * *]*	*53*	*55, [1, 2, 4, 3]*	*Branch from node*
[2 1 *]*	*53*	*55, [1, 2, 4, 3]*	*Branch from node*
*[2 1 4 *]*	*53*	*53, [2, 1, 4, 3]*	*Leaf node, calculate* $\sum C_i$
*[2 1 3 *]*	*54*	*53, [2, 1, 4, 3]*	*Cut node*
[2 4 *]*	*54*	*53, [2, 1, 4, 3]*	*Cut node*
[2 3 *]*	*54*	*53, [2, 1, 4, 3]*	*Cut node*
*[3 * * *]*	*60*	*53, [2, 1, 4, 3]*	*Cut node*
*[4 * * *]*	*57*	*53, [2, 1, 4, 3]*	*Cut node*

Table 3.25 Execution Steps for $F3||\sum C_i$

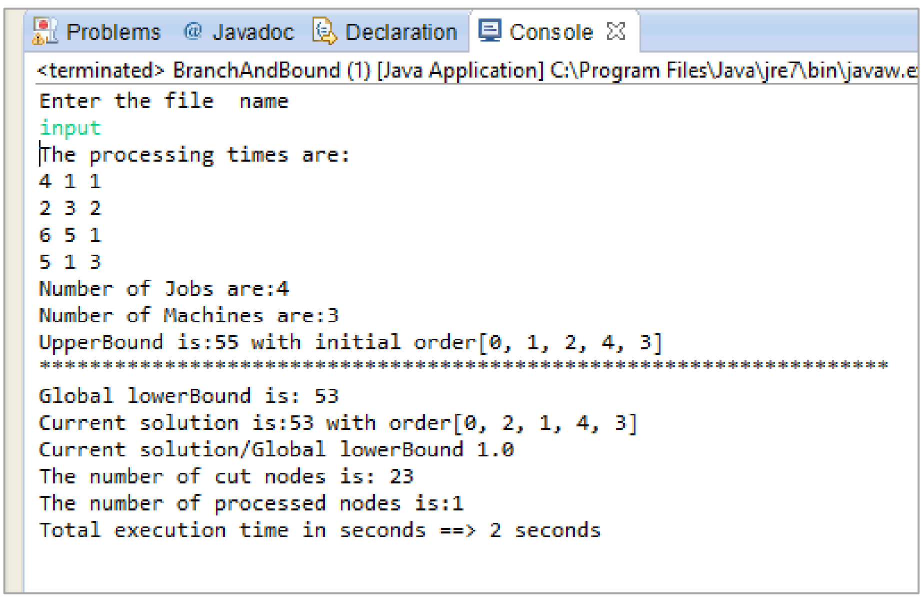


Figure 3.3 Screenshot of the Output for the p_{i1}, p_{i2}, p_{i3} Values Given in Table 3.5

3.6 Branch and Bound Fm-*perm*$||\sum C_i$ ($m \geq 3$)

Branch and bound approach for $F3$-*perm*$||\sum C_i$ can be generalized to m machines. In this case at each node we determine m machine based lower bounds and the overall lower bound is the maximum of the m-lower bounds [8].

Thus the lower bound at node T is,

$$\text{LB [T]} = \sum_{i \in M} C_i + \sum_{i \notin M} C_i$$

$$= \sum_{i \in M} C_i + \max(LB_1, LB_2, \text{LB}_{3\ldots}\ LB_m)$$

Generally to calculate the lower bound on a machine *x*, we assume the possibility that the processing on machine *x* is continuous for the unassigned job set $\{U\}$.

LB_x = earliest time that the first job in the set *U* can start on machine *x* + sum of completion times of jobs in set *U* on machine *x* (here schedule the jobs in the increasing order of processing times on machine *x*. Let *r+1, r+2,…,n* represent the sequence of jobs with increasing p_{ix} values) + sum of processing times of jobs in set *U* on remaining *(m-x)* machines.

