8-1-2014

A Taxonomy of Polynomially Solvable Shop Problems with Limited Number of Machines or Jobs

Megha Sairam Darapuneni
University of Nevada, Las Vegas, darapun3@unlv.nevada.edu

A TAXONOMY OF POLYNOMIALLY SOLVABLE SHOP PROBLEMS WITH LIMITED NUMBER OF MACHINES OR JOBS

By

Megha Sairam Darapuneni

A Thesis submitted in partial fulfillment

of the requirements for

Master of Science in Computer Science

Department of Computer Science

Howard R. Hughes College of Engineering

The Graduate College.

University of Nevada, Las Vegas

May, 2014.

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Megha Sairam Darapuneni

entitled

A Taxonomy of Polynomially Solvable Shop Problems with Limited Number of Machines or Jobs

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

Master of Science in Computer Science
Department of Computer Science

Wolfgang Bein, Ph.D., Committee Chair

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

Laxmi Gewali, Ph.D., Committee Member

Venkatesan Muthukumar, Ph.D., Graduate College Representative

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

August 2014

ABSTRACT

A Taxonomy of Polynomially Solvable Shop Problems with Limited Number of Machines or Jobs

By

Megha Sairam Darapuneni

Dr. Wolfgang Bein, Examination Committee Chair

Professor, Department of Computer Science

University of Nevada, Las Vegas.

Among shop scheduling problems, job shop and mixed shop are one of the most general models encompassing open shop and flow shop. Many job shop problems are NP hard, but there are numerous cases, which possess polynomial solutions when the number of jobs or the number of machines (or both) is limited.

This thesis gives an overview of methods and algorithms for solving – in polynomial time – such special shop problems, including open, flow, job shop and mixed shop. The tools used include Monge interchange, dynamic programming, greedy techniques and sweep line algorithms and the primary focus of this thesis is to give a taxonomy of such problems with their solutions. Additionally the thesis outlines a neighborhood search technique which uses the disjunctive graph model and which can be applied as a heuristic for a wide range of NP-hard shop problems.

ACKNOWLEDGEMENT

It is with immense gratitude that I acknowledge the support and help of my Professors, friends and family. Firstly, I wish to thank my committee chair, Professor Dr. Wolfgang Bein, a great enthusiast and a teacher. Without his continuous support, guidance and encouragement, this thesis would not have been possible. I feel lucky to be his teaching assistant and enjoyed every moment working with him .

I would like to thank my graduate advisor and committee member, Dr. Ajoy K.Datta for giving me an opportunity to me as a graduate assistant in computer science department and trusting in me. I would also like to thank Dr. Laxmi P. Gewali, Dr. Venkatesan Muthukumar for readily accepting my invitation to serve on my committee.

I would like to thank my friends and classmates who stood by my side and helped me throughout my Masters. I would be indebted to the department of computer science and the graduate college-University of Nevada, Las Vegas for making me feel like home and supporting me financially.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The main purpose of this paper is to study and assess various methods for solving the shop scheduling problems in polynomial time. In addition, we look into methods for achieving near optimal solutions for NP-hard shop problems. We cover a wide range of polynomially solvable shop problems along with some of the special and harder cases.

Scheduling in general is the process of decision making and a time management tool to utilize the available resources in a effective manner for a productive outcome. Making up a schedule is a common activity for every person in every walk of life. Though it may seem to be an easy process, there have been many NP-hard[5] scheduling problems in computer science which have not been resolved for the past 60 years and more . The difficulty and the wide application of scheduling is what makes this a hot topic of research even today. Here, we are going to discuss about different scheduling shop problems along with some of the algorithms for solving these problems.

Among the shop problems, the job shop scheduling problem is considered the most difficult case . It is regarded as one of the most difficult NP-hard, combinatorial problem. The flow shop and the open shop scheduling problems are considered special cases of the more general job shop problem . So, we start with the classic geometric approach for solving the job shop scheduling problem with two machines. Then, we are going to discuss the two machine job shop algorithm, flow shop problem using the Johnsons algorithm and then the two machine open shop algorithm . Similarly, we deal with a set of polynomially solvable shop problems. Finally, we discuss the disjunctive graph model and its application in meta

heuristic methods like local neighborhood search and simulated annealing for finding near optimal solutions for NP-hard problems.

Let us take a brief look into what is coming in the following chapters and how this paper is organized. Chapter 1 deals with the introduction and the outline of this paper. In Chapter 2, we are introduced to the concept of scheduling, scheduling problems and the notations used. In this chapter, we get an insight into the scheduling problems and are presented with examples for the different shop scheduling problems. We start with the job shop scheduling problem in chapter 3 where we will be looking into the classical geometric approach for solving the job shop problem with two jobs. In chapter 4, we discuss the two machine job shop problem. Chapter 5 explains the Johnson's algorithm for solving the two machine flow shop problem. Chapter 6 gives us an insight into the two machine open shop problem and the algorithm. In chapter 7, we are going to discuss the mixed shop problems. In Chapter 8, we look into some of the methods for dealing with NP-hard shop problems. Lastly in chapter 9, we will be finishing off our paper with a good conclusion.

Chapter 2

Scheduling Problem Notations and Examples

2.1 Introduction to Scheduling and Scheduling Notations

Let us suppose we have a certain number of jobs to be done with a limited amount of resources available(i.e. time, workforce, etc). we need to learn the concept of scheduling to complete our task at ease. Let us first learn what scheduling means. **Scheduling** is the process of deciding how to assign our available resources between a variety of possible tasks trying to optimize one or more performance measures. As we can understand, there can be many types of scheduling problems based on many factors. Among those problems, we are going to deal with Job Shop Scheduling Problem.

According to Peter Brucker [1], a schedule can be defined as : Given *m* number of machines M*j* (*j* = 1,2,3,.....m) and *n* number of jobs J*i* (*i* = 1,2,3,4,....n). A **schedule** can be defined as, allocating one or more time intervals to one or more machines for any given job. A Schedule is usually represented by using **Gantt** charts.

2.1.1 Notations in Scheduling

To understand the representation of a scheduling problem, we are required to learn the meaning of the following notations :

α Notation - Machine Environment,

β Notation - Job Characteristics and

γ Notation - Optimality Criterion.

A scheduling problem is defined by the above elements where the α notation represents the machine environment, β represents the job characteristics and γ represents the optimality

criterion of a given problem. Each of these notations inturn have a lot of possibilities. These notations were brought into light by Graham et al.

Any job j_i can consist of *n* number of operations $o_{i1},o_{i2},.....,o_{in}$. Each operation o_{ij} can in turn have a processing requirement p_{ij}. Likewise ,we can have many more factors coming into play in a scheduling problem like :

Processing time p_{ij} - Processing time can be defined as the amount of time taken to process an operation o_{ij} on a particular machine.

Idle time - Idle time can be defined as the amount of time a machine is kept idle without any operation being processed on it .

Release time r_i - This attribute implies when a particular job or operation would be available for processing.

Due date(Deadline) d_i -This attribute implies when a job is supposed to be completed.

Completion time C_i - It is the completion time of a any job present in our schedule.

Makespan max{C_i ; i = 1,2,3.....,*n*} - Makespan can be defined as the maximum of completion times of all jobs on our schedule.

Total Flow Time $\sum_{i=1}^{n} Ci$ - Sum of completion times of all the jobs on our schedule.

Lateness L_i - It is the extra time taken by a job beyond its deadline.

$$L_i = C_i - d_i .$$

Earliness E_i - It is the difference between the deadline and completion time of a job and is greater than or equal to zero.

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

Tardiness T_i - It is the difference between the completion time and the due date and cannot be less than zero.

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

Absolute deviation $D_i = | C_i - d_i |$.

Squared deviation $S_i = (C_i - d_i)^2$.

Unit Penalty U_i

$$U_i = \begin{cases} 0 \text{ if } C_i <= d_i ; \\ 1 \text{ if otherwise.} \end{cases}$$

2.2 Machine Environment

The machine environment is denoted by α symbol where $\alpha = \alpha_1\alpha_2$. we can have the following possible values for α_1 = o, P , Q , R, PMPM, QMPM , G , X , O , J , F where o is nothing but void. When α_1= o, then $\alpha = \alpha_2$.

α_2 represents the number of machines and can be any positive integer. When α_2 = x, where x represents a random positive integer, it implies a fixed number of machines present .When the number of machines present is arbitrary, α_2 would be set to α_2 = o.

Each of these notation when considered alone (with α_1 = o or α_2 = o) or with any other combinations imply a meaning. Following are some of the important cases we could see in the machine environment for a given problem :

When **α_1 = o**, the given scheduling problem implies that each of the job has to be processed on a **dedicated machine**.

When any job in our given schedule is allowed to be processed on any of the machines**(Parallel Machines)** available M_1, M_2,....., M_m then in the machine environment , α_1 can have values P,Q or R[13] **($\alpha_1 \epsilon$ { P , Q , R }).**We need to remember that a machine

can process at most one job at a time and a particular job cannot be processed on more than one machine at a time.

When the value of $\boldsymbol{\alpha_1 = P}$, it implies that the machines are **identical parallel machines** .This means that any job on our schedule would require p_j units of time to be processed on any machine as all the machines are identical in this case.

When the value of $\boldsymbol{\alpha_1 = Q}$, this implies we have **uniform parallel machines** in our scheduling problem . In this scenario, each machine j has a speed $s_j > 0$. Let us consider a job j here which has to be processed entirely on machine j, the total processing time for job i to be processed on machine j can be given as $p_{ij} = p_i / s_j$.

When the value of $\boldsymbol{\alpha_1 = R}$, then the machines on present in our scheduling problem imply **unrelated parallel machines** . These machines are different from uniform parallel machines as machines in this environment are not uniform and their relative performance on each job varies i.e. each of the speed of any machine in this environment depends on both the machine and the job to be processed. Thus, for a given job i and a machine j, time required to process job i on machine j $p_{ij} = p_i / s_{ij}$.

When the value of $\boldsymbol{\alpha_1 = PMPM}$ or $\boldsymbol{\alpha_1 = QMPM}$, this implies we are given with a set of multi - purpose machines[14] . In case of PMPM, the machines have identical speeds and when it is the case of QMPM, the multi - purpose machines would be having uniform speeds.

When the value of $\boldsymbol{\alpha_1 \epsilon \{ G, X, O, J, F \}}$,these kind of problems are called **Shop Problems.** In this shop environment, each job J_i would be in turn made up of operations $O_{i1},O_{i2},.....,O_{in}$ where each operation is required to be processed on any specific machine .The processing time of operations vary and can be zero. In addition, there can be a

precedence relation between the operations. This kind of a general model is called General shop and is represented by $\alpha_1 = \mathbf{G}$. All the other kinds of shop problems are a variation to the general shop problem.

2.3 SHOP PROBLEMS

2.3.1 Job Shop Problem

When the value of $\alpha_1 = \mathbf{J}$, the given problem is called a job shop scheduling problem. In this kind of problem, there is an order in which the operations have to be performed. An operation O_{in} cannot be performed unless all the predecessor operations in the order are completed.

2.3.2 Flow Shop Problem

A Flow shop problem is represented by $\alpha_1 = \mathbf{F}$. We can say that a flow shop problem is a special case of a job shop problem. In a flow shop problem, the order in which the operations are being processed is the same for all the jobs. But , all the jobs need not have the same processing times on a given machine i.e. different jobs may have different processing times on a same machine. If all the machines have the same job order, then we call it a permutation flow shop.

2.3.3 Open Shop Problem

An open shop problem can be represented by $\alpha_1 = \mathbf{O}$. An open shop problem is similar to a flow shop problem except that there is no constraint on all the jobs having the same order of operations i.e. operations belonging to a job can be processed in any order but no more than one operation of the same job is allowed to process at the same point of time[15].

2.3.4 Mixed Shop Problem

A mixed shop problem is represented by $\boldsymbol{\alpha_1 = X}$. In a mixed shop problem , some of the jobs may have a specified machine order as in a job shop and some of the jobs may have their operations performed in an arbitrary machine order as in an open shop .So ,we can say that a mixed shop is a combination of job shop and open shop.

The value of α2 in our machine environment represents the number of machines present in a given scenario. If the value of α2 is equal to some positive integer Z where Z = 1,2,3,4.....,then it implies that Z number of machines present. If α2 = k where k represents some arbitrary number , it implies some fixed number of machines available where as when α2 = o ,then the number of machines present is arbitrary .

2.4 JOB CHARACTERISTICS

The job characteristics of a schedule is represented by β which is made of six elements β_1, β_2, β_3, β_4, β_5 and β_6. Each of these elements specify a certain job character of the given schedule.

When $\boldsymbol{\beta = \beta_1}$, it indicates that **preemption** is allowed in the schedule which means that a job or a certain operation could be paused at any time and can be resumed later even on a different machine . A job or an operation can be preempted more than once . When preemption is allowed in a schedule, we set β_1 = pmtn .

When $\boldsymbol{\beta = \beta_2}$, it indicates that a **precedence** relation exists among the jobs which means that jobs can be dependent on each other .A given job J_m must be completed in order to start a job J_n and a job J_o would start processing only after job J_n completes. These precedence relations could be imagined in the form of an acyclic directed graph G = (V, E) where vertices V = {1,2,3,4,.....,n } are the jobs and edges E represents the precedence relation

where (m, n) ϵ E if and only if J_n starts only after completion of job J_m .This precedence is represented as $J_m \rightarrow J_n$. When there is a precedence relation among jobs, we represent β_2 = prec. There are more than one kind of precedence relations possible and are represented by chains, trees, an intree, an outtree, or a series parallel graph changing the β_2 representation accordingly.

If β_2 = tree , it means that G could either represent an intree or an outtree but if β_2 = intree , then G represents an intree . An intree is a rooted tree where the maximum out degree of any node in the tree is one. Thus an intree looks like it is directed towards the root .

If β_2 = outtree , then G represents an outtree . An outtree is a rooted tree where the maximum in degree is one .Thus, an outtree could look like it is directed away from root (like an inverted tree).

If β_2 = chains, then G represents a set of chains . A chain is a tree where in the out degree and the in degree can be at most one .

If β_2 = sp-graph, then G represents a series parallel graph . When the graph here is made up of only one vertex and is series parallel, it is called a Base graph. when we form a graph G = (V1 $\cup$ V2 , E1 $\cup$ E2) joining the vertices and edges of graphs G1 and G2 resulting in a series parallel graph, it is called a Parallel composition . When we form a series parallel graph G joining the vertices and edges of G1 and G2 and also joining the edges from the sinks of G1 to the sources of G2, it is called a Series composition.

When $\boldsymbol{\beta = \beta_3}$, then it indicates that each job may have a release time .This means that a particular job J_i having release time r_i = z, then job J_i would be available for processing after 'z' units of time. When there are release times specified for jobs on our schedule and represented by r_i then $\beta_3 = r_i$.

When $\boldsymbol{\beta = \beta_4}$, then we may be talking about a restriction on the processing times or the number of operations . When β_4 is equal to $p_{ij} = 1$, then it implies that all the jobs on the schedule are supposedly having a processing time of one unit . We may also see a case where $p_i = n$, implying that jobs have n processing time requirement where 'n' is a constant value.

When $\boldsymbol{\beta = \beta_5}$, then each of the jobs J_i may be having a deadline d_i .This means that the job J_i must be completed by the time d_i . When deadlines are specifies for the jobs, we represent $\beta_5 = d_i$.

When $\boldsymbol{\beta = \beta_6}$, it implies that batching in allowed in our schedule. A batch is nothing but a group of jobs processed together on a given machine . The finishing time of all job present in a given batch is equal to the finishing time of the batch . These batches require a set up time prior to getting started . The setup time for all the batches in a given schedule would be the same .

Batching problems are of two kinds : $\beta 6$ = p - batch problems and $\beta 6$ = s - batch problems .In a p-batching problem, the length of the batch is nothing but the maximum of the processing times all the jobs present in that given batch where as in a s-batching problem, the length of the batch is the sum of the processing times of all the jobs present in that given batch .

2.5 OPTIMALITY CRITERIA

Optimality criteria is represented by γ . Every scheduling problem is associated with a cost and our feasible schedule should try minimize the total cost for the schedule. Generally, the total cost function is of two types : Bottleneck objective and sum objectives . A bottle neck

objective function refers to minimizing the total time taken by a schedule to finish the last job where as the sum objective function refers to minimizing the total sum of completion times of all the jobs present on the schedule .

There can be many kinds of objective functions and most of them depend on attributes like completion times C_i, deadlines d_i of any given job J_i . Most commonly used objective functions are :

Makespan

This can be related to the bottleneck objective function and is given as :

$C_{max} = \text{Max} \{ C_i , \text{Where } i = 1, 2, \ldots\ldots, n \}$.

When Makespan is the objective function in our problem, we represent $\gamma = C_{max}$.

Total Weighted Flow Time

The total flow time is given as : $\sum C_i$

The total weighted flow time is given as : $\sum C_i w_i$ from $i = 1, 2, \ldots\ldots, n$.

Lateness

$L_i = C_i - d_i$.

Earliness

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

Tardiness

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

Absolute Deviation

$D_i = | C_i - d_i |$

Standard Deviation

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

Unit Penalty

$U_i = 0$ if $C_i <= d_i$;

$U_i = 1$ otherwise .

2.6 Shop Problems and Examples

2.6.1 General Shop Problem

We can define a general shop scheduling problem as follows . Consider we have *n* number of jobs where *i* = 1,.....,n and m number of machines M_1,.....,M_m . A job *i* be made of more than one operation O_{ij} where j = 1,.....,n_i with a processing time of P_{ij} . An operation O_{ij} is required to be processed on a particular machine μ_{ij} where $\mu_{ij} \in \{ M_1,.....,M_m \}$. Among these operations, there can exist a precedence relation . A job can only be processed on one machine at a point of time and a machine can process only one job at any point of time.

Our objective function here is to find a feasible schedule such that it minimizes the respective objective function given in the problem.

All the shop problems we discuss here are a special case of the general shop problem.

2.6.2 Job Shop Problem Example

The job shop problem is considered a special case of the general shop problem and the flow shop problem. Here we have a precedence constraint of the form $O_{ij} \rightarrow O_{i,j+1}$ where j = 1,.....n_i -1 .The machine order constraint is already defined in the problem as follows :

$J \mid\mid \sum C_i$

Job J1	$M_1(3)$	$M_2(2)$	$M_3(3)$
Job J2	$M_1(2)$	$M_3(2)$	$M_2(4)$
Job J3	$M_2(3)$	$M_1(1)$	$M_3(1)$

Table 2.1 An example problem illustrating job shop scheduling problem

In Table 2.1, consider job J_1. We have a precedence constraint such that operation O_{11} on M_1 should be processed before starting O_{12} on M_2 and so on. Likewise, we have the machine order defined for every job.

Using the rules of dispatching, we can construct a schedule minimizing the make span. Now, let us see how trying to optimize the make span effects the sum of completion times.