Let *r* be the last job in the ordered set of scheduled jobs *M*, then

$$LB_x = \max\{C_{r,x}, \max_{1 \le y \le x-1}\{C_{r,y} + \min_{i \notin M} \sum_{j=y}^{x-1} p_{i_k,j}\}\} + \sum_{k=r+1}^{n} (n-k+1) p_{i_k,x} + \sum_{k=r+1}^{n} \sum_{j=x+1}^{m} p_{i_k,j}$$

CHAPTER 4

RESULTS

This chapter gives a detailed view of the results obtained by applying Branch and Bound algorithm for permutation $F3\text{-}perm||\sum C_i$ scheduling problem.

4.1 Assumptions

Following assumptions are made while implementing the algorithm:

1. The algorithm initializes the branch and bound tree with an initial feasible schedule and an initial upper bound. Initial feasible schedule is obtained by arranging the jobs in the increasing order of their sum of processing times.

2. Initial Upper bound can be obtained by calculating the sum of completion times of initial feasible schedule.

4.2 Parameters that Determine Performance of the Algorithm

Initial Upper Bound

Initial Upper bound is obtained by calculating the sum of completion times of initial feasible schedule.

Global Lower bound

The minimum lower bound on the highest level nodes corresponds to global lower bound. We start the branch and bound tree with initial upper bound. Then we try to branch in this tree by trying to fix each of the jobs as the first job in the sequence. From the data given in Table 3.5 and Figure 3.2, there are four jobs, so possible branches are four.

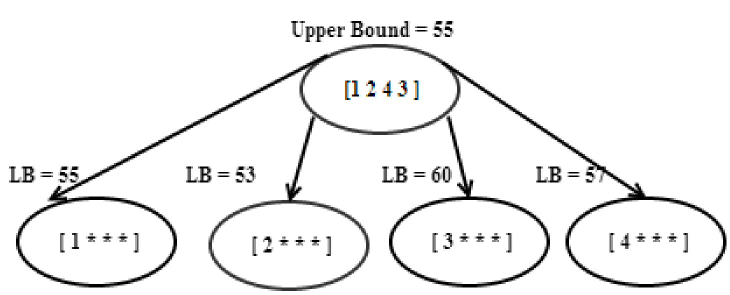


Figure 4.1 Branching Tree Showing the Highest Level Nodes

Then we calculate the lower bound for these nodes. The minimum lower bound on these highest level nodes corresponds to the global lower bound because all the corresponding branches emerging from these nodes have lower bound greater than or equal to it. For the above problem, global lower bound = 53.

Current Best Solution

Cost of the schedule obtained by applying branch and bound algorithm represents the current best solution.

Performance Ratio

$$\%\ increase\ over\ the\ optimal\ solution = \frac{Current\ best\ solution}{Global\ LB} * 100$$

% increase over the optimal solution is used to analyze the performance of branch and bound algorithm. Branch and bound is one of the heuristic to determine near optimal solution, but it does not guarantee to provide an optimal solution. Therefore we use performance ratio in order to determine the percentage of deviation of current best solution obtained from optimal.

Execution Time

Execution time is the time taken by the branch and bound program to determine the current best solution. From the above results we see that, branch and bound performs much better in practice than the complete enumeration.

Number of Eliminated Sequences

If the lower bound of the sub problem is greater than or equal to upper bound, then this sub problem cannot yield a better solution and we stop branching from the corresponding node in the branching tree. Thus we prune all the branches emerging from that node.

If there are n jobs and if a node at k^{th} level is pruned, then we eliminate $(n-k)!$ sequences from processing.

4.3 Results for Various p_{i1}, p_{i2} and p_{i3} Values

Following results are obtained by applying branch and bound algorithm. Computational results for up to 20 jobs are given for 3 machine permutation flow shop problem when the objective is minimizing the sum of completion times.

4.3.1 Random p_{i1}, p_{i2} and p_{i3} Values

For randomly chosen p_{i1}, p_{i2} and p_{i3} values given in the Table 4.1 the results obtained by executing branch and bound algorithm for the objective function $\sum C_i$ are presented in Table 4.2 and Table 4.3.