Here is a machine oriented Gantt Chart for the schedule constructed.(See Figure 2.1)

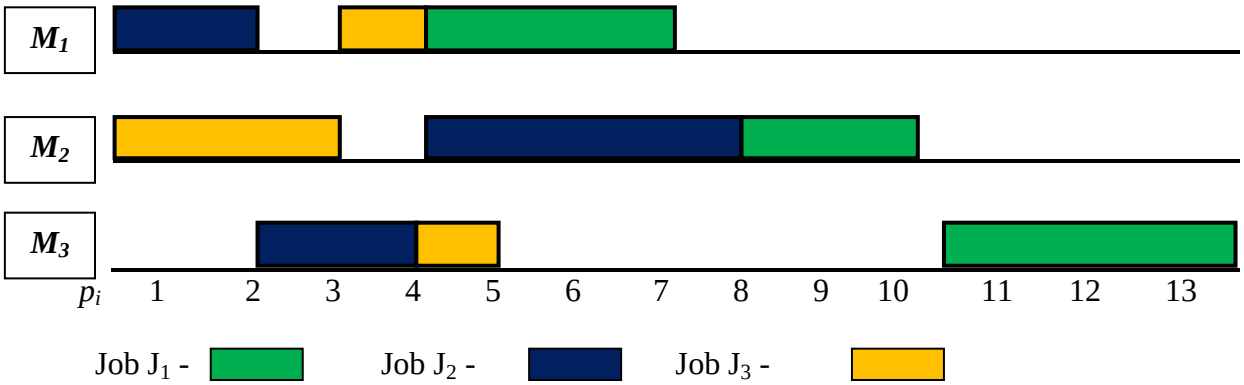


Figure 2.1 Gantt Chart for the Job Shop Problem minimizing Total Completion times.

Here, $\mathbf{C_{max}} = 13$; $\sum C_i = 5 + 8 + 13 = 26$.

Let us see if we can further optimize the make span and how it effects the completion time.

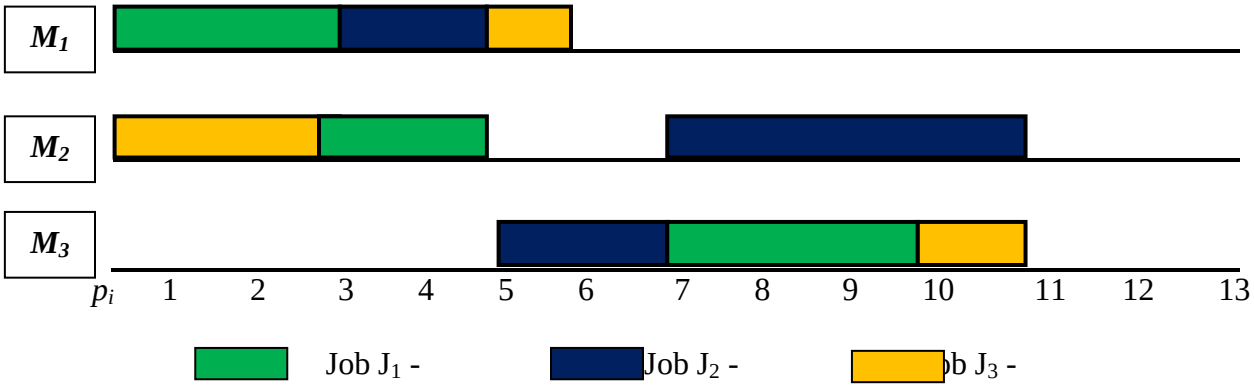


Figure 2.2 Gantt Chart for the Job Shop Problem minimizing the Make span.

Here $\mathbf{C_{max}} = 11$, $\sum C_i = 10 + 11 + 11 = 32$.

From Figure 2.2,it is clear that the make span is minimized from the previous schedule but the sum of completion times has increased drastically .This change may not be the same for every scheduling problem though.

2.6.3 Flow Shop Problem Example

A flow shop is a special case of the general shop where each job can consist of m operations and an operation O_{ij} is required to be processed only on M_j where j = 1,.....,m. There exists a precedence constraint of the form $O_{ij} \rightarrow O_{i,j+1}$ for each i = 1,.....,m. The only difference between the job shop and flow shop is that the operation number j here represents the machine on which O_{ij} is processed which means that in a flow shop, each job is processed first on machine 1 and then machine 2 and so on.

Let us consider the following flow shop scheduling problem. In a flow shop, the machine order remains the same for all the jobs as seen below. Given are the processing times of each operation belonging to a particular job and to be only processed on that particular machine.

	M_1	M_2	M_3
Job J1	2	2	3
Job J2	1	4	3
Job J3	4	1	4

Table 2.2 An example problem illustrating Flow shop scheduling problem

Now using basic rules of dispatching , let us come up with a schedule which minimizes the make span(see Figure 2.3).

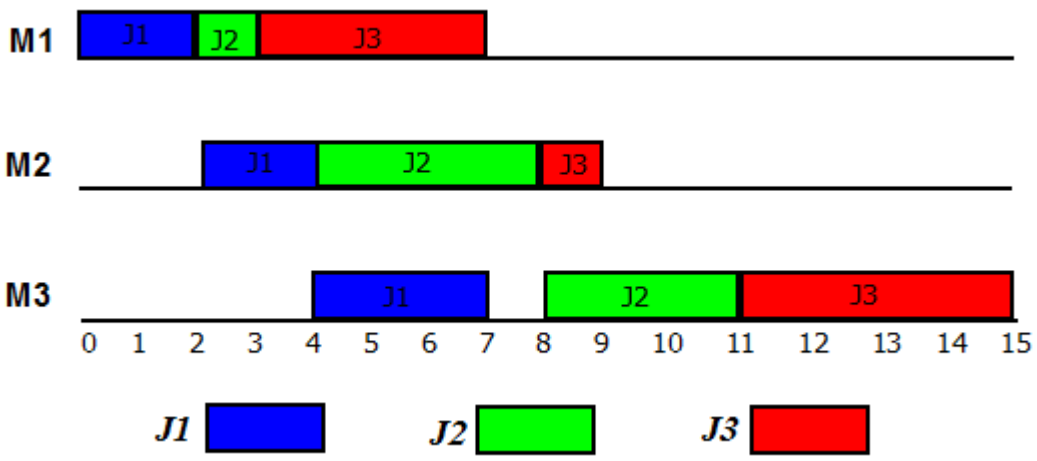


Figure 2.3 Gantt Chart 1 for the Flow shop problem.

Apparently from Figure 2.3,

C_{max} = 15 and

$\sum C_i = 7 + 9 + 15 = 31$.

Let us see if we could come up with a better schedule (See Figure 2.4)

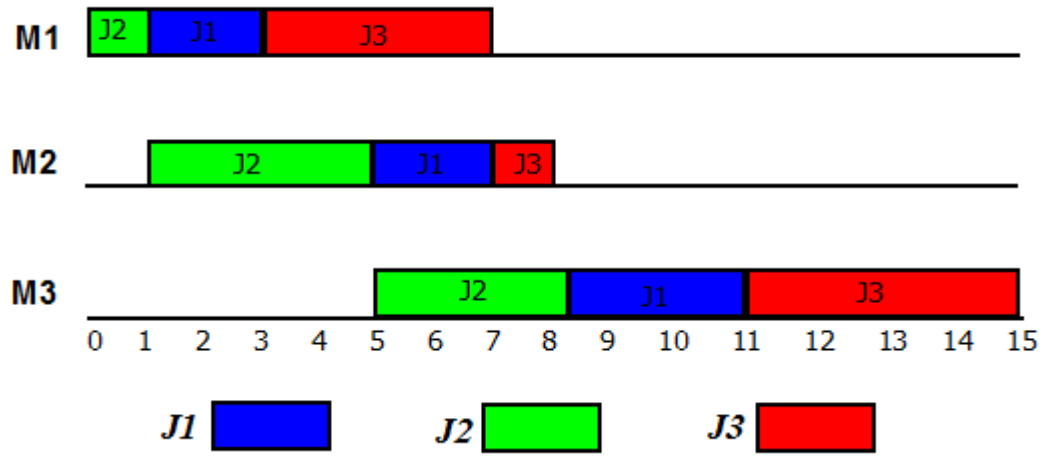


Figure 2.4 Gantt Chart 2 for the Flow Shop Problem

From Figure 2.4,

C_{max} = 15 and

$\sum C_i = 7 + 8 + 15 = 30$.

Apparently, there has not been a change in the make span from the previous schedule but the total completion time had been minimized.

2.6.4 Open Shop Scheduling Problem Example :

Open shop is a special case of the general shop problem where each job *i* can consist of m operations j = 1,.....,m and operation O_{ij} is to be processed on machine M_j . The operations can be processed in an arbitrary order unlike the job shop and the flow shops.

Let us consider the following open shop problem where the numbers represent the processing times of the operations on their respective machines.

	M_1	M_2	M_3
Job J1	1	2	3
Job J2	4	1	4
Job J3	0	4	5

Table 2.3 An example problem illustrating Open shop scheduling problem.

Using the rules of dispatching, let us try to construct a schedule such that we minimize the make span .

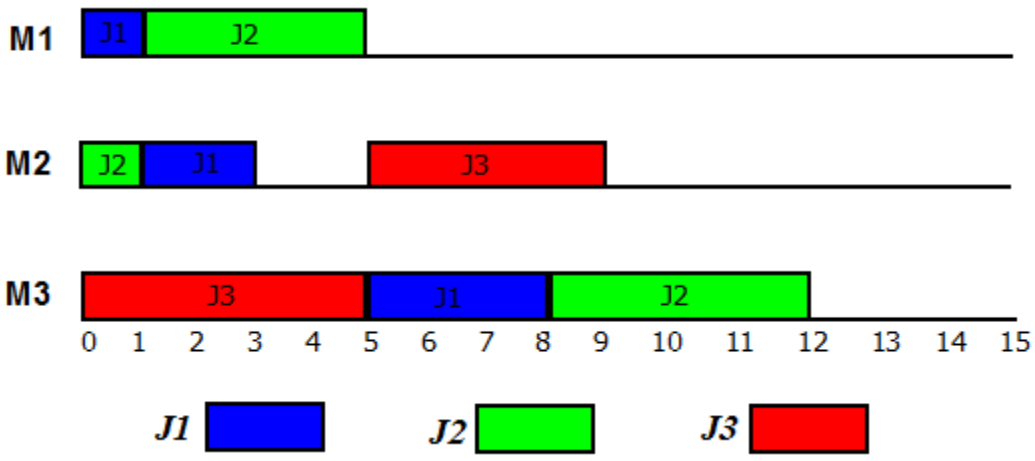


Figure 2.5 Machine Oriented Gantt Chart Minimizing the Make span.

Here,

C_{max} = 12 and

$\sum C_i = 5 + 9 + 12 = 26$.

In Figure 2.5, the make span is optimized as the completion time on machine 3 cannot be reduced any further.

2.6.5 Mixed Shop Problem

A mixed shop problem is a combination of the open shop and the job shop problem. It is denoted by X. So, a mixed shop consists of both job shop jobs and open shop jobs.

CHAPTER 3

Job Shop Problem : Using a Geometric Approach

3.1 Job Shop Scheduling Problem

Let us suppose we have n number of jobs to be processed where i = 1, 2, 3, ... , n and m number of machines M_1, M_2 , ... , M_m. Each job i is in turn made up of n_i number of operations $O_{i,1}$, $O_{i,2}$, $O_{i,3}$, ... , $O_{i,ni}$ where each job has an operation precedence constraint i.e. the operations should be processed in an order $O_{i,j} \rightarrow O_{i,j+1}$. This implies that operation $O_{i,j+1}$ is available for processing only after operation $O_{i,j}$ is completed. Each of these operations $O_{i,j}$ can be processed only on a particular machine M_m with a specific processing time $P_{i,j}$ requirement. The goal is to find a feasible schedule which optimizes the objective function specified in our problem.

3.2 Problem with two jobs : Geometric Approach

A job shop problem with two jobs with objective function C_{max} is polynomially solvable using the geometric approach. The geometric approach reduces the two job shop scheduling problem to a shortest path problem in a two dimensional plane with rectangular shaped objects present as obstacles[2] . We deal with the case where the objective function is Makespan.

J | n = 2 | C_{max}

Given below are the attributes for the problem to be solved :

Job J1	**M2(2)**	**M1(1)**	**M3(2)**	**M1(3)**
Job J2	M2(1)	M3(1)	M1(3)	M2(1)

Table 3.1 Example to demonstrate the geometric approach

In Table 3.1, first row represents all the operations belonging to job J_1 and the second row corresponds to all the operations belonging to job J_2.All the operations are represented by the machine number they are to be processed on along with the processing time requirement which is shown in the braces. As this is a job shop problem there is a precedence constraint which is implied in the figure in the order the operations are shown in each row i.e. in the first row operation $O_{1,1}$ which is to be processed on machine M2 comes before operation $O_{1,2}$ which is to be processed only on machine M1.

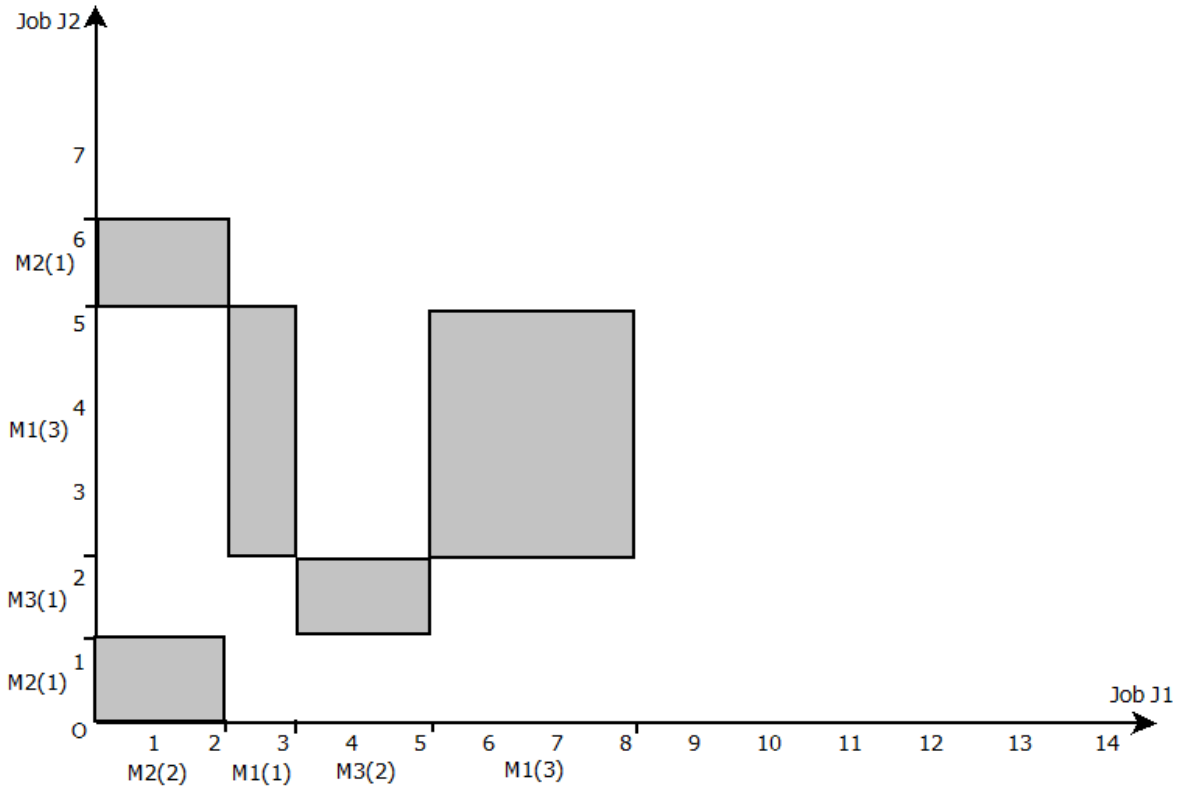


Figure 3.1 Graphical representation of a job shop problem with two jobs

Firstly, the problem is reduced to a shortest path problem in a two dimensional plane with rectangle objects acting as obstacles. Let us see how the graph can be developed so that we can represent our problem.

Each of the jobs are represented on one of the axis .As we would be having only two jobs in our case, we will be representing job J_1 on X- axis and job J_2 on Y-axis.

Now as each axis represents an axis on the graph, we would be representing the respective operations of each job on their respective axis in the order in which the operations are to be processed .(See Figure 3.1)

The processing times of these operations are represented by the intervals on their respective axis (J_1 - X axis ; J_2 - Y axis) .We will label each of these intervals with the machine number on which the respective operation is to be performed.

For each operation in job J_1, any operation in job J_2 shares a common machine, we represent that region with a rectangular box. Likewise, we do this for all the operations in job J_1 representing the operations sharing the same machine with rectangle obstacles.

Now, a feasible schedule corresponds to a path in the following graph starting from point O to point F.As we can notice in Figure 3.1, we could have more than one paths as we have more than one schedule for any given problem but there exists only one feasible schedule. Figure 3.2 represents all the paths possible in our graph. Each path is represented by a unique color.

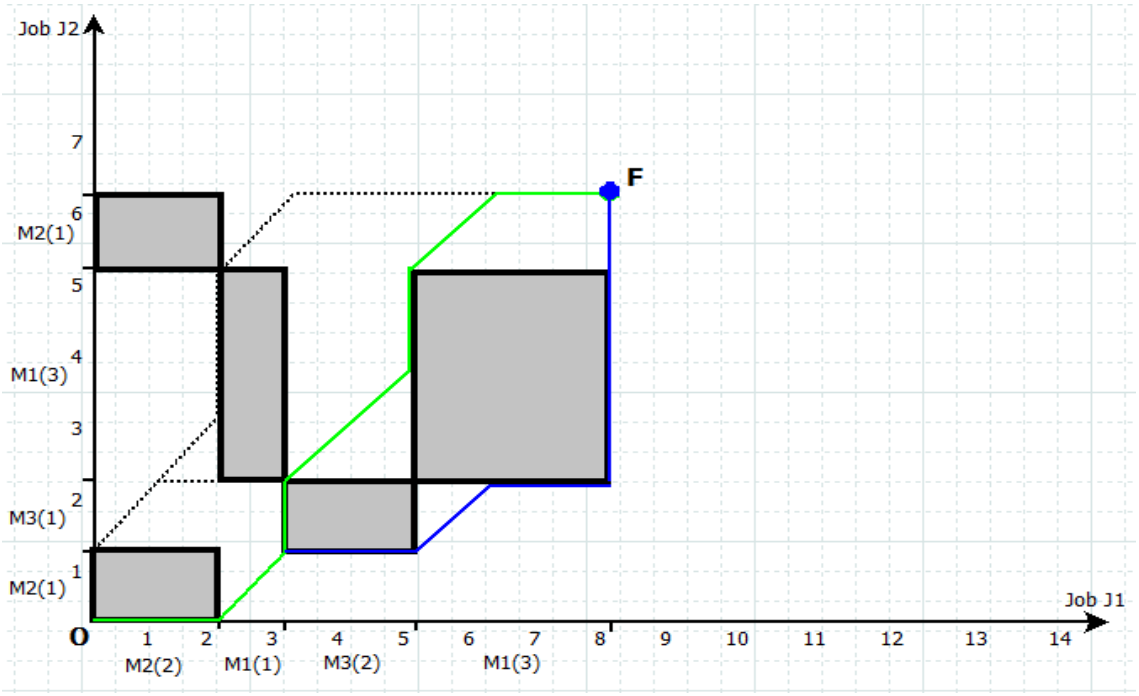


Figure 3.2 Graph representing possible paths

Each path shall be holding the following properties :

The path is made up of segments which can be either horizontal , progressing along X axis or vertical, progressing along Y axis or head diagonal along a 45 degree angle. A horizontal or vertical path implies that at a point of time only one job is being processed but a diagonal path indicates that both jobs are being processed at that point of time .