Job i	p_{i1}	p_{i2}	p_{i3}
1	6	4	2
2	8	5	8
3	1	1	1
4	5	8	3
5	9	3	1
6	9	2	4
7	7	6	6
8	4	3	7
9	6	3	2
10	4	3	1
11	7	1	4
12	2	9	3
13	7	2	8
14	3	6	1
15	2	6	1
16	1	8	5
17	4	5	3
18	9	3	2
19	4	6	1
20	6	5	7

Table 4.1 Random p_{i1}, p_{i2} and p_{i3} values for n up to 20

For the first n jobs	*Initial UB*	*Global LB*	*Current Best solution*	*(Current solution/Global LB)*100*	*Execution time(sec)*
n=10	357	331	334	1.0090635	2
n=11	428	397	400	1.0075567	2
n=12	490	432	446	1.0324074	2
n=13	575	512	526	1.0273438	2
n=14	623	558	581	1.0412186	2
n=15	671	594	623	1.0488216	2
n=16	769	623	681	1.093097	5
n=17	846	688	756	1.0988373	22
n=18	940	787	855	1.0864041	29
n=19	1025	859	940	1.0942957	213
n=20	1139	961	1045	1.0874089	548

Table 4.2 $\sum C_i$ Results for Random Values of p_{i1}, p_{i2} and p_{i3}

For the first n jobs	*No of eliminated sequences*	*No of processed sequences*	*Current best order*
n=10	3628786	14	[3, 10, 8, 1, 9, 4, 2, 7, 5, 6]
n=11	39916782	18	[3, 10, 8, 4, 9, 1, 11, 7, 2, 5, 6]
n=12	479001576	24	[3, 12, 10, 9, 8, 4, 1, 11, 7, 2, 5, 6]
n=13	6227020766	34	[3, 12, 10, 9, 8, 4, 1, 11, 13, 7, 2, 5, 6]
n=14	87178291172	28	[3, 14, 10, 8, 12, 9, 13, 4, 1,11, 7, 2, 5, 6]
n=15	1307674367966	34	[3, 15, 10, 8, 14 ,13, 12, 9, 1, 11, 4, 2, 7, 5, 6]
n=16	20922789887949	51	[3, 16, 10, 9, 8, 15, 13, 12, 11, 14, 1, 4, 2, 7, 5, 6]
n=17	355687428095950	50	[3, 16, 10, 9, 8, 15, 13, 12, 11, 14, 1, 17, 4, 2, 7, 5, 6]
n=18	6402373705727933	67	[3, 16, 10, 9, 8, 15, 13, 12, 11, 14, 1, 17, 4, 2, 7, 5, 18, 6]
n=19	6402373705727933	66	[0, 3, 16, 10, 9, 8, 15, 14, 11, 17, 19, 13, 12, 1, 7, 4, 5, 2, 18, 6]
n=20	2432902008176639922	78	[3, 16, 10, 9, 8, 15, 14, 11, 17, 19, 13, 12, 1, 7, 20, 4, 2, 5, 18, 6]

Table 4.3 Results of Branch and Bound Algorithm for *n* up to 20

4.3.2 Large Values of p_{i1}, p_{i2} and p_{i3}

For the large values of p_{i1}, p_{i2} and p_{i3} given in the Table 4.4 the results obtained by executing branch and bound algorithm for the objective function $\sum C_i$ are presented in Table 4.5.

Job i	p_{i1}	p_{i2}	p_{i3}
1	375	12	142
2	632	452	758
3	12	876	124
4	460	542	523
5	528	101	789
6	796	245	632
7	532	230	543
8	14	124	214
9	257	527	753
10	896	896	214
11	532	302	501
12	456	856	963
13	789	930	21
14	630	214	475
15	214	257	320
16	573	896	124
17	218	532	752
18	653	142	147
19	214	547	532
20	204	865	145

Table 4.4 Large Values of p_{i1}, p_{i2} and p_{i3}

For the first n jobs	*Initial UB*	*Global LB*	*Current solution*	*(Current solution/Global LB)*100*	*Execution time(sec)*	*No of processed sequences*
n=10	30633	25538	28882	1.1309421	2	13
n=11	36190	30647	34278	1.1184782	2	11
n=12	43939	36772	41281	1.1226205	2	14
n=13	51139	43886	48298	1.1005332	2	15
n=14	58536	50891	55588	1.0922953	2	20
n=15	64357	54276	59175	1.0902609	3	33
n=16	72334	62114	66636	1.0728016	3	44
n=17	81438	66690	74394	1.1155195	24	71
n=18	90860	75024	81613	1.0878252	32	87
n=19	103635	79767	88553	1.1101458	241	146
n=20	118496	84475	96059	1.1371293	1241	282

Table 4.5 $\sum C_i$ Results for Large Values of p_{i1}, p_{i2} and p_{i3}

Job i	p_{i1}	p_{i2}	p_{i3}
1	1	20	1
2	2	19	2
3	3	18	3
4	4	17	4
5	5	16	5
6	6	15	6
7	7	14	7
8	8	13	8
9	9	12	9
10	10	11	10
11	11	10	11
12	12	9	12
13	13	8	13
14	14	7	14
15	15	6	15

Table 4.6 Increasing p_{i1}, p_{i3} and Decreasing p_{i2} Values

4.3.3 Increasing p_{i1}, p_{i3} and Decreasing p_{i2} Values

For a particular case where the values of p_{i1}, p_{i3} increases and p_{i2} decreases as the job index increases are given in the Table 4.6 and the results obtained by executing branch and bound algorithm for the objective function $\sum C_i$ are presented in Table 4.7.

For the first n jobs	*Initial UB*	*Global LB*	*Current solution*	*(Current solution/Global LB)*100*	*Execution time(sec)*	*No of processed sequences*
n = 3	204	200	200	1.0	2	3
n = 5	300	290	290	1.0	2	8
n = 7	539	504	504	1.0	2	32
n = 9	834	750	750	1.0	2	81
n=10	1000	880	880	1.0	2	117
n=11	1177	1012	1012	1.0	2	162
n=12	1366	1144	1147	1.0026224	2	211
n=13	1568	1274	1285	1.0086342	2	263
n=14	1784	1400	1430	1.0214286	2	317
n=15	2015	1520	1582	1.0407895	12	358

Table 4.7 $\sum C_i$ Results for Increasing p_{i1}, p_{i2} and Decreasing p_{i3} Values

4.3.4 p_{i1}, p_{i3} Increases and then Decreasing

The values of p_{i1}, p_{i3} increases as the job index increases and decreases after $i > (n+1) / 2$ and the values of p_{i2} decreases as the job index increases and increases after $i > (n+1) / 2$ are given in the Table 4.8. The results obtained by executing branch and bound algorithm for the objective function $\sum C_i$ are presented in Table 4.9.

Job i	***p_{i1}***	***p_{i2}***	***p_{i3}***
1	1	20	1
2	2	19	2
3	3	18	3
4	4	17	4
5	5	16	5
6	6	15	6
7	7	14	7
8	8	13	8
9	9	12	9
10	10	11	10
11	10	11	10
12	9	12	9
13	8	13	8
14	7	14	7
15	6	15	6
16	5	16	5
17	4	17	4
18	3	18	3
19	2	19	2
20	1	20	1

Table 4.8 p_{i1}, p_{i3} Increasing and then Decreasing

For the first n jobs	*Initial UB*	*Global LB*	*Current solution*	*(Current solution/Global LB)*100*	*Execution time(sec)*	*No of processed sequences*
n=15	2045	1775	1775	1.0	2	268
n=16	2317	2011	2011	1.0	2	304
n=17	2617	2266	2266	1.0	2	349
n=18	2948	2540	2540	1.0	2	406
n=19	3313	2833	2833	1.0	2	478
n=20	3715	3145	3145	1.0	2	568

Table 4.9 $\sum C_i$ Results for p_{i1}, p_{i2} Increasing and then Decreasing

4.4 Efficiency of the Algorithm

In order to validate practically the efficiency of branch and bound algorithm, the results of the algorithm are compared with the results obtained by generating all the $n!$ permutations sequences, when number of jobs $(n \leq 12)$. From the results obtained, we see that branch and bound performs much better in practice than the complete enumeration. From the experiments, we notice that instead of searching entire solution space branch and bound algorithm pruned many nodes and this considerably reduced the computational time.