No path can go through any of the obstacles present in the diagram.If a path passes through a region in the obstacle, it implies that both jobs are being processed that too on the same machine which is not allowed .So,any path is not allowed to pass through the interiors of an obstacle.

The length of the path can be given as length of horizontalparts + length of vertical parts +length of diagnol parts .Length of the horizontal parts(or vertical) can be noticed along the X-axis(Y-axis) where as length of the diagnolpart is equal to (length of dianol part)/√2.

We can have more than one path from O to F but only the shortest path among them is considered as a optima schedule.

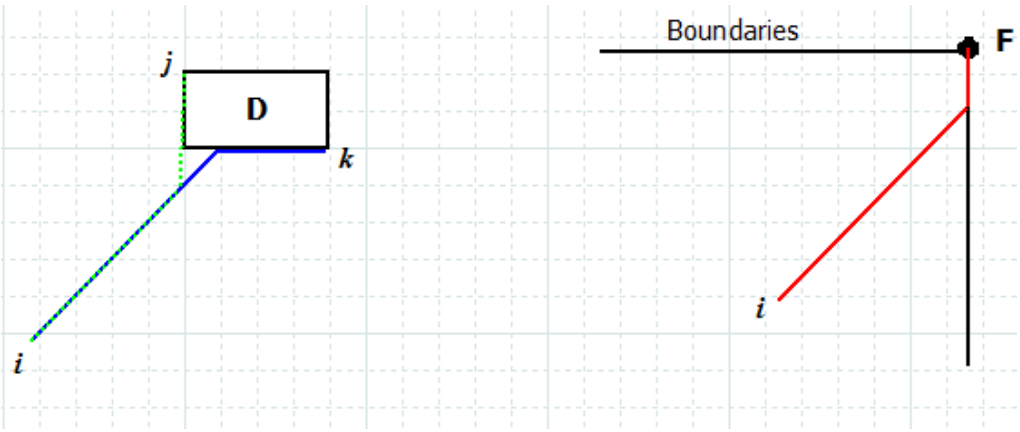
The next step is to construct our network N = (V, A, d) where V represents the set of vertices, A represents the set of arcs and d is the set of distances. This network can be constructed as follows :

Firstly, we recognize the set of vertices V . V consists of points O, F and all the north-west and south-east corners of all the obstacles present in our graph. O is considered as a degenerate point where the south-east corner and the north west-corner meet.

To construct the set of arcs A, consider any vertex i ϵ V / {F} i.e. any vertex from the set V except point F . We go diagonally at 45^0 from this point in the north-east direction. Any vertex *i* can at most have two successors We would be having two scenarios here.

In the first case (See Figure 3.3(*b*)), the only successor to point *i* is F.The path (*i*, *F*) makes the arc here and consists of the path from point i to the boundary and along the boundary to the point *F*.

In the second case(See Figure 3.3(*a*)), the path encounters a obstacle D instead of the boundaries resulting in two successors to the point *i*. The path goes from point i to the obstacle's north-west corner or its south-east corner resulting in two arcs (*i, j*) and (*i, k*) where *j* is the north-west corner and '*k*' is the south east corner of obstacle D. So, point *i* will be having two successors - the north-east corner and south-west corner of obstacle D .(as shown in Figure 3(a)).



(*a*) (*b*)

Figure 3.3 Possible Successors of a Vertex *i* .

Now, in the process of constructing our graph, the next step is to order the obstacles .This ordering is done according to the lexicographic order of the north-west corners . Let us suppose we have obstacles D_i and D_j whose north-west corners have co-ordinates (x_i , y_i) and (x_j , y_j) . Then ,

if $y_i < y_j$ then $D_i < D_j$

else if $y_i = y_j$ and $x_i < x_j$ then $D_i < D_j$

else

$D_j < D_i$.

Likewise, if the total number of restricted regions are r .Then , the ordering or these obstacles would look as follows :

$D_1 < D_2 < D_3 < \ldots\ldots\ldots\ldots < D_r$.

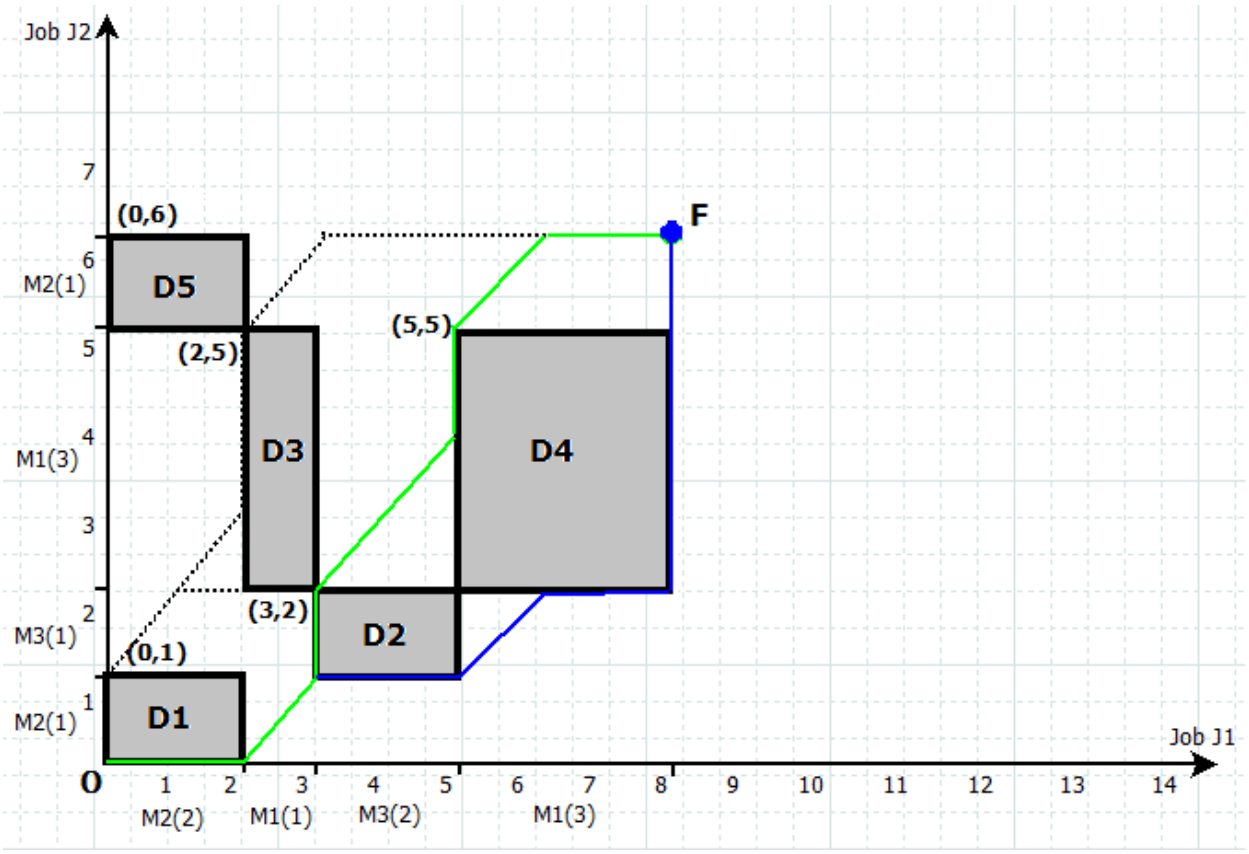


Figure 3.4 Graph after ordering the obstacles.

3.2.1 Constructing the Network

We will be using a sweep line to construct our network N = G(V , A , d) where V is the set of vertices, A set of arcs and d is the set of distances. The resulting network will be a weighted acyclic directed graph. This network N can be constructed in *O(n log n)* steps where *n* is the number of obstacles in the plane and the shortest path in the network N can be found in *O(n)* time[3] . Let us see the steps in constructing this graph.

Firstly, a sweep is applied across the graph using a north-east south-west directed line . This line is moved from north-west to south-east direction parallel to itself. This line equation can be given as $y - x = c$.

The set S holds the set of restricted regions which are currently in contact with the sweep line along with the intervals imposed by them on the sweep line . The set S is updated whenever changes occur during the sweep i.e. when the sweep line encounters a north-west corner or a south-east corner of an obstacle.

When the sweep line encounters a north-west corner '*x*' of an obstacle D_i , D_i is inserted into S. If S holds any other element D_j with NW corner ' *k* ' and SE corner ' *l* ' which is next to D_i in the lexicographic order, then (*x* , *k*) and (*x* , *l*) are inserted into A and the distances d(*x* , *k*) and d(*x* , *l*) are calculated. If no such element exists is S, then (*x*,*F*) is inserted into A and the distance d(*x*,*F*) is calculated.

When the sweep line encounters a south-east corner '*y*' of any obstacle D_i ,then the set S is checked for any element D_j with NW corner '*k*' and SE corner '*l*' that is next to D_i in lexicographic order. If such an element exists, then (*y*,*k*) and (*y*,*l*) are inserted into A and the distances d(*y*,*k*) and d(*y*,*l*) are calculated. If no such element exists in S, then (*y*,*F*) is inserted into A and distance d(*y*,*F*) is calculated. Finally, D_i is removed from S.

Let us see how we can construct a graph network for our problem using the sweep line (See Figure 3.5) .

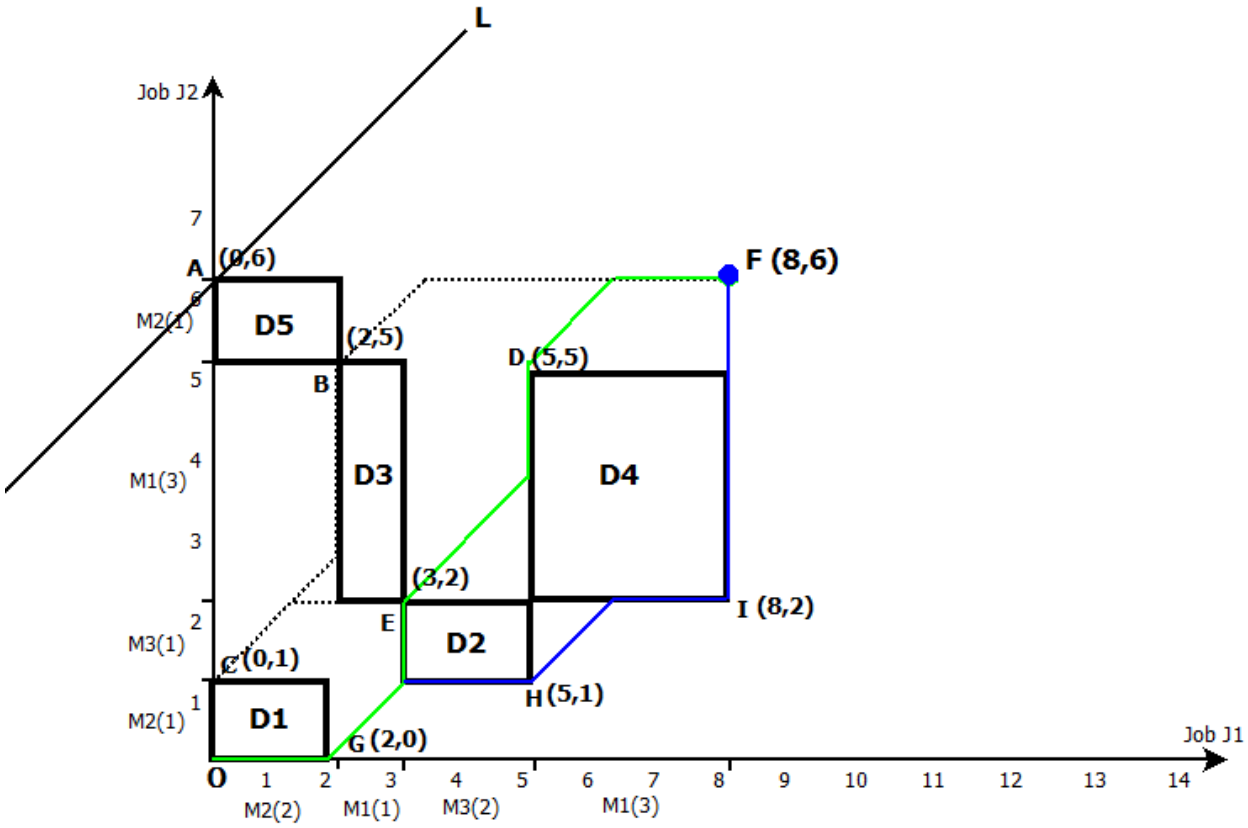


Figure 3.5 Graph 1 with Sweep Line for constructing the Network N.

Let us consider we have the set S as a bag to hold elements and a sweep line L that helps us construct our graph. All the changes in the sweep and the bag are noted. Let us see how the set S and A would look like for every change that occurs during the sweep .The graph is constructed as follows :

Step 1 :

Initially, the sweep line(L) $y - x = c$ intersects the north-west corner A of the obstacle D_5.(Figure 5(a)). As set S contains only D_5 , (A,F) is inserted into A and the distance d(A,F) is calculated. Apparently from Figure 5(*a*), d(A,F) = 8 .

D_5				

Step 2 :

The sweep line encounters the south east corner of D_5 or the north west corner of D_3.

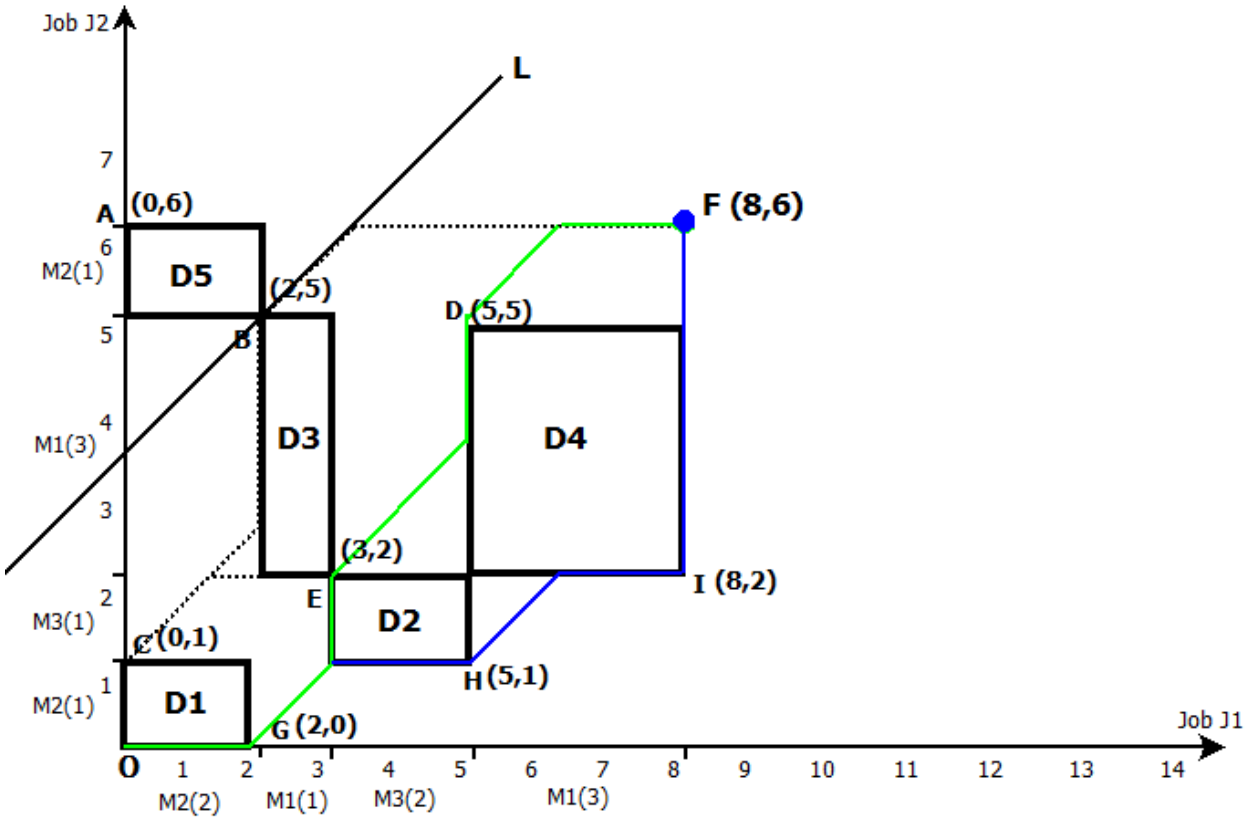


Figure 3.6 Graph 2 with Sweep Line for constructing the Network N.

Figure 3.6 Graph representing different stages for constructing the Network N.

As the sweep line hits the south east corner of obstacle D_5 , the set S is checked for any elements next to D_5 in lexicographic order. As S now holds only D_5 (B,F) is inserted into A and the distance d(B, F) is calculated. Apparently from Figure 5(b),d(B, F) = 6.Now, D_5 is removed from S.

As B is also the north-west corner of D_3,D_3 is inserted into S which is currently empty. Arc (B,F) and its distance has been already calculated and inserted into A.

D_3				

Step 3:

The sweep line intersects the north-west corner C of obstacle D_1 .