Branch and bound algorithm is used to determine near optimal solution, but it does not guarantee to provide an optimal solution. Therefore we use performance ratio in order to determine the percentage of deviation of branch and bound solution from optimal. From the results obtained we see a difference of approximately 1.1% between the branch and bound solution and optimal solution.

CHAPTER 5

CONCLUSION AND FUTURE WORK

In this thesis, we have presented, evaluated and implemented the branch and bound algorithm to minimize the sum of completion times for three machine permutation flow shop problem. We presented the lower bounds, upper bounds and performance ratio for the various problems. In general, our results consistently give solutions with a ratio of better than 1.1% of optimal. We indeed observed that a significant number of sub problems can be eliminated from further consideration, if the initial upper bound is tight. Therefore we can use some good heuristics to obtain better initial upper bound.

As n grows, the branch and bound algorithm is obviously exponential in time but performs much better in practice than the complete enumeration. The future work would be to improve lower bounds for minimizing the sum of completion times of n jobs over m machines.

APPENDIX

```
package sumci;

import java.io.*;
import java.util.*;

public class BranchAndBound {
	static public int njobs, nmachines;
	static public int[][] p;
	static int[][] c;
	static long cb_ub = 100000000;
	static long global_lb = 100000000;
	static String cb_order;
	Map<Integer, Integer> sorted_pmac1;
	Map<Integer, Integer> sorted_pmac2;
	Map<Integer, Integer> sorted_pmac3;
	static int[] job_arr;
	static long processd_node = 0;
	static long count = 0;

	// Method to read data from input file
	public static void readData(String filename) {
		Scanner sc = null;
		try {
			sc = new Scanner(new FileReader(filename));
		} catch (Exception e) {
			System.out.println("could not find the file ");
		}

		njobs = sc.nextInt();
		nmachines = sc.nextInt();
		p = new int[njobs + 1][nmachines + 1];

		System.out.println("The processing times are:");
		for (int j = 1; j <= njobs; j++) {
			for (int m = 1; m <= nmachines; m++) {
				p[j][m] = sc.nextInt();
				System.out.print(p[j][m] + " ");
			}
			System.out.println();
		}

		System.out.println("Number of Jobs are:" + njobs);
		System.out.println("Number of Machines are:" + nmachines);

		sc.close();
	}

	// Method to calculate completion time of the given jobs and sumci for the
	// given schedule
	public int calComp(int[] a) {
		c = new int[njobs + 1][nmachines + 1];
		int sumci = 0;
		for (int j = 1; j <= a.length - 1; j++) {
			for (int m = 1; m <= nmachines; m++) {
```

```
					c[a[j]][m] = Math.max(c[a[j]][m - 1], c[a[j - 1]][m])
									+ p[a[j]][m];
				}
		}
		for (int j = 1, m = nmachines; j <= a.length - 1; j++)
				sumci = sumci + c[j][m];
		return sumci;
}

public long fact(long n) {
		if ((n == 0 || n == 1))
				return 1;
		else
				return n * fact(n - 1);
}

// Method to calculate LB1, LB2, LB3
public int calclb(int[] temparr, int[] temparr2, int[] temparr3,
				boolean[] used, int level) {

		int lb1 = 0, lb2 = 0, lb3 = 0, max_job;

		int compvl = c[temparr[level]][2];
		int comp = c[temparr[level]][3];