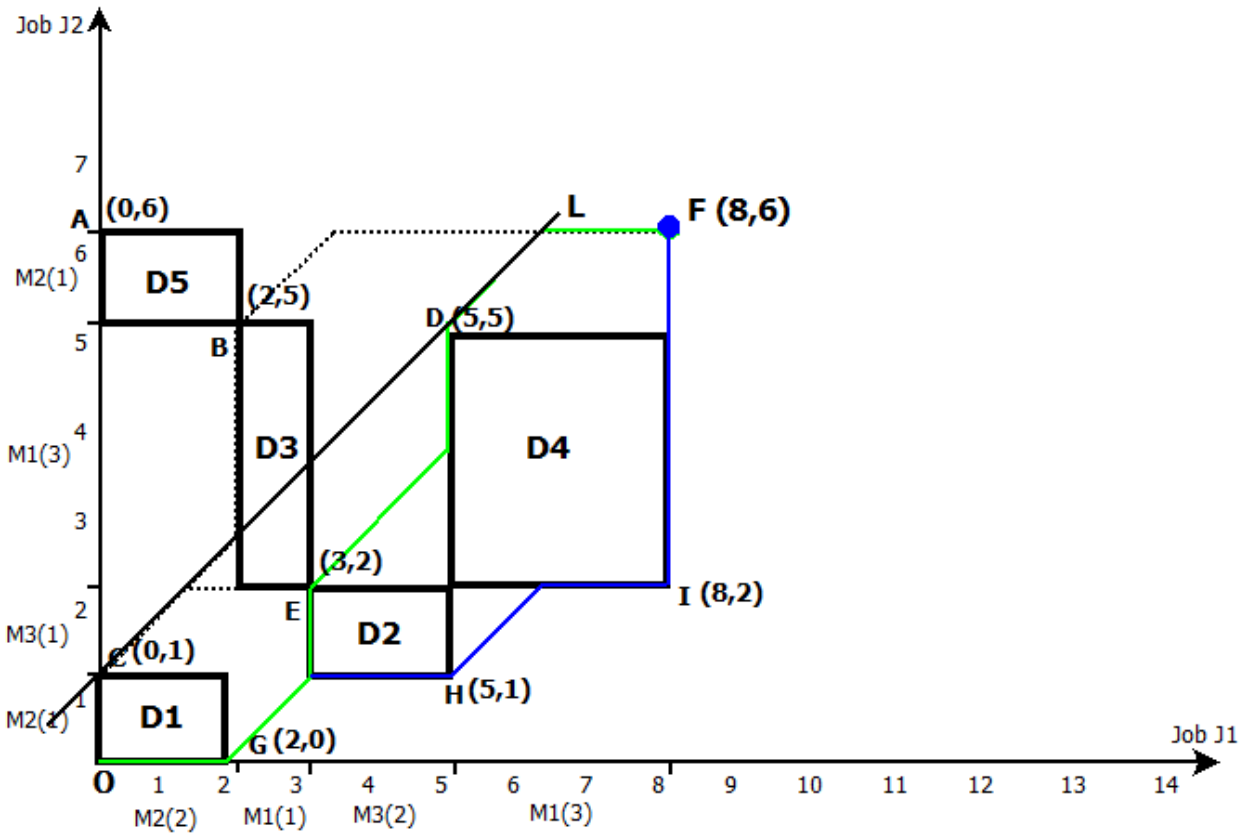


Figure 3.7 Graph 3 with Sweep Line for constructing the Network N.

Now, the sweep line is in contact with the north west corner of obstacle D_1 .So,D_1 inserted into the set S. Now, S is checked for any elements which are next to D_1 in lexicographic order. As, we can see S currently holds D_3 other than D_1. D_3 is next to D_1.So,arcs (C, B) and (C, E) are inserted into A and the distances d(C, B) and d(C, E) are noted.

D_3	**D_1**			

Step 4 :

Sweep line L is in contact with the north-west corner D of the restricted region D_4.

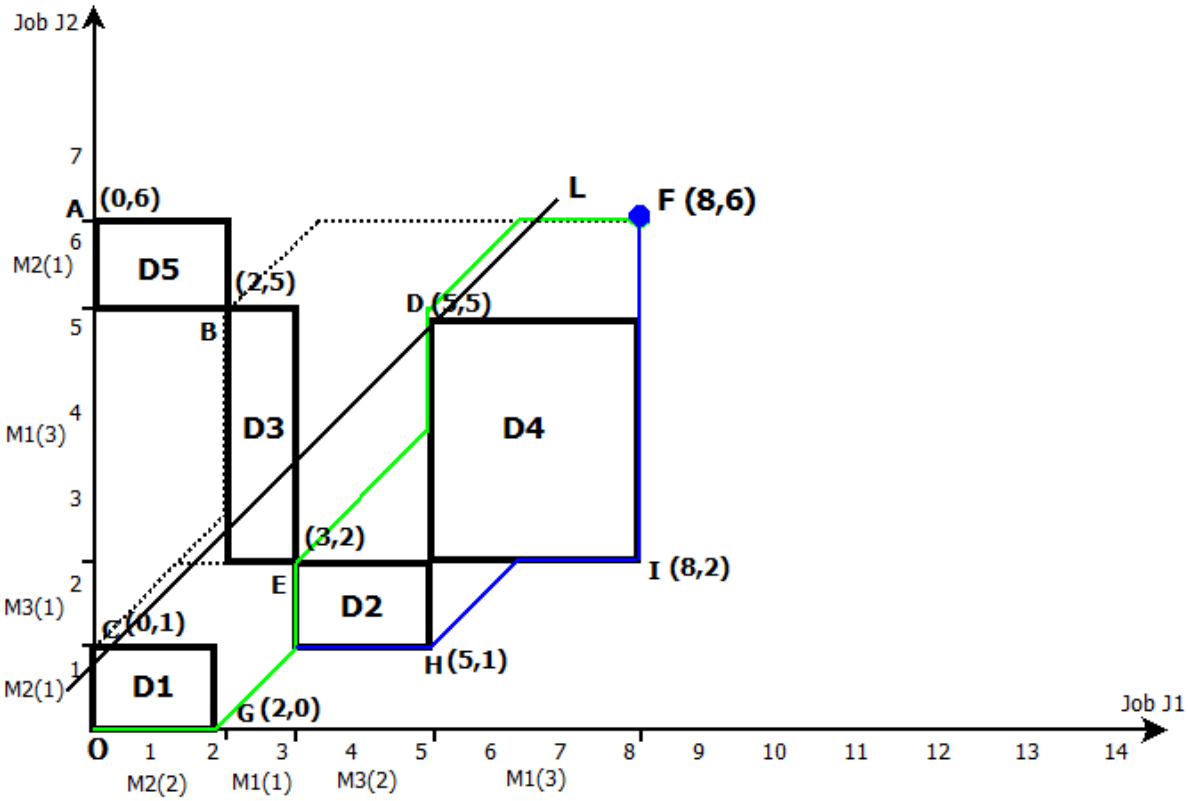


Figure 3.8 Graph 4 with Sweep Line for constructing the Network N.

As L is in contact with the NW corner of D_4, it is inserted into S.

D_3	**D_1**	**D_4**		

As we can see S does not hold any element next to D_4 in lexicographic order .So, arc (D, F) is inserted into A and the distance d(D, F) is noted.

Step 5:

Sweep line L comes into contact with the vertex O .

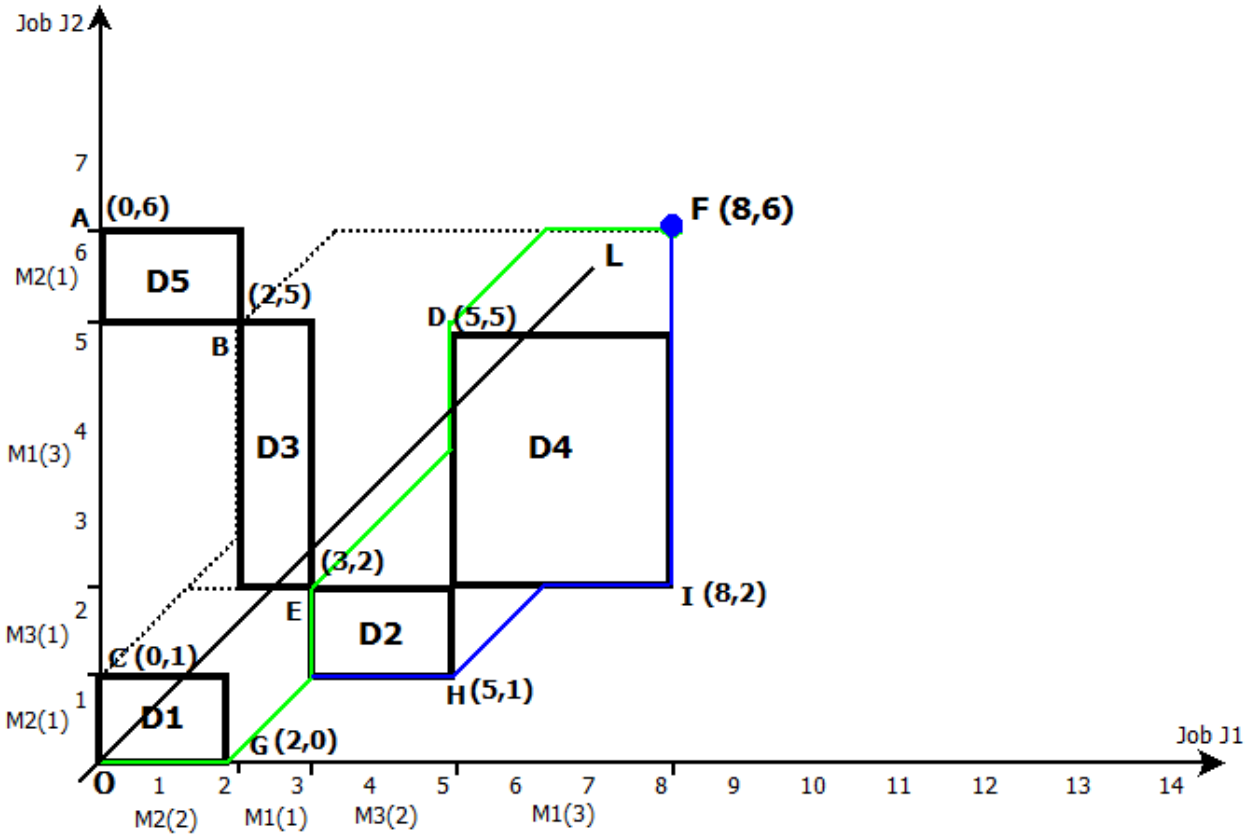


Figure 3.9 Graph 5 with Sweep Line for constructing the Network N.

As we have said before ,O is a degenerate point where the NE and SW vertices meet. This can be seen as a special case. Apparently, from the graph, if we consider it as a vertex belonging to region D_i, then D_1 is the next element to it in S .So arcs (O, C) and (O, G) are inserted into A and the distances d(O, C) and d(O, G) are noted. Finally ,this region is deleted from the set S.

Similarly when the sweep line hits F, no changes would occur in S and A as F is considered the last point in our graph and there would be no elements next to it belonging to the set V.

Step 6 :

Sweep line L is in contact with the SE corner of D_3 or the NW corner of D_2 E.

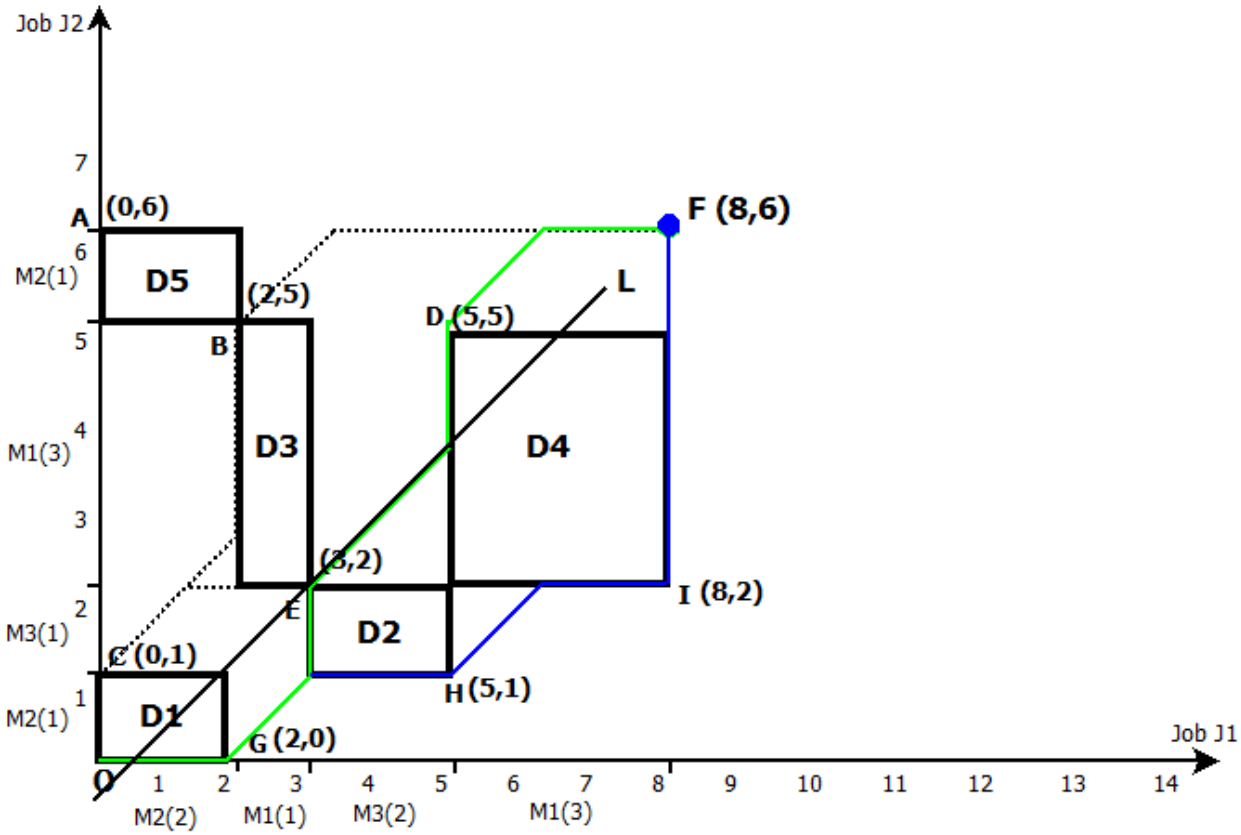


Figure 3.10 Graph 6 with Sweep Line for constructing the Network N.

As L is in contact with the SE corner of D_3,we check for any elements next to D_3. As S contains D_4 which is higher in lexicographic order than D_3, arcs (E, D) and (E, I) are inserted into A and the distances d(E, D) and d(E, I) are noted. Now, $D_3$is removed from S. As E is also the NW corner of D_2 ,it is inserted into S.

D_1	D_4	D_2		

D_4 is the next element to D_2 in the set S. Apparently, from the Figure the NW and SE corners of D_4 are denoted as D and I respectively. So, arcs (E, D) and (E, I) are inserted into A and their respective distances are noted.

Step 7:

The Sweep line L is in contact with the SE corner of D_1.

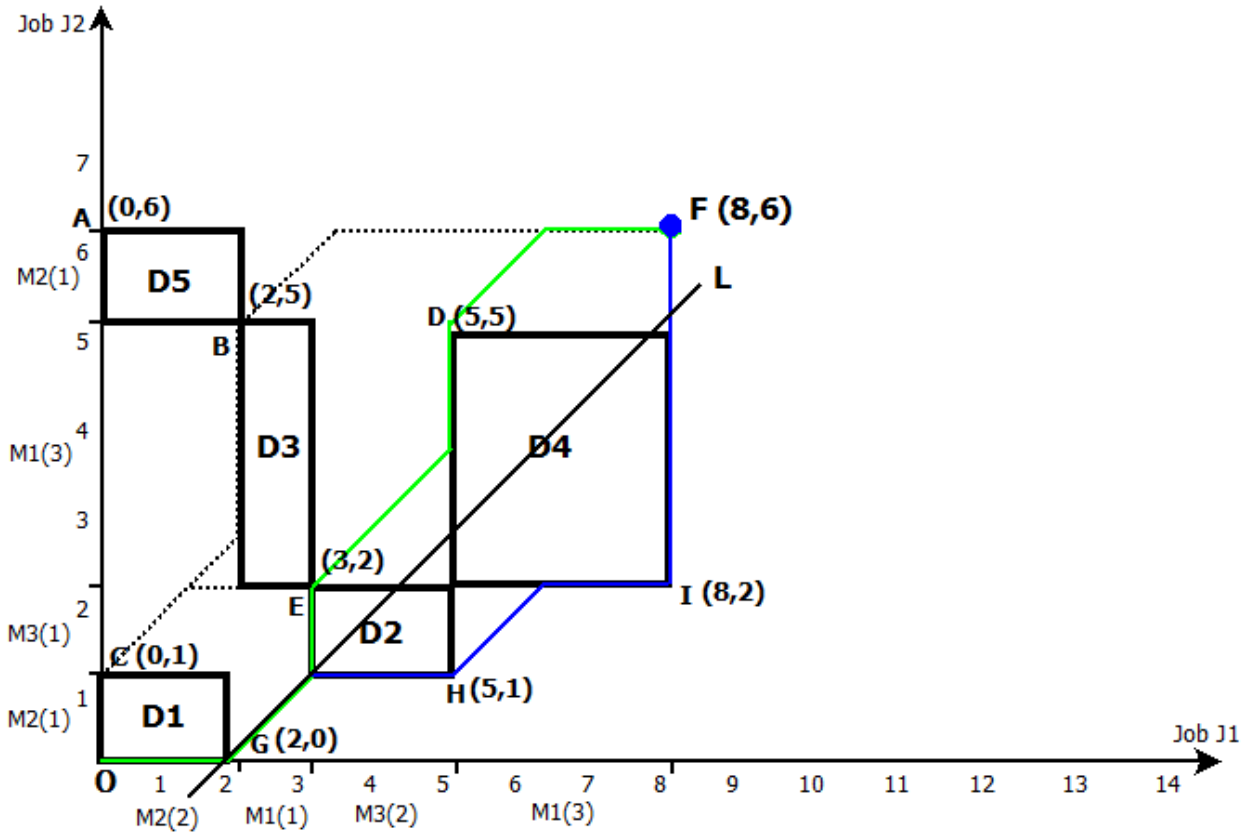


Figure 3.11 Graph 7 with Sweep Line for constructing the Network N.

As L is in contact with G,SE corner of D_1 set S is checked for any elements next to D_1 .Here we have D_2 and D_4 which are higher than D_1 in lexicographic order but D_2 comes first .So, D_2 (with NE corner E and SE corner H) is considered as the next element to D_1. So, arcs (G,

E) and (G, H) are inserted into A and their respective distances are noted.Finally,D_1 is removed from S.

D_4	**D_2**			

Step 8 :

Sweep line L is in contact with the SE corner of D_2.

As the sweep line hits the south east corner of obstacle D_2 , the set S is checked for any elements next to D_2 in lexicographic order.