		int sumpi = 0;
		for (int i = 1; i <= level; i++) {
				sumpi = sumpi + p[temparr[i]][1];
		}
		int j = level + 1;
		for (Integer index : sorted_pmac1.keySet()) {
				if (used[index]) {
				} else {
					temparr[j] = index;
					j++;
				}
		}

		int pos = level + 1;
		for (Integer ind : sorted_pmac2.keySet()) {
				if (used[ind]) {
				} else {
					temparr2[pos] = ind;
					pos++;
				}
		}
		int i3 = level + 1;
		for (Integer it : sorted_pmac3.keySet()) {
				if (used[it]) {
				} else {
					temparr3[i3] = it;
					i3++;
				}
		}

		int val = Math.max(compvl, sumpi + p[temparr[level + 1]][1]);
		int intermediate = Math.max(val + p[temparr2[level + 1]][2], comp);
```

```
		for (int k = level + 1; k <= njobs; k++) {

			lb1 = lb1 + sumpi + ((njobs - k + 1) * p[temparr[k]][1])
					+ p[temparr[k]][2] + p[temparr[k]][3];
			lb2 = lb2 + val + ((njobs - k + 1) * p[temparr2[k]][2])
					+ p[temparr2[k]][3];
			lb3 = lb3 + intermediate + ((njobs - k + 1) * p[temparr3[k]][3]);
		}
		// System.out.println("lb1"+" "+lb1+" "+"lb2"+" "+lb2+" "+"lb3"+" "+lb3);
		if (lb1 > lb2 && lb1 > lb3)
			max_job = lb1;
		else if (lb2 > lb1 && lb2 > lb3)
			max_job = lb2;
		else
			max_job = lb3;

		return max_job;
	}

	// Method to generate a node(new partial sequence of jobs)
	public void generateNode(int[] arr, int level) {

		for (int job = 1; job <= njobs; job++) {
			boolean[] used = new boolean[njobs + 1];
			int[] fixed_Jobarr = new int[level + 1];
			int[] temparr_P1 = new int[njobs + 1];
			int[] temparr_P2 = new int[njobs + 1];
			int[] temparr_P3 = new int[njobs + 1];

			int current_lb = 0;

			for (int i = 1; i < level; i++) {
				temparr_P1[i] = arr[i];
				temparr_P2[i] = arr[i];
				temparr_P3[i] = arr[i];
				fixed_Jobarr[i] = arr[i];
				used[arr[i]] = true;
			}
			if (used[job_arr[job]]) {
			} else {
				temparr_P1[level] = job_arr[job];
				temparr_P2[level] = job_arr[job];
				temparr_P3[level] = job_arr[job];
				fixed_Jobarr[level] = job_arr[job];
				used[job_arr[job]] = true;

				int finishedjobs = calComp(temparr_P1);
				if (level == njobs && finishedjobs < cb_ub) {
					cb_order = Arrays.toString(temparr_P1);
					cb_ub = finishedjobs;
					processd_node++;
					// System.out.println("cb_order" + " " + cb_order + " "
					// + "cb_ub" + " " + cb_ub);
				} else {
					int max_job = calclb(temparr_P1, temparr_P2, temparr_P3,
```

```
                    used, level);
                current_lb = finishedjobs + max_job;
                // System.out.println("current lb" + " " + current_lb + " "
                // + Arrays.toString(fixed_Jobarr));
                if (current_lb >= cb_ub) {
                    count = count + fact(njobs - level);
                } else {

                    generateNode(fixed_Jobarr, level + 1);
                }
            }
        }
    }
}

// To sort jobs based on processing times of machine1, machine2, machine3
public void sortingJobs() {
    Map<Integer, Integer> unsorted_pmac1 = new HashMap<Integer, Integer>();
    Map<Integer, Integer> unsorted_pmac2 = new HashMap<Integer, Integer>();
    Map<Integer, Integer> unsorted_pmac3 = new HashMap<Integer, Integer>();

    for (int i = 1; i <= njobs; i++) {
        unsorted_pmac1.put(i, p[i][1]);
        unsorted_pmac2.put(i, p[i][2]);