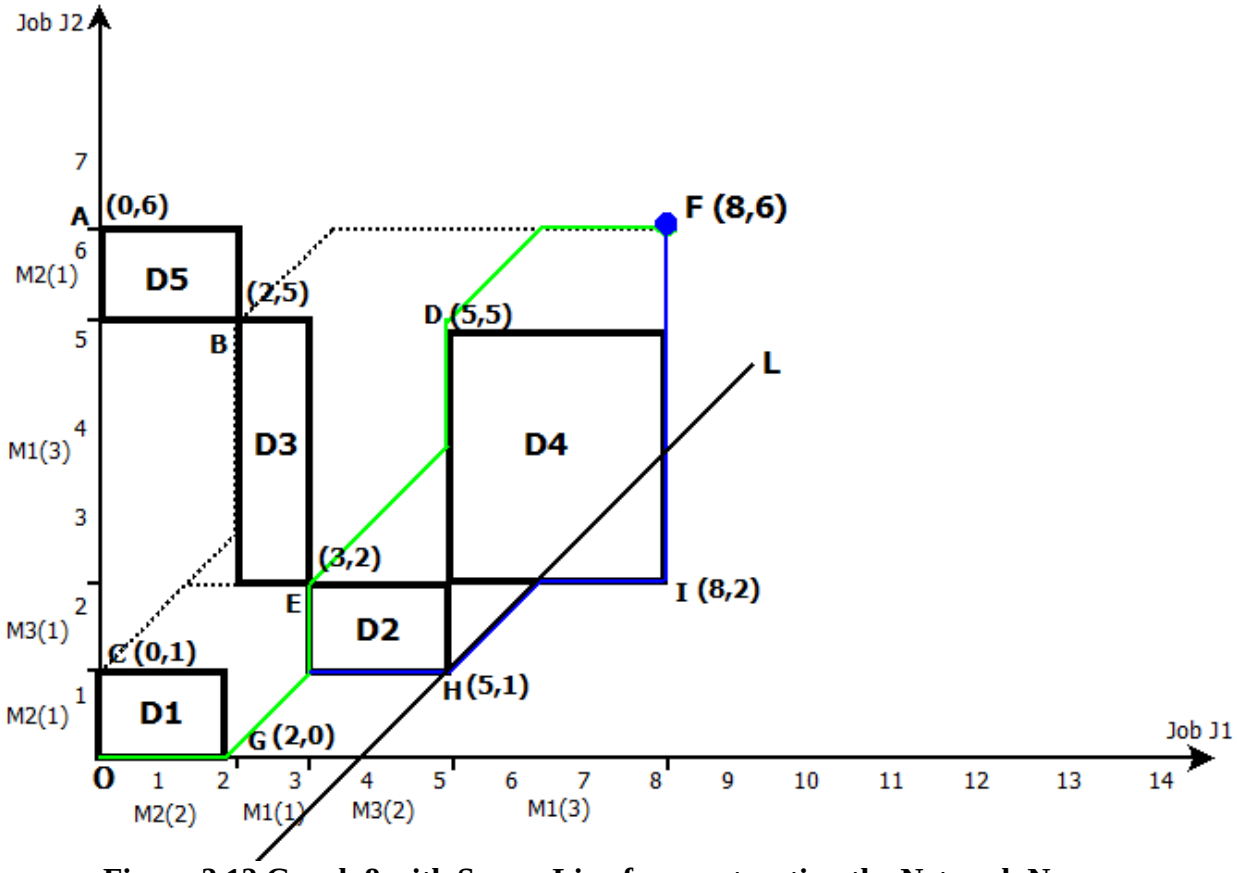


Figure 3.12 Graph 8 with Sweep Line for constructing the Network N.

As S now holds only D_4 which is next to D_2 , arcs (H, D) and (H, I) are inserted into A and the distances d(H, D) and d(H, I) are calculated. Now, D_2 is removed from S.

D_4				

Step 9 :

Now the sweep line is in contact with the SE corner I of D_4.

As we can see S currently holds no other element other than D_4. So, arc (I, F) is inserted into A and its distance d(I, F) is calculated and noted.Finally,D_4 is removed from S making it an empty set.

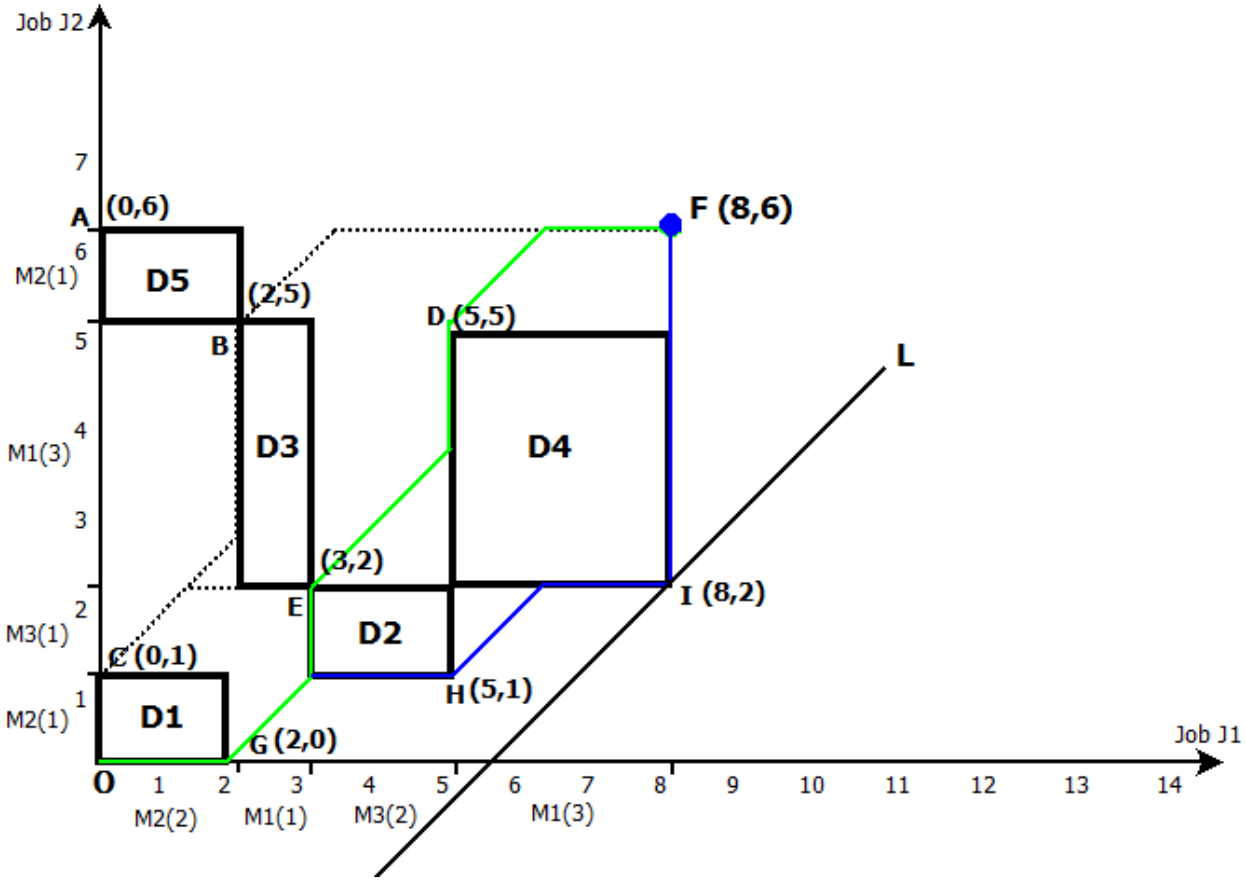


Figure 3.13 Graph 9 with Sweep Line for constructing the Network N.

The Sweep line L has swept the whole region where, changes occur when it comes into contact with any i ϵ V/ { F } creating a set of arcs A and their respective distances d constructing our graph G (V, A, d) .

Finally , let us see what elements we have in V, A and d :

V = { O, A, B, C, D, E, G, H, I }

A = { (A, F), (B, F), (C, B), (C, E), (D, F), (O, C), (O, G), (E, D), (E, I), (G, E), (G, H), (H, D), (H, I), (I,F) }

d = { (A, F) = 8, (B, F) = 6, (C, B) = 4, (C, E)= 3, (D, F) = 3 , (O, C) = 1, (O, G) = 2, (E, D) = 3, (E, I) = 5, (G, E) = 2, (G, H) = 3, (H, D) = 4, (H, I) = 3, (I,F) = 4. }

Finally, the resulting acyclic weighted directed graph for our problem is as follows :

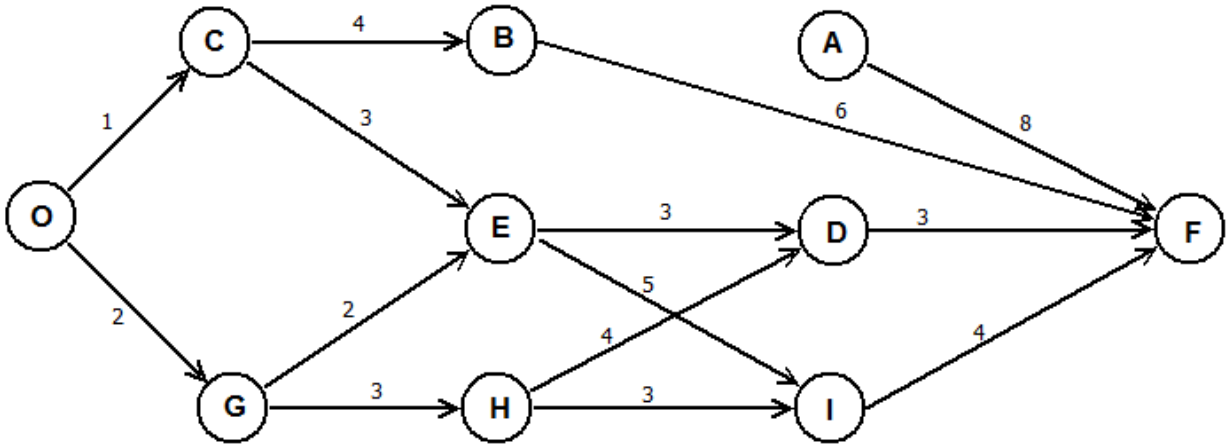


Figure 3.14 Network N Constructed from the job shop problem.

On Observing Figure 3.14, I found the following shortest paths from O to F :

Path 1 :

O → G → E → D → F .

The total weight of this path would be :

2 + 2 + 3 + 3 = 10.

Path 2 :

O → C → E → D → F .

The total weight of this path :

1 + 3 + 3 + 3 = 10.

Let us see the corresponding schedules for the paths mentioned above

Schedule for Path 1 :

The Gantt chart for the schedule according to path 1 looks as follows .

Apparently, here

C_{max} = 10

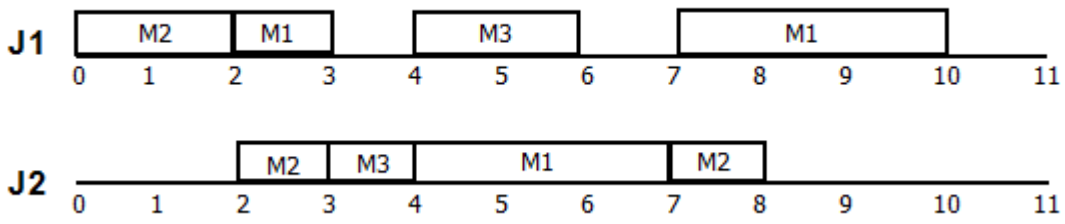


Figure 3.15 Job Oriented Gantt Chart for path 1.

Schedule for path 2 :

The schedule corresponding to path 2 looks as follows :

Here, apparently Makespan is equal to 10

C_{max} = 10.

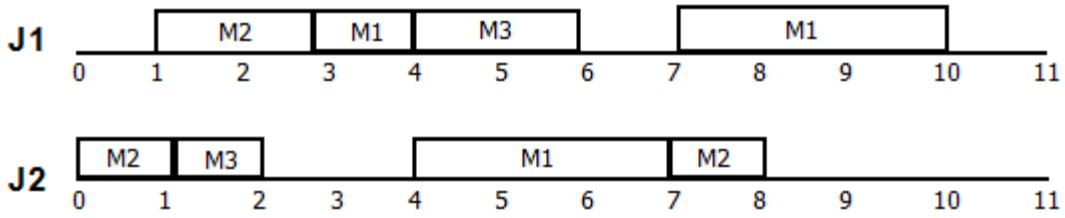


Figure 3.16 Job Oriented Gantt Chart for path 2.

CHAPTER 4

Solving Job Shop Problem with a limited number of machines

Among the shop problems, Job shop scheduling problems are considered the most popular and difficult ones to solve. In this Chapter, let us have a look at the job shop problems which can be solved in polynomial time when the number of machines are limited.

4.1 Job shop problem with two machines

Among some of the only few job shop problems which can be solved in polynomial or pseudo polynomial time, a two machine job shop problem with at most two operations per job problem is one among them. We discuss this algorithm in the following section 4.1.1.

4.1.1 $J2 \mid n_i \leq 2 \mid C_{max}$

This problem is solved by reducing it into a two-machine flow shop problem which is solved using the Johnson's algorithm[6] (Refer to section 5.1).

Let us see how this problem is solved following the steps below:

All the jobs in our problem are divided into the following subsets:

$\mathbf{I_1}$ – Set of all the jobs which are executed only on machine 1.

$\mathbf{I_2}$ – Set of all the jobs which are executed only on machine 2.

$\mathbf{I_{1,2}}$ – Set of all the jobs which are executed on machine 1 and then on machine 2.

$\mathbf{I_{2,1}}$ – Set of all the jobs which are executed on machine 2 and then on machine 1.

For the set of jobs in $I_{1,2}$, find the optimal sequence $R_{1,2}$. This is going to be a two machine flow shop problem which can be solved using Johnson's algorithm.

Similarly, for the set of jobs in $I_{2,1}$, find the optimal sequence $R_{2,1}$ .This is nothing but a two machine flow shop problem which can be solved using Johnson's algorithm.

Now on machine one ,schedule jobs belonging to $I_{1,2}$ according to the sequence $R_{1,2}$. Then schedule jobs belonging to I_1 in an arbitrary order. Now schedule jobs belonging to $I_{2,1}$ according to the sequence $R_{2,1}$.

Now on machine two , schedule jobs belonging to $I_{2,1}$ according to the sequence $R_{2,1}$. Now schedule jobs belonging to I_2 arbitrarily. Lastly schedule jobs belonging to $I_{1,2}$ according to the sequence $R_{1,2}$.

A schedule is said to be an **active** schedule if we cannot perform any of those jobs earlier without violating scheduling constraints.

Let us consider the resulting schedule from the above algorithm is an active schedule. Then, there would be at least one machine processing all the jobs without any idle time.

Let us say $\sum_{i \,\epsilon\, I2,1} Pi, 2 \leq \sum_{i \,\in I1,2} Pi1 + \sum_{i \in I1} Pi1$

i.e. sum of processing times of all the jobs belonging to $I_{2,1}$ on machine two is less than or equal to sum of processing times of all the jobs belonging to $I_{1,2}$ on machine one plus sum of all the processing times of all the jobs belonging to I_1 on machine one. Then, there will be no idle time on machine one. Otherwise, there would be no idle time on machine two.

4.1.2 J2 | N=K | C_{max}

In the previous section , we have shown that a two machine job shop problem can be solved when the number of operations are fixed to at most two operations. It has been proved that J2 || C_{max} problem is NP-hard[23].But, the two machine job shop problem where the number

of jobs is fixed to a constant *k* has been solved in polynomial time even when the machines are repeated in a job i.e. $\mu_{i,j} = \mu_{i,j+1}$ for a given job *i*. In this algorithm, the problem J2 | N = *k* | C_{max} is reduced to a shortest path problem in an acyclic network consisting of $O(r^k)$ vertices where r is the maximum length of a job (number of operations) i.e.

$$r = \max_{i=1}^{n} n_i .$$

The time complexity for constructing this network is equal to $O(r^{2k})$ steps . Thus, the total time complexity for solving this two machine job shop problem is $O(r^{2k})$ [4] . However, the three machine job shop problem $J3||\ n = 3 \mid C_{max}$ and $J3||\ n = 3 \mid C_{max}$ with $k = 3$ is proven to be NP-hard[5].

CHAPTER 5

Dealing with Flow Shop Problem

5.1 Johnsons Algorithm for solving two machine flow shop problem

F2 || C_{max}

This is one among the very few flow shop problems and the only one with C_{max} criterion which can be solved in polynomial time .An algorithm has been developed by Johnson[6] which can solve the two machine flow shop problem with C_{max} criterion in polynomial time. Let us consider the following two machine problem and solve it using the Johnson's algorithm .

	P_{i1}	P_{i2}
J_1	2	5
J_2	4	3
J_3	1	6
J_4	5	3

Table 5.1 An example problem illustrating two machine Flow shop scheduling problem using Johnson's Algorithm.

Algorithm 5.1.1 Johnson's Algorithm Pseudo code

Algorithm F2 || C_{max}

X : = {1,2,.....,n}; T : = Φ ; R : = Φ ;

While X ≠ Φ DO

BEGIN

Find i^* , j^* with $P_{i^*j^*}$ = min { $P_{i,j}$ | i ϵ X ; j = 1, 2 };

If j^* = 1 THEN T : = T o i^* ELSE R : = i^* o R ;

X = X\{ i^*}

END;

L: = T o R

Firstly, we are given a flow shop scheduling problem with two machines and any number of jobs with arbitrary processing times. Let X be the set of all the processing times P_{ij} and set T and R initialized to null .

Select job i* with least processing time. If two jobs have minimum processing times , select any one of the jobs randomly. If the selected processing time belongs to an operation on machine 1(j = 1),then insert that job i* to the end of the list T. Otherwise, insert the corresponding job to the beginning of the list R. Now, remove the job i* from the list X .

We repeat the above step until there are no jobs left in the list X.

Now , we have a final list L which is formed with combining list T and R .

L = T o R

This list L is nothing but our optimal sequence of jobs on both machines 1 and 2 .

Applying these steps to our problem , we would get

Set T = { 3 , 1 }

I = { 4 , 2 }.

Therefore , L = T o R

i.e. L = { 3 , 1 , 4 , 2 } .

Let us construct a Gantt chart based on the above sequence and see what the optimum Makespan is.

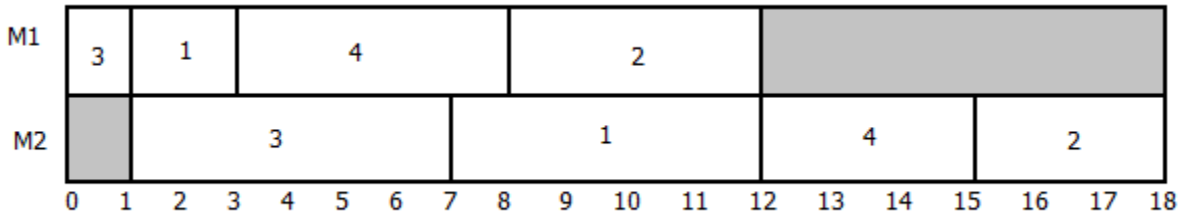


Figure 5.1 Gantt Chart representing the schedule obtained from Johnson's Algorithm.

Here $C_{max} = 18$.

According to Lemma 6.8,it states that For the problem $F_m \mid\mid C_{max}$ an optimal schedule exists with the following properties :

The job sequence on the first two machines is the same .

The job sequence on the last two machines is the same.