        unsorted_pmac3.put(i, p[i][3]);
    }
    sorted_pmac1 = sortByComparator(unsorted_pmac1);
    sorted_pmac2 = sortByComparator(unsorted_pmac2);
    sorted_pmac3 = sortByComparator(unsorted_pmac3);

}

private Map<Integer, Integer> sortByComparator(
        Map<Integer, Integer> unsortedhm) {

    List list = new LinkedList(unsortedhm.entrySet());
    Collections.sort(list, new Comparator() {
        @Override
        public int compare(Object o1, Object o2) {
            return ((Comparable) ((Map.Entry) (o1)).getValue())
                    .compareTo(((Map.Entry) (o2)).getValue());
        }
    });

    Map sortedMap = new LinkedHashMap();
    for (Iterator it = list.iterator(); it.hasNext();) {
        Map.Entry entry = (Map.Entry) it.next();
        sortedMap.put(entry.getKey(), entry.getValue());
    }
    return sortedMap;
}

public static void main(String args[]) throws IOException {

    long start = System.currentTimeMillis();
```

```
		Scanner read_filename = new Scanner(System.in);
		System.out.println("Enter the file  name");
		String filename = read_filename.next();
		read_filename.close();
		BranchAndBound.readData(filename);
		BranchAndBound obj = new BranchAndBound();

		Map<Integer, Integer> initial_arr = new HashMap<Integer, Integer>();

		for (int i = 1; i <= njobs; i++)
			initial_arr.put(i, p[i][1] + p[i][2] + p[i][3]);
		initial_arr = obj.sortByComparator(initial_arr);

		job_arr = new int[njobs + 1];
		int position = 1;
		for (Integer index : initial_arr.keySet()) {
			job_arr[position] = index;
			position++;
		}

		int initial_ub = obj.calComp(job_arr);

		if (cb_ub > initial_ub) {
			cb_ub = initial_ub;
			cb_order = Arrays.toString(job_arr);
			System.out.println("UpperBound is:" + cb_ub + " "
					+ "with initial order" + cb_order);
		}

		obj.sortingJobs();

		for (int i = 1; i <= njobs; i++) {
			int[] sub1_arr = new int[njobs + 1];
			int[] sub2_arr = new int[njobs + 1];
			int[] sub3_arr = new int[njobs + 1];
			boolean[] use = new boolean[njobs + 1];
			sub1_arr[1] = job_arr[i];
			sub2_arr[1] = job_arr[i];
			sub3_arr[1] = job_arr[i];
			// System.out.print("[" + sub1_arr[1] +"]");
			use[job_arr[i]] = true;
			int finished = obj.calComp(sub1_arr);
			int lb = obj.calclb(sub1_arr, sub2_arr, sub3_arr, use, 1);
			int result = lb + finished;
			global_lb = Math.min(global_lb, result);
			// System.out.println(" "+"lb" + " " + result);

		}

		// For Branching the problem P
		obj.generateNode(job_arr, 1);

		System.out

.println("*****************************************************************");
```

```
        System.out.println("Global lowerBound is:" + " " + global_lb);
        System.out.println("Current solution is:" + cb_ub + " " + "with order"
                + cb_order);

        float percent = (float) cb_ub / global_lb;
        System.out
                .println("Current solution/Global lowerBound" + " " + percent);

        System.out.println("The number of cut sequences is:" + " " + count);
        System.out.println("The number of processed sequences is:"
                + (obj.fact(njobs) - count));

        long end = System.currentTimeMillis();
        System.out.println("Total execution time" + " in seconds ==> "
                + (end - start) / 1000 + " seconds");

    }

}
```