In a flow shop , if the optimal or final schedule has job sequences $\pi_1 , \pi_2 , \ldots , \pi_m$ where $\pi_1 = \pi_2 = \ldots = \pi_m$, then it is called a permutation flow shop . The above Lemma states that when we are presented with at most three machines ,the optimal solution is equal to that of the corresponding permutation flow shop. This does not apply when we have more than three machines in the flow shop.

5.2 Lemma 1 :

Let L : = L(1) , ... , L(n) be a list constructed by Algorithm $F2 \mid\mid C_{max}$. Then

$\min \{P_{i1} , P_{j2} \} < \min \{ P_{j1} , P_{i2} \}$

implies that job i appears before job j in L .

Proof :

Let us assume that $P_{i1} < \min \{ P_{j1} , P_{i2} \}$,

then it means that $P_{i1} < P_{i2}$.

This would mean that according to Johnson's Algorithm , job i is added to the end of the list T.

Now, let us see what are the possible cases of dealing with job j.

If job j is added to R, then apparently it comes after job i which is in the set T proving that job j comes after job i .

Else if job j appears in T , it would be after job i .This is because $P_{i1} < P_{j1}$ .So, job j is performed after job i when $\min \{P_{i1} , P_{j2} \} < \min \{ P_{j1} , P_{i2} \}$.

Similarly,

If $P_{j2} < \min \{ P_{j1} , P_{i2} \}$, it would belong to set R which makes P_{i1} the next minimum value and to be inserted into set T.

Therefore, job j appears after job i is done in L.

5.3 Lemma 2 :

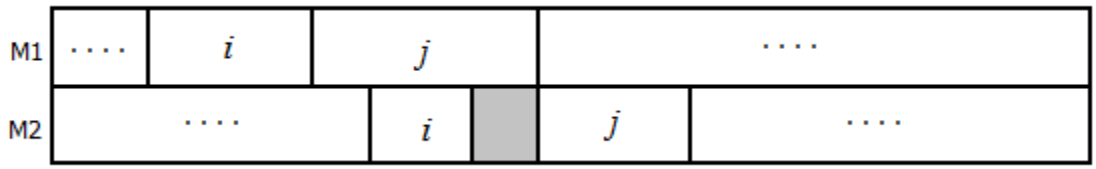
Consider a schedule in which job j is scheduled immediately after job i . Then

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

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

Proof :

Let us consider that job j comes immediately after job i, then we will be having the following type of scenarios :(See Figure 5.2)



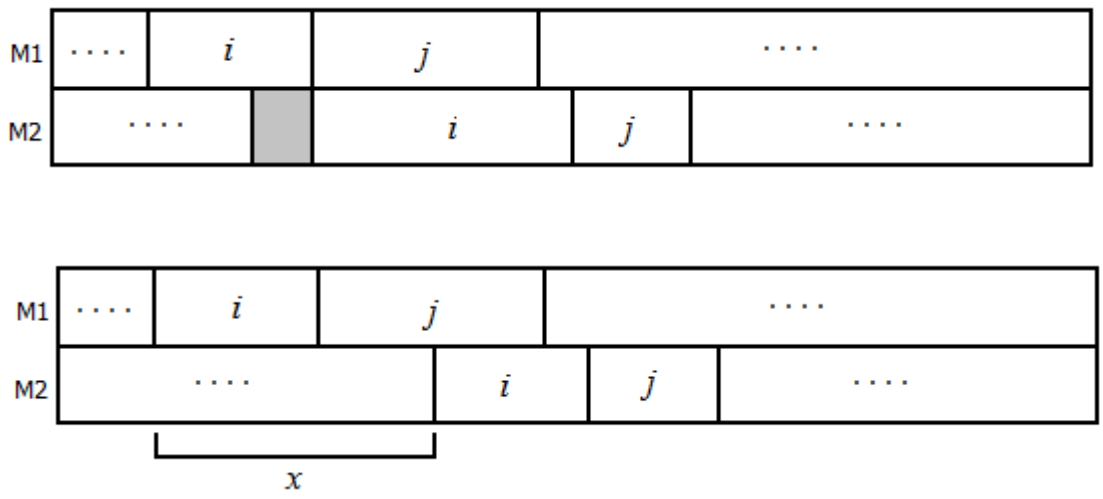


Figure 5.2 Scheduling representing different scenarios when job *j* comes after job *i* .

Let us denote the length of the time period from the beginning to job i to the end of job j by W_{ij} .Then, W_{ij} is nothing but the maximum of sum of processing times of jobs on machine 1 and machine 2 as follows .

$W_{ij} = \max \{ P_{i1} + P_{j1} + P_{j2} , P_{i1} + P_{i2} + P_{j2} , x + P_{i2} + P_{j2} \}$

We can write the same equation as

$W_{ij} = \max \{ P_{i1} + P_{j2} + \max\{ P_{j1} , P_{i2} \} , x + P_{i2} + P_{j2} \}.$

In the same manner , if job j comes before job i then

$W_{ji} = \max \{ P_{j1} + P_{i2} + \max\{ P_{i1} , P_{j2} \} , x + P_{i2} + P_{j2} \}.$

Let us say that job i is scheduled immediately after job j , then we can say

$\min \{ P_{j1} , P_{i1} \} \leq \min \{ P_{i1} , P_{j2} \}$

By multiplying the above inequality with -1 , we get

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

Now adding $P_{i1} + P_{i2} + P_{j1} + P_{j2}$ to both sides ,we get

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

Simplifying the above equation, we get

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

which would imply the following

$W_{ji} \leq W_{ij}$.

which means that by swapping the jobs i and j , the C_{max} value will remain the same.

5.4 Theorem 1 :

The sequence L : L(1) , ... , L(n) constructed by the algorithm F2 || C_{max} is optimal.

Proof :

Let us consider that O is a set of all optimal sequences .

Let us say our sequence L obtained by using the algorithm does not belong to this set O.

$L \notin O$.

Consider a sequence R where $R \in O$ i.e. R is an optimal sequence .

Let us say that sequence L and R have a similar job order till n-1 with n being the last job(maximal).

L(v) = R(v) where v = 1,2,....., n-1 .

Let L(n) = i and R(n) = j where $i \neq j$.

In the sequence R, job i may or may not be a immediate successor to job j. Let us say that job k is scheduled between job j and i .Now, job sequence R and L will look as follows

L : 1, 2, ... , n-1, i, k, j.

R : 1, 2, ... , n-1, j, k, i.

In L ,as job k is scheduled immediately after job i we can say (From Lemma 6.9)

$\min \{ P_{k1} , P_{i2} \} \geq \min \{ P_{i1} , P_{k2} \}$.

This implies that in sequence R ,we can swap k and i without increasing the C_{max} value (from Lemma 6.10).Job i is swapped with its immediate predecessor without increasing the objective value. Finally, we get a sequence R' which will still be optimal and belongs to the set O and R'(v) will now be equal to L(v) i.e.

R'(v) ϵ O ;

R' (v) = L (v) where v = 1, 2, ..., n.

contradicting that n is maximal

Therefore L is optimal and L ϵ O.

CHAPTER 6

Open Shop Problems

6.1 Two machine open shop problem O2 || C_{max}

In an open shop problem, each job i contains a number of operations O_{ij} (j = 1, ... ,m) where an operation O_{ij} has to be processed only on machine M_j . These operations can be performed in an arbitrary order i.e. there is no precedence relation between the operations .

When the processing times of the operations are arbitrary and with no preemption allowed, the two machine open shop scheduling problem with make span as optimality criterion is the only problem that can be solved in polynomial time . Let us have a look at the algorithm defined to solve this problem.

O2 || C_{max}

Let us assume the two machines present in our problem as machine A and machine B .Let a_i , b_i be the processing times of operations belonging to job i on machine A and B respectively.

Now, we define two sets I and J which would hold the jobs with following properties :

$I = \{ i \mid a_i \leq b_i ; i = 1, ... ,n \}$ and

$J = \{ i \mid b_i < a_i ; i = 1, ... ,n \}$.

The above notation simply means that set I consists of all the jobs where the processing time of operation on machine A is greater than or equal to the processing time of operation on machine A and set J consists of all those jobs whose operations on machine B have a greater processing time than the operations on machine A.

Now,

$a_r = \max \{ a_i \mid i \in I \}$ and

$b_r = \max \{b_i \mid i \in J\}$.

This means that a_r is the maximum processing time among all a_i of jobs belonging to set I and b_r is the maximum processing time among b_i of jobs belonging to set J .

We will be considering two cases here :

Case 1 :

When $a_r > b_r$ (Here r is nothing but the job in a_r), an optimal schedule is constructed in the following manner :

On machine A ,schedule all jobs in I except r (I - r) in an arbitrary manner , then all jobs in J in arbitrary order and lastly job r.

Now on machine B, schedule job r first then all the jobs in I except r (I - r)in the same order as scheduled on machine A and lastly all jobs in J in the same order as followed on machine A.

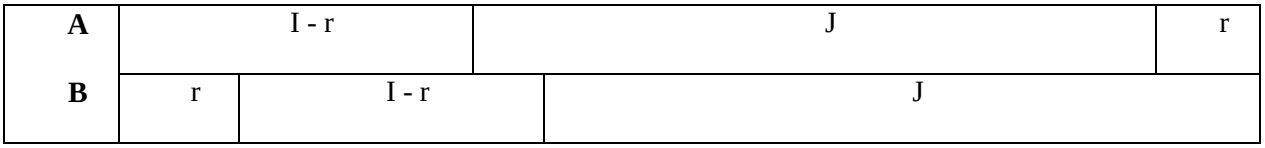


Figure 6.1 Optimal Schedule for $O2||C_{max}$ when $a_r > b_r$.

Case 2 :

When $a_r \leq b_r$ (In this case r is the job in b_r), an optimal schedule is constructed in the following manner :

On machine A , schedule job r ,then all jobs in J - r in arbitrary order and lastly all jobs in I in arbitrary order.

On machine B , schedule all jobs in J -r in the same order as on machine A ,then all jobs in I in the same order as in machine A and lastly job r.

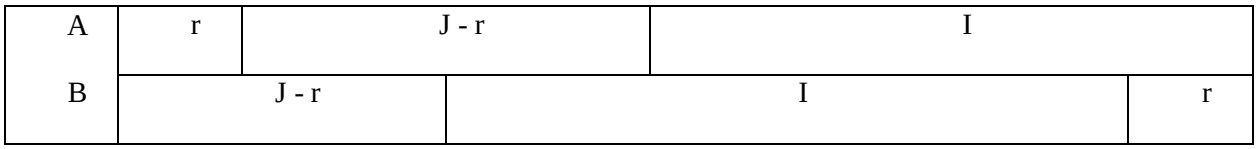


Figure 6.2 Optimal Schedule for O2||C_{max} when $a_r \leq b_r$.

6.2 An Example problem :

	M_1	M_2
J_1	3	3
J_2	4	1
J_3	2	5
J_4	4	2
J_5	5	8

Table 6.1 An example problem illustrating O2||C_{max} Algorithm.

In Table 6.1,we have two machines M_1 and M_2 and five jobs with their respective operations on both the machines.

Firstly, let us find the set I and J.

Set I = set of jobs whose $a_i \leq b_i$.

Here apparently, jobs 1,3 and 5 hold this property .So ,these jobs are inserted into set I .

I = { 1, 3 , 5 }.

Set J = set of jobs whose $b_i > a_i$.

Here apparently , the remaining jobs 2 and 4 hold this property. So they are inserted into set J.

J = { 2, 4 }.

Now, let us find the processing times a_r and b_r from the respective sets I and J.

Among all the jobs in I, job 5 has the maximum a_i value of 5.

Therefore,

$a_r = a_5 = 5$.

Among all jobs in J , job 4 has the maximum b_i value of 2 .

Therefore ,

$b_r = b_4 = 2$.

As we can see, apparently here $a_5 > b_4$ (5 > 2).

So, we follow case 1.

Now r = 5 .

and the schedule constructed following our algorithm would look like :

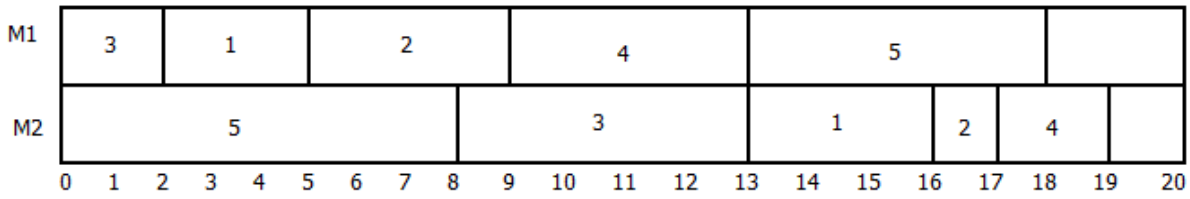


Figure 6.3 Machine Oriented Gantt Chart representing the schedule obtained using O2||C_{max} Algorithm.

Here, C_{max} = 19.

CHAPTER 7

Mixed Shop Scheduling Problems

7.1 Mixed Shop Problems

A mixed shop can be defined as a combination of open shop and job shop. This means in a mixed shop, we will have a mixture of open shop jobs and job shop jobs. The order in which the operations are processed is fixed in a job shop job but in a open shop job, the order in which the operations of the job are processed can be arbitrary. We denote the mixed shop problems with X.

As mixed shop problems consist of both job shop jobs and open shop jobs, only some of the mixed shop problems have been solved in polynomial time. Among them are $X2|n_i \leq 2|C_{max}$ and $X2|pmtn; n_i \leq 2|C_{max}$,which have been solved in O(n log n) steps[8]. An $O(r^3 + n)$ algorithm[9] has been defined for solving $X2|n_J = 2;pmtn|C_{max}$ problem and the problem $X2|n_J = k; n_O = l \mid C_{max}$ has been solved in $O(r^{3n}_J 2^n_O)$time.

7.2 Mixed Shop with two jobs

$X \mid pmtn ; n = 2 \mid C_{max}$

Shakhlevich and Sotskov[7] have shown that the preemptive mixed shop problem with two jobs i.e. one open shop job and one job shop job with an arbitrary regular objective function can be solved using an O(*r*) algorithm where r is the maximum number of operations among the jobs. They have also shown that the problems $X \mid n = 2 \mid C_{max}$ and $X \mid n=2 \mid \sum C_i$ are NP-hard.

Let us assume that the given jobs are J_1 and J_2 where job J_1 represents the job shop job and job J_2 represents the open shop job. Let O_{1j} be operations belonging to the job shop job J_1

where j = 1,.....,n_1 where operation O_{1j} has a processing time of P_{1j} and is supposed to be processed on machine $\mu_j \in \{M_1,.....,M_m\}$.Let O_{2j} be the operations belonging to the open shop job J_2 where j = 1,.....,n_2 = m where O_{2j} has a processing time of P_{2j} and is supposed to be processed on machine M_j .

let T_i be equal to the total processing time of J_i where *i* = 1,2.

$$T_i = \sum_{j=1}^{n_i} P_{ij} \quad (i = 1, 2.)$$

Now, we have two cases to consider here : $T_1 \geq T_2$ and $T_1 < T_2$.

7.2.1 Case 1 :

$T_1 \geq T_2$

Lets us say there exists an index *l* ($1 \leq l \leq n_1$) such that

$$\sum_{j=1}^{l} P_{1j} = T_2 .$$

This means that the sum of processing times of operations in job1 which are performed on machines $M_{1,}.....,M_l$ must be equal to T_2. If not ,we split an operation on J_1 into two operations.

Now, for every value of k = 1,....., m we say

$$S_k = P_{2k} + \sum_{\substack{\mu_j = M_k \\ j \leq l}} P_{1j} .$$

and $S = S_{k^*} := \max_{k=1}^{n} S_k$.

Now when $S \leq T_2$ and when $S > T_2$,we use different approaches to solve our problem.

7.2.1.1 Sub case 1 : $S \leq T_2$

In this case, it is possible to construct a schedule where C_1 (completion time of J_1) = T_1 and C_2(Completion time of J_2) = T_2 which is an optimal schedule. Assume that we only consider operations O_{11},.....,O_{1l} belonging to J_1 and construct a schedule U with a make span of T_2, then it equivalent to proving that our schedule is optimal. In addition when the operations $O_{1,l+1}$,....., O_{1,n_1} belonging to J_1 are scheduled without any interruption on their respective machines, we get $C_1 = T_1$ and $C_2 = T_2$.

Firstly, we need to come up with a schedule U. To do this, we make use of the two job open shop problem algorithm (equivalent to O2||C_{max}) .We consider our above problem as two job

open shop problem with processing times

$p_{1k} = \sum_{\substack{\mu_j = M_k \\ j \le l}} P_{1j}$ and $p_{2k} = P_{2k}$ for k = 1,.....,m.

Let us see how we can solve this using the following example problem :

J1	M1 (4)	M3 (3)	M2 (2)	M1 (1)	M3 (2)	M4 (6)
J2	M1 (5)	M2 (2)	M3 (3)	M4 (4)		

Figure 7.1 An example mixed shop problem for sub case 1.

In Figure 7.1, J_1 represents the job shop job and J_2 represents the open shop job along with their respective operations and processing times. As J_1 is a job shop job, it has to be processed in

the job order mentioned in the example. Operations in J_2 can be processed in an arbitrary order.

Let us calculate the values of T_1 and T_2.

Here apparently $T_1 = 4+3+2+1+2+6 = 18$.

$T_2 = 5+2+3+4 = 14$.

We have $T_1 > T_2$.

Apparently, we do not have a index *l* here such that $\sum_{j=1}^{l} P_{1j} = T_2$.So,let us break the last operation of J_1 on M_4 into two operations as following :

J1	M1 (4)	M3 (3)	M2 (2)	M1 (1)	M3 (2)	M4 (2)	M4 (4)
J2	M1 (5)	M2 (2)	M3 (3)	M4 (4)			

Figure 7.2. Problem after breaking operations on J_1 .

Now, from the above figure we can say $l = 4$.We do not consider the last operation on J_1 and calculate the corresponding S_k for k = 1,.....,m.

$S_1 = 5+4+1 = 10$; $S_2 = 2+2 = 4$; $S_3 = 3+3+2 = 8$; $S_4 = 4+2 = 6$.

From the above $k^* = 1$ and $S = S_{k^*} = 10$.

As $S < T_2$, we follow sub case 1 .

Let us now truncate this problem into a two job open shop problem as follows :

i	1	2	3	4
a_i	5	2	5	2
b_i	5	2	3	4

Table 7.1 Two Job Open Shop problem

In Table 7.1, $\boldsymbol{a_i}$ implies all the operations belonging to J_1 and $\boldsymbol{b_i}$ implies all the operations belonging to

J_2 where i is the machine number.

Applying the two job open shop algorithm from * ,we get

I = { 1 , 2 , 4 }

J = { 3 }

$a_r = a_1 = 5$;

$b_r = b_3 = 3$.

$a_r > b_r$ and r = 1.