BIBLIOGRAPHY

[1] Parker, R. G, *Deterministic scheduling theory*, First Edition, Chapter 5, Chapman & Hall, 1995.

[2] P. Brucker, *Scheduling Algorithms,* Fifth Edition, Chapter 1, 3, 6, Springer Publications, March 2007.

[3] Michael L. Pinedo, *Scheduling Theory Algorithms and Systems*, Fourth Edition, Chapter 2, Springer Publishers, January 2012.

[4] Joseph Y-T. Leung, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, 2004.

[5] Adusumilli K, Bein D, Bein W, *A Genetic Algorithm for the Two Machine Flow Shop Problem*, IEEE Computer Society Press, 2008.

[6] S. M. Johnson, *Optimal two- and three-stage production schedules with setup times included*, Naval Research Logistics Quarterly, Vol. 1, No. 3, 1954, pp. 61-68.

[7] E. Ignall and L. Schrage, *Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Prob-lems*, Operations Research, Vol. 13, No. 3, 1965, pp. 400-412.

[8] S. P. Bansal, *Minimizing the Sum of Completion Times of n Jobs Over m Machines in a Flowshop—A Branch, Bound Approach*, AIIE Transactions, Vol. 9, 1977, pp. 306-311.

[9] R. H. Ahmadi and U. Bagchi, *Improved Lower Bounds for Minimizing the Sum of Completion Times of n Jobs Over m Machines in a Flow Shop*, European Journal of Operational Research, Vol. 44, 1990, pp. 331-336.

[10] Amit Nagar, Sunderesh S. Heragu and Jorge Haddock, *A Branch-and-Bound Approach for a Two-machine Flowshop Scheduling Problem,* Operational Research Society, Vol. 46, No. 6, 1995, pp. 721-734.

[11] G. B. McMahon and P. G. Burton, *Flow-Shop Scheduling with the Branch-And-Bound Method, Operations Research* , Vol. 15, No. 3, 1967, pp. 473-481.

[12] M. R. Garey, D. S. Johnson and Ravi Sethi, *The complexity of flow shop and job shop scheduling*, Mathematics of Operations Research, 1976, pp. 117-129.

[13] T. Gonzalez and S. Sahni, *Flow shop and Job Shop Schedules: Complexity and Approximation*, Operations Research, Vol.26, 1978, pp. 36-52.

[14] Kenneth R. Baker, *A Comparative Study of Flow-Shop Algorithms, Operations Research*, Vol. 23, No. 1, 1975, pp. 62-73.

[15] W. C. Yeh, *An efficient branch-and bound algorithm for the two-machine bicriteria flowshop scheduling problem,* Journal of Manufacturing Systems, Vol. 20, No. 2, 2001, pp. 113-123.

[16] C. S. Chung, J. Flynn and O. Kirca, *A Branch and Bound Algorithm to Minimize the Total Flow Time for m-Machine Permutation Flowshop Problems*, International Journal of Production Economics, Vol. 79, No. 3, 2002, pp. 185-196.

[17] S. Kouki, M. Jemni and T. Ladhari, *Solving the Permutation Flow Shop Problem with Makespan Criterion using Grids*, International Journal of Grid and Distributed Computing Vol.4, No. 2, June, 2011.

[18] R. A. Dudek, S. S. Panwalker and M. L. Smith, *The Lessons of Flowshop Scheduling Research*, Operations Research, Vol. 40, No. 1, 1992, pp. 7-13.

[19] Z. A. Lomnicki, A Branch-and-Bound Algorithm for the Exact Solution of the Three- Machine Scheduling Problem, Operations Research Vol. 16, No. 1, 1965, pp. 89-100.

[20] B. J. Lageweg, J. K. Lenstra and A. H. G. Rinnooy Kan, *A General Bounding Scheme for the Permutation Flow-Shop Problem*, Operations Research , Vol. 26, No. 1, 1978, pp. 53-67

[21] M. K. Yang and C. R. Das, *Evaluation of a Parallel Branch-and-Bound Algorithm on a Class of Multiprocessors*, IEEE Transactions on Parallel and Distributed Systems , Vol. 5, No. 1, 1994, pp. 74-86.

[22] Shaukat A. Brah and John L. Hunsucker, *Branch and bound algorithm for the flow shop with multiple processors*, European Journal of Operational Research, Vol. 51, 1991, pp. 88-99.

[23] J. Gao, and R. Chen. *An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems, Scientific Research and Essays,* Vol. 6, 2011, pp. 3094-3100

[24] C. N Potts, *An adaptive branching rule for the permutation flow-shop problem*, European Journal of Operational Research, Vol. 5, 1980, pp. 19-25.

VITA

Graduate College

University of Nevada, Las Vegas

Swapna Kodimala

Degrees:

Bachelor of Technology in Information Technology, 2012

Jawaharlal Nehru Technological University

Master of Science in Computer Science, 2014

University of Nevada Las Vegas

Thesis Title: A Branch and Bound Method for Sum of Completion Permutation Flow Shop

Thesis Examination Committee:

Chair Person, Dr. Wolfgang Bein, Ph.D.

Committee Member, Dr. Laxmi P. Gewali, Ph.D

Committee Member, Dr. Ajoy K. Datta, Ph.D.

Graduate College Representative, Dr. Zhiyong Wang, Ph.D.