The resulting schedule will be as follows :

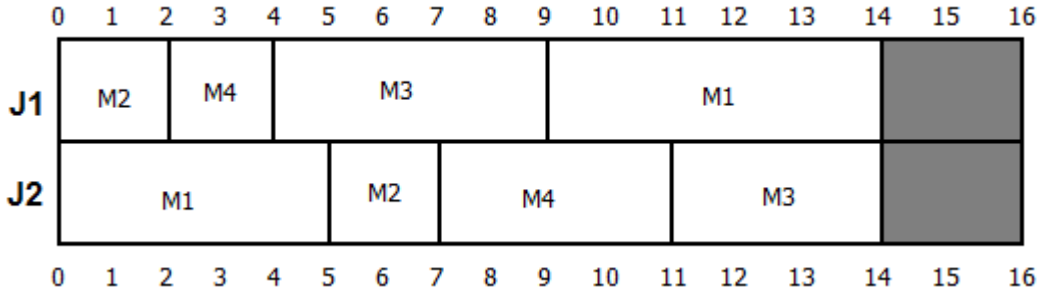


Figure 7.3 Resulting Schedule obtained after using the algorithm in 6.1 .

Now, the last step in this algorithm is to cut the schedule in Figure 7.3 along the original operations of J_1 into smaller pieces and reschedule these pieces such that we get a preemptive schedule for the above truncated problem. Adding the operations of J_1 scheduled after T_2 will give us the final optimal schedule.

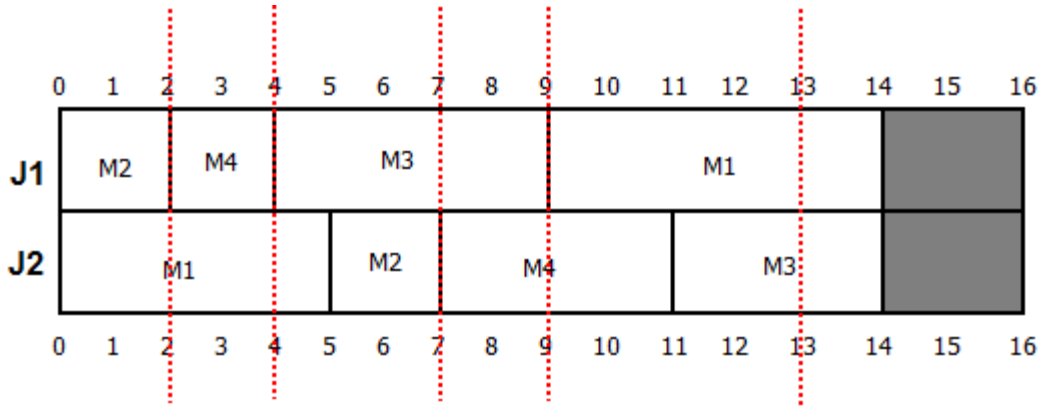


Figure 7.4 Schedule representing the cuts along the operations of J_1 .

We split the schedule along the red dotted lines in our example which represent the operations in J_1 (See Figure 7.4).This leaves us with the following smaller operations in J_2 .

J_2 : M1(2) ; M1(2) ; M1(1) ; M2(2) ; M4(2) ; M4(2) ; M3(2) ; M3(1) .

After rescheduling these operations and adding the operations in J_1 after T_2 , our final optimal schedule will be :

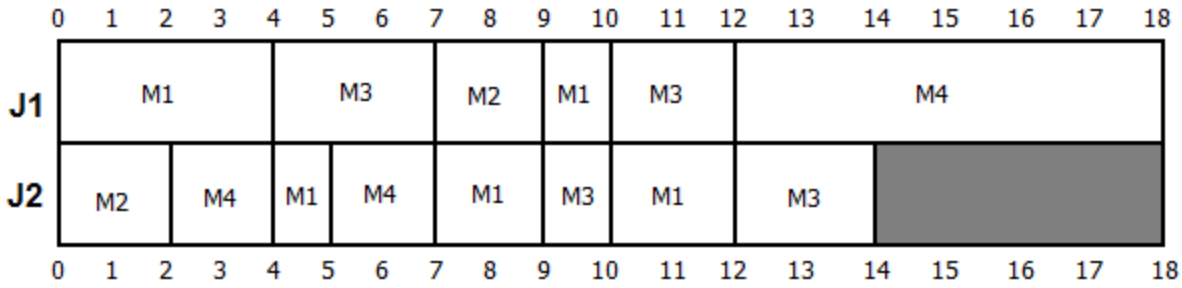


Figure 7.5 Optimal Schedule for sub case 1 instance of X $|n=2;pmtn|C_{max}$

In figure 7.5 ,

$C_1 = T_1 = 18$.

$C_2 = T_2 = 14$ and

$C_{max} = 18$.

7.2.1.2 Sub case 2 : $S = S_k^* > T_2$

In this case, there exists no schedule with $C_i = T_i$ for i =1,2.Even when $C_2 = T_2$,there will be at least one operation O_{1j} where $\mu_j = M_{k^*}$ and $j \leq l$ which would finish later than T_2 making the completion time of J_1 greater than T_1.

Let us consider the following example to explain how to proceed in this case (See 7.6)

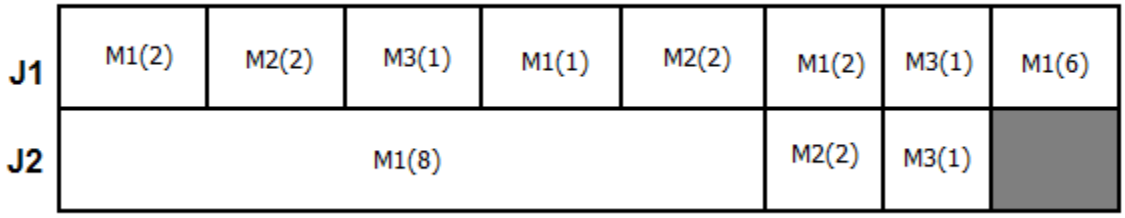


Figure 7.6 Example problem illustrating sub case 2 .

In Figure 7.6 ,

$T_1 = 17 > T_2 = 11$.

So we follow case 1 and find an index l such that $\sum_{j=1}^{l} P_{1j} = T_2$. For this ,the last operation in J_1 on machine 1 needs to be ignored for now.

We have l = 3 and so

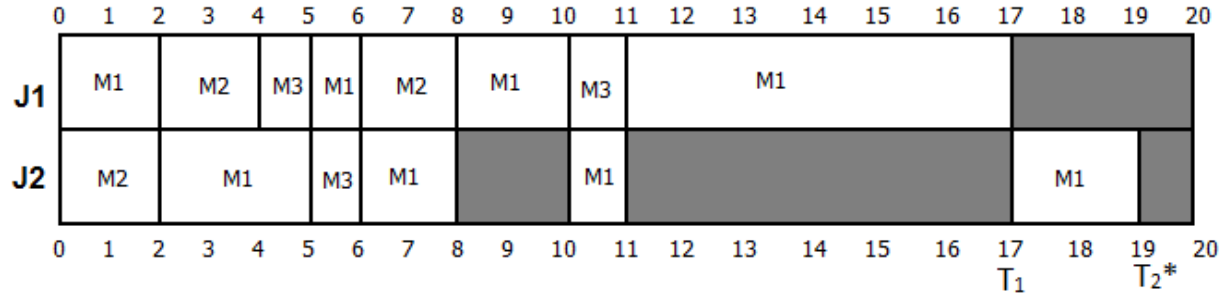
$S_1 = 13$; $S_2 = 6$; $S_3 = 3$.

Apparently, Here $S_1 > S_2 > S_3$ and $S_{k^*} = S_1 = 13$ and $k^* = 1$.

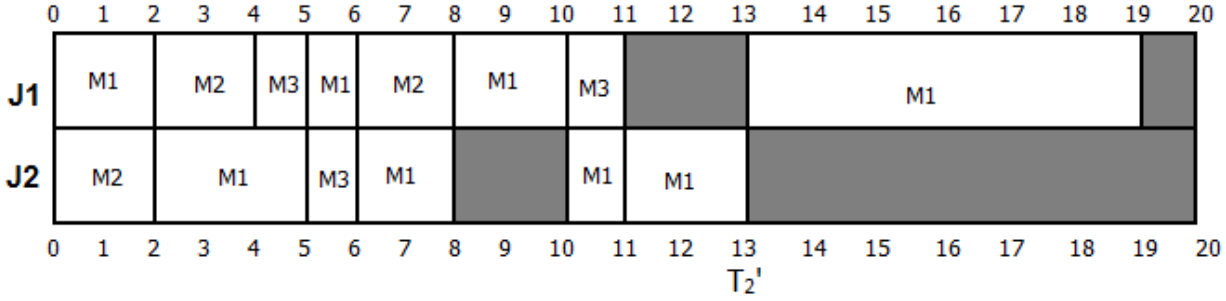
We have $S_{k^*} > T_2$.

Let us first consider the following cases

i. Here $C_1 = T_1$ and we are minimizing C_2 as follows



ii. Here $C_2 = T_2'$ and $T_1 < C_1 < T_1^*$ as follows



iii. Here $C_2 = T_2$ and we minimize C_1 as follows

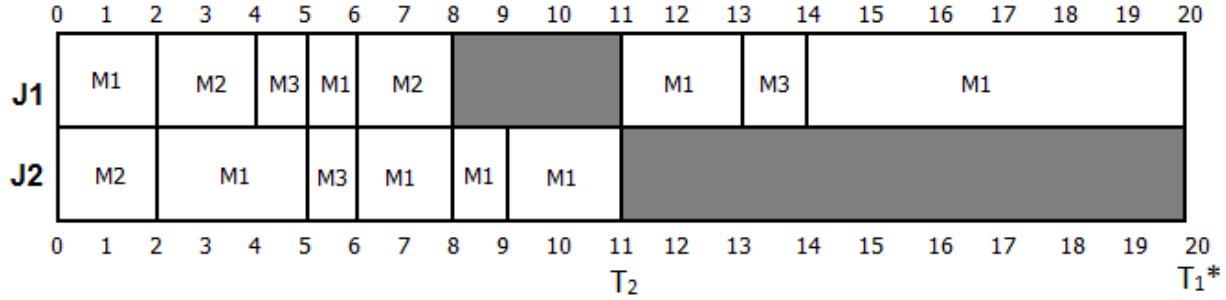


Figure 7.7 Possible Schedules for the sub case 2 where $S > T_2$ and $T_2 \leq T_1$

In this example, we have k* = 1 .Now, Let us denote the minimal C_2 value in case 1 by T_2^* and the minimal C_1 value in case 3 by T_1^* .

For every value of $T_2 \leq T \leq T_2^*$,we define

$f(T) = \min \{ f(s,T) \mid T_1 \leq s \leq T_1^* \}$

Then,

$F(T^*) = \min\{ f(T) \mid T_2 \leq T \leq T_2^* \}$ will be our optimal solution . We consider at the most n_1 different values of T to calculate T_1^* .

we start from the optimized schedule where $T = T_2^*$ (case i) and go till $T = T_2$ .From the figure, we can say that to decrease the value of T from T_2^* in case i ,the only way to do this is to increase s from T_1 to T_2^* i.e. we move the last operation of J_1 on M_1 to the right by $T_2^*-T_1$ units and then move the last operation of J_2 to the far left as possible.

Now,

$f(T_2') = \min \{ f(T) \mid T_2' \leq T \leq T_2^* \}$

We further try to decrease the value of T . In our example ,we reduce T from $T = T_2'$ to $T = T_2$ by further moving the operations of J_1 to the left and the operation of J_2 to the left as in case iii. This will be the last instance in our example as T cannot be further reduced (as seen in the figure) and now $T = T_2$ and $C_1 = T_1^*$.Finally ,we have $F(T^*) = \min\{ f(T_2^*), f(T_2'), f(T_2) \}$.In our case f(s,T) will be the make span of the schedule at that instance. In simple words, we start from $T = T_2^*$ and calculate the Makespan. Now, we try to reduce T from T_2^* by changing the value of s and end up with case ii . We further reduce T till $T = T_2$ noting down the make span for every instance. Now ,the optimal schedule will be the one with the minimum make span value.

In our example , we know $F(T^*) = \min\{ f(T_2^*), f(T_2'), f(T_2) \}$ which is equal to

$F(T^*) = \min\{ 19 ,19 , 20 \}$,

Hence, we can say that case i or case ii can be our optimal schedule here.

7.2.2 Case 2 :

$T_1 < T_2$

In this case ,we can define

$S_k = P_{2k} + \sum_{\mu_j = M_k} P_{1j}$ for k = 1,.....m.

As in the previous case, we will be having two cases here when $S_k \leq T_1$ and $S_k > T_1$.

7.2.2.1 sub case 1 : $S_k \leq T_1$

As in our case 1 ,when $S_k \leq T_1$,there exists an optimal schedule with $C_i = T_i$ where i = 1,2.

We follow the same procedure as in case 1 except that here $J_1 < J_2$ and we add the extra jobs of J_2 (instead of J_1 as in previous case) to the optimal schedule obtained using the $O2||C_{max}$ algorithm.

7.2.2.2 sub case 2 : $S_k > T_1$

When $S_k > T_1$ and $S_k \leq T_2$, then we will have an optimal schedule with $C_i = T_i$ where i = 1,2.But when $S_k > T_1$ and $S_k > T_2$, we follow the same procedure as in the previous sub case 2(when $T_1 > T_2$).

CHAPTER 8

DEALING WITH NP-HARD PROBLEMS

8.1 Proof that O3 || C_{max} is NP-Hard

There are efficient algorithms for solving the non preemptive open shop problem where the number of machines m = 2 .Does this mean that there exists a similar algorithm to solve the non preemption open shop problem for m =3 ?We would be able to answer this question by proving the fact that O3 || C_{max} is NP-hard[15]. For this purpose, Gonzalez and Sahni[10] reduced the partition problem into a instance of a three machine open shop problem. So, in simple terms this would be our problem:

Given an non preemptive open shop problem where m = 3, n number of jobs with processing time $P_{i,m}$ where $1 \leq i \leq n$ and $1 \leq m \leq 3$ .Let us say we have a dead line D, so do we have a schedule where the make span is less than or equal to D ? The partition problem is: can the set of all objects J = { j_1,.....,j_n } with weights w(j_i) be partitioned into two sets J_1 and J_2 such that the sum of the weights of jobs in each set be equal to T/2 where T = $\sum$ w(j_i) where $1 \leq i \leq n$.

Let us consider the following instance of the open shop problem

From partition problem J = { j_1,....., j_n } , let us construct an open shop problem with 3n+1 number of jobs and three machines m = 3.Let us say we have all the jobs with only one operation except for one job which has three operations ,one on each machine with a processing time of T/2 units.

Let us say we have n number of jobs processing on each machine plus the extra special job. Let these n jobs imply the objects in the set J and the processing times imply their respective weights. Then, the processing times would look like :

$w(j_i) = P_{i,1}$; $P_{i,2} = P_{i,3} = 0$; for $1 \leq i \leq n$;

$w(j_i) = P_{i,2}$; $P_{i,1} = P_{i,3} = 0$; for $n+1 \leq i \leq 2n$;

$w(j_i) = P_{i,3}$; $P_{i,1} = P_{i,2} = 0$; for $2n+1 \leq i \leq 3n$;

$P_{3n+1,1} = P_{3n+1,2} = P_{3n+1,3} = T/2$;

where $T = \sum_{i=1}^{n} w(ji)$ and $D = 3T/2$.

We now say that this problem can be done with a make span less than or equal to D only if there exists a partition for J.

Proof :

Let us prove this by contradiction. Firstly, assume that we have a schedule with a make span less than or equal to 3T/2 and that J has no partition.

As we know job 3n+1 has a processing of T/2 units on each machine and since preemption is not allowed ,this job has to be processed all the time on any one of the machines.(see Figure 8.1)

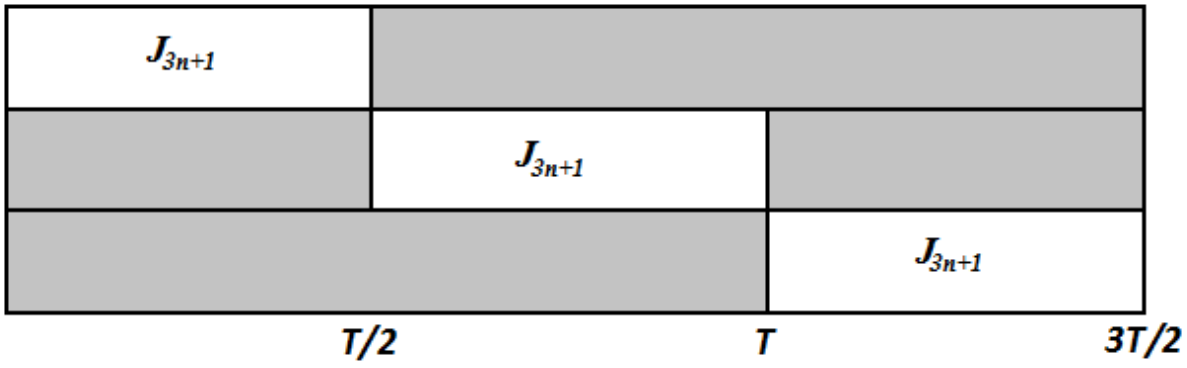


Figure 8.1 Architecture representing a instance for proving $O3||C_{max}$ is NP-hard.

As preemption is not allowed, there exists at least one operation from job 3n+1on any of the machine l such that it starts at T/2 and is processed till T units of time. On this machine l, apparently there would be two disjoint blocks of idle times with a length of T/2 units each. As we know , n number of jobs are to be processed on each machine with a total processing time of T.

As J has no partition, it cannot be divided into two sets where each set has a total processing time of T/2 and so all the time preceding the job j_{3n+1} on machine l cannot be used. So, all the remaining jobs to be processed on machine l will require more than T/2 units after processing job 3n+1 . This makes the finishing time exceed 3T/2 time units contradicting our assumptions.

Therefore0, we cannot have a schedule with a Makespan less than or equal to D = 3T/2 when the set J has no partition. Hence, the non preemption open shop make span problem is NP-hard when m = 3.

Since the above partition problem in not NP-complete in the strong sense(unless P = NP) , we cannot say that the open shop problem in NP-hard in the strong sense. However, it has been proven that an open shop problem is NP-hard in the strong sense with arbitrary number of machines by reducing the 3-partition problem to it.

8.2 THE DISJUNCTIVE GRAPH MODEL

The Disjunctive graph model is a commonly use method to represent general shop scheduling problems mainly job shop. It can be used to construct optimal schedules.

For a given instance of a shop problem, a disjunctive graph $G(V, C, D)$ can be defined as follows:

V : V denotes a set of vertices corresponding to the operations of any job in the schedule. This set has additional two vertices called the source and the sink (0,*).Each of these nodes have an associated weight to it which is nothing but the processing times of that particular operation the node is representing the weights of source and the sink are zero.

C : *C* is a set of conjunctive arcs which reflect the precedence constraints initially connecting every two consecutive tasks of the same job.

D : Undirected disjunctive edges belonging to set D connect mutually unordered tasks which require the same machine for their execution (a disjunctive edge can be represented by two opposite directed arcs).

The job shop scheduling problem requires to find an optimal order of all tasks on the machines, resulting in a schedule with the minimal length. In the disjunctive graph model, this is equivalent to directing all disjunctive arcs, i.e. to turn each undirected disjunctive arc into a directed arc. This is called a selection S. Thus, we can define a selection S as a set of directed disjunctive arcs. The disjunctive arcs which have been directed are called fixed. By fixing directions of all disjunctive edges, the execution order of all conflicting tasks requiring the same machine is determined and a complete schedule is obtained. We have to make sure that :

- Our resulting graph is acyclic and
- each disjunctive undirected edge is converted to a fixed directed arc.
- Then, the resulting length of the longest path from the source to sink i.e. the sum of all the processing times, in our new graph should be minimal which is nothing but out make span.

Now , a selection S is called a complete selection if :

- Every disjunctive arc in the graph is fixed and

- The graph G(S) = (V , C ∪ S) does not contain any cycles i.e. an acyclic graph .

  Apparently, any feasible schedule will make an complete selection. So, when we have scheduling problem we can say that our goal is to find a complete selection which optimizes the objective function. For a given complete selection S, the make span or maximum of completion times (C_{max}) is equal to the length of the longest path (sum of processing times of operations along this path) from the source 0 to the sink * .This longest path is called the critical path.

8.3 Disjunctive Graph Example using a job shop problem

In Table 8.1,we have a job shop problem and let us try to give a optimal schedule and represent that schedule using disjunctive graphs .

Job J1	M3(2)	M1(4)	M2(2)
Job J2	M2(4)	M1(3)	
Job J3	M1(3)	M2(2)	M3(3)

Table 8.1 An example job shop problem to illustrate the disjunctive graph model.

Figure 8.2 represents a disjunctive graph for a schedule for the job shop problem in Table 8.1.On directing the disjunctive arcs here , we get a schedule here.(see Figure 8.3).

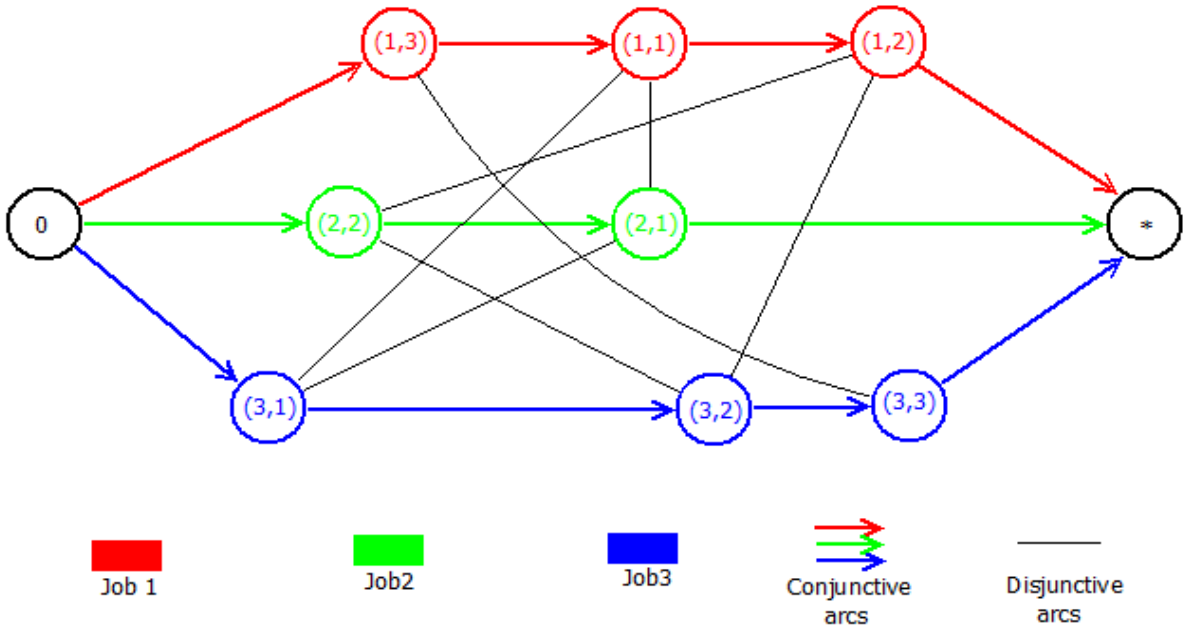


Figure 8.2 Disjunctive Graph Representing the problem in Table 8.1 .

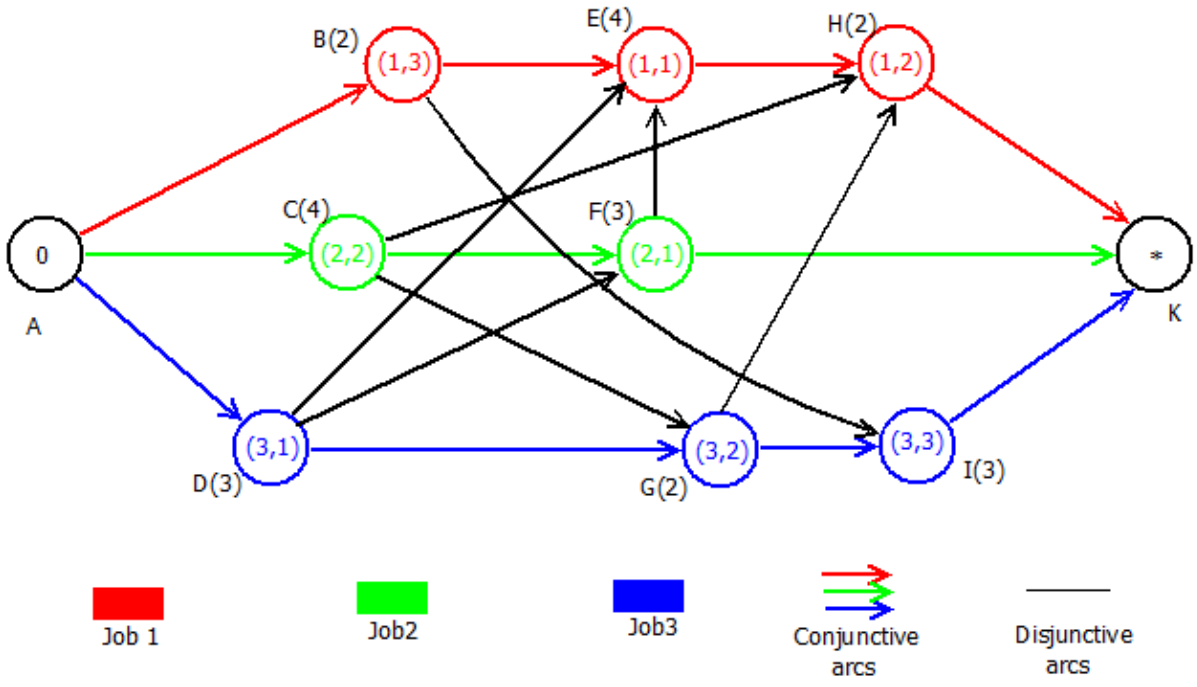


Figure 8.3 Disjunctive Graph representing a schedule for the problem in Table 8.1

On observation, the longest path in figure 8.3 is A →C→F→E→H→K resulting in a make span of 13. A Gantt diagram for the above disjunctive graph in Figure 8.3 follows

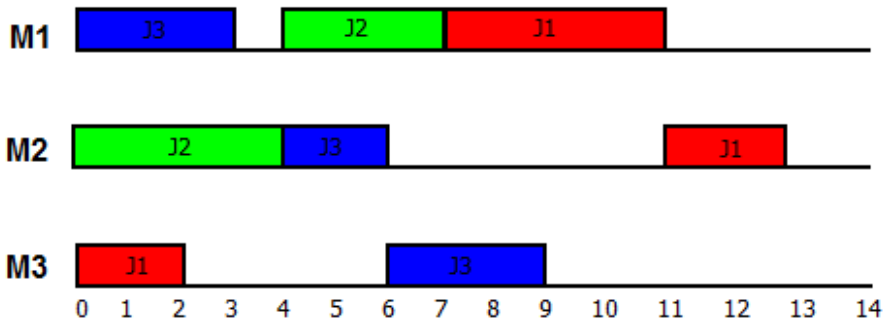


Figure 8.4 Gantt Chart for a Schedule represented in the Disjunctive graph-Figure 8.3

From figure 8.4, C_{max} = 13.

Let us swap one of the disjunctive arcs and see if we could come up with a better schedule .

Let us change the order of jobs on machine one. From the disjunctive graph, we can see that job 1 is being performed on machine 1 after job 3 .Let us swap the edges in the graph so that job 3 comes after job 1 on machine 1 and let us see how it affects the make span.

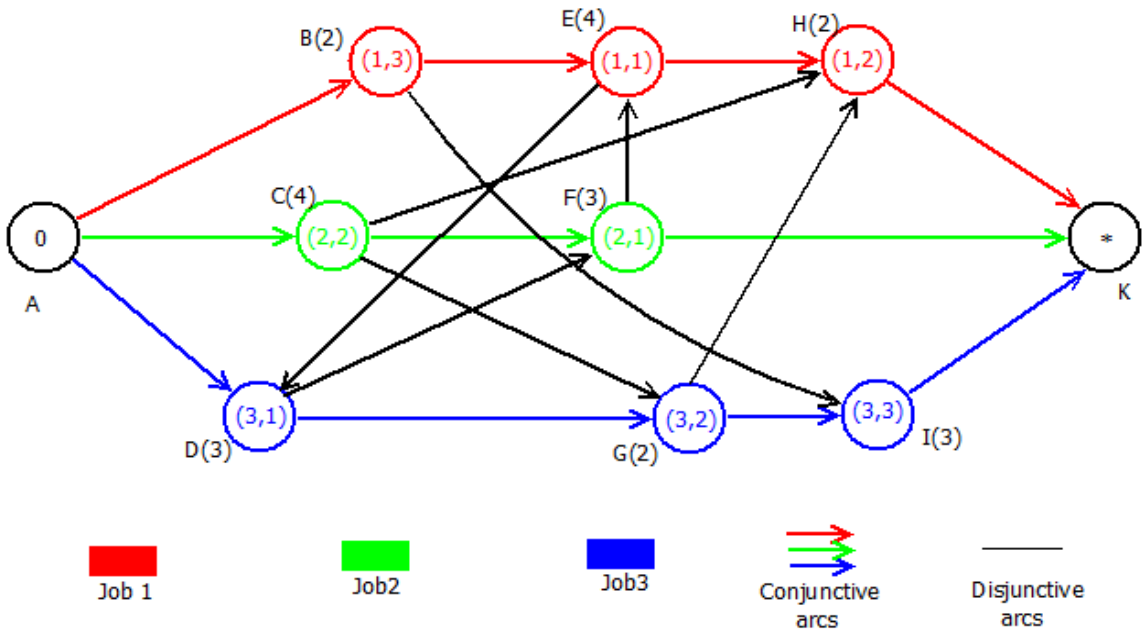


Figure 8.5 Disjunctive Graph representing a schedule

In figure 8.5, the longest path is A→C→F→E→D→G→I→K resulting in a make span of 19 which is much worse than our previous schedule. Likewise , local neighborhood search techniques can be used to reach optimal or near optimal solution(See Section 8.5) . Thus, a disjunctive graph not only is a way of representing a schedule but also helps to construct better ones.

8.4 Neighborhoods

In a combinatorial optimization problem, we are faced with finding an optimal object from a given finite set of objects. The optimality here relates to some criterion which decides the quality of each solution. The set of feasible solutions is discrete or can be reduced to discrete from which our goal is to find the best solution. Mostly, these kind of problems are NP-hard[12]. So, we focus on finding good approximation algorithms using local search methods.

Neighborhood plays an important role in deciding the quality of the local search algorithm. An efficient neighborhood leads to a high quality local optima. Let us define a neighborhood in our job shop scheduling problem context.

Consider a job shop scheduling problem represented using the disjunctive graph G = (V,C,D) (Refer to Section 8.2). Let S be a complete selection in G corresponding to an acyclic graph G(S) = (V,C ∪ S). An immediate neighbor of S can be defined as the set of all complete selections obtained from S by reversing an arc (*v*,*w*) belonging to the critical path where *v* and *w* are processed on the same machine .This neighbor is denoted by $N_1(S)$. Likewise, we may have $N_2(S)$, $N_3(S)$, and so on.

8.4.1 Lemma 1 : Consider a complete selection S with *p* as the longest path in G(S) . Let (*v*,*w*) be a arc in *p* such that *v* and *w* are processed on the same machine. Then , S' which is derived from S by reversing (*v*,*w*) will also be a complete selection.

Proof :

Let us consider G(S') derived from G(S) is cyclic.

Then, this implies that arc (*w*,*v*) obtained by reversing (*v*,*w*) in S belongs to the cycle *c* in S' as G(S) has no cycles .

We know that (*v*,*w*) does not belong to cycle *c*.

Therefore, *c* has more than two vertices.

In addition, we can say that *v* and *w* are the only vertices which belong to both c and the longest path *p*.

Now, let us say that we replace the arc (*v*,*w*) in *p* by the sub path from *v* to *w* in *c* .Then, there will be a path in G(S) longer than *p*. This is a contradiction to the fact that *p* is the longest path in G(S).

Thus by proof of contradiction, we can say that G(S') is acyclic and thus a complete selection.

We represent the set of all complete selections obtained from a complete selection S by reversing an arc(*v*,*w*) in the critical path in G(S) by $N_1(S)$ where *v*,*w* are supposed to be processed on the same machine.

A neighborhood N is called opt-connected , if from S we can find an optimal solution from a limited number of moves . This means that we can reach a optimal solution S_k from $S_0 = S$ where we have a sequence of complete selections S_0 , S_1 , . . . ,S_k such that $S_{i+1} \in N(S_i)$ for i = 0,1.....,k-1.It has been shown that N_1 and N_2 is opt-connected[1].

8.5 Local Neighborhood Search or Hill Climbing

Combinatorial optimization problems relate to the set of problems in which we face a situation where we have to find an optimal solution among a finite or infinite set of solutions. Here, optimality may relate to some criterion which decides the quality of each solution. Many of these combinatorial optimization problems are NP-hard[12] .As we know, NP- hard problems cannot be solved in polynomial times. So, our interest is in the algorithms which can find near optimal solutions with decent running times. Local search algorithms fall under this category of approximation algorithms.

A local neighborhood search or hill climber algorithm searches the immediate neighbors for a better solution. It simply accepts the neighbor solutions that are better than the current solution. We recursively do this process until the algorithm could not find a better solution than the current solution. Then, the algorithm stops and the current solution would be our optimal or near optimal solution. This solution here is called the local optima. The disadvantage of the local search is that the algorithm may be caught in a poor local optima. Thus the local optima may not be the best solution always. To overcome this problem ,we define the simulated annealing algorithm in section 8.6 that finds the best solution possible in a wider space.

Only a few shop problems have been solved in polynomial time till date. Many of them remain to be NP-hard problems. Local neighborhood search can be used in our scenario to achieve near optimal solutions. Let us consider the job shop problem example defined in table 8.1.Let us see how we can use local search to find a better solution .Firstly , to apply the local search we use the disjunctive graph model. Represent the job shop problem using the disjunctive graph as in Figure 8.1. Our optimality criteria be make span.

We start with a complete selection S initially. Let G(S) = (V , C ∪ S) be the corresponding graph of the selection S. Figure 8.3 represents the graph G(S) .Apparently, the make span here is 13 . Let us now do a local neighborhood search to find a better solution. Consider the neighbor $S_1 = N(S)$ defined in the below figure 8.6 .

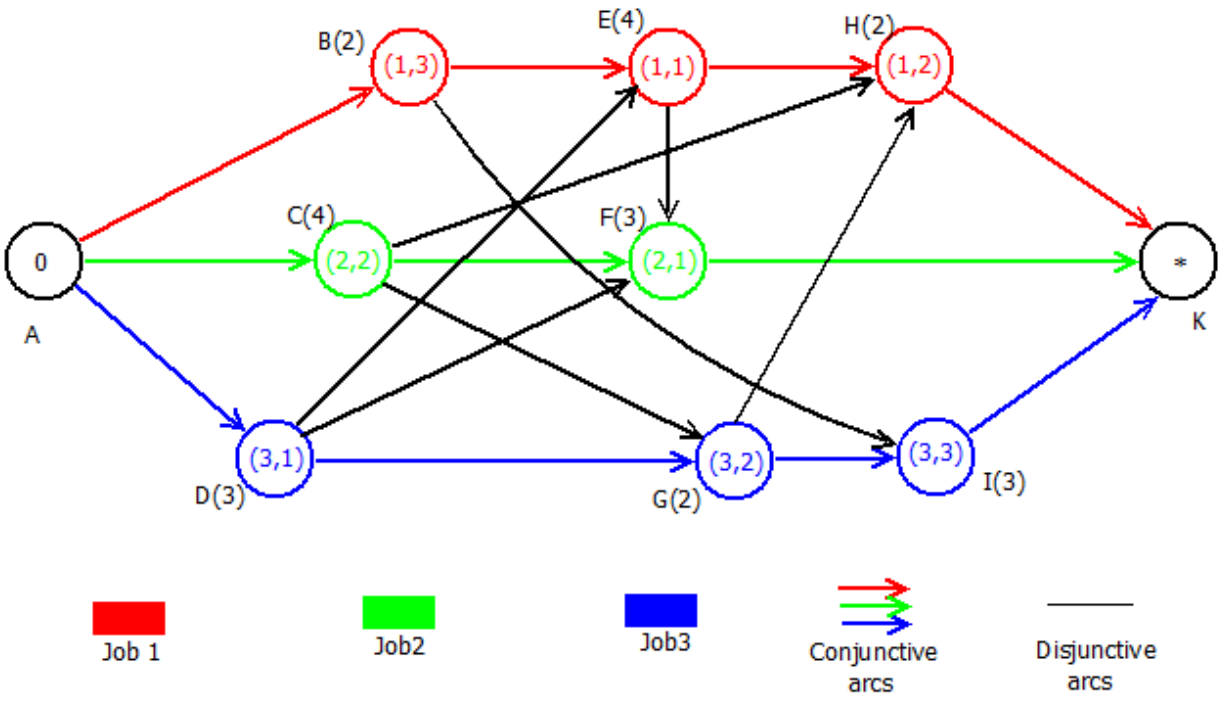


Figure 8.6 Graph representing a neighbor of S.

On observing figure 8.6,apparently the longest path here is A→B→E→F→K (Here we have more than one longest path. We choose one among them) which is our make span = 9.Apparently this is better than our previous solution from figure 8.2 which is our local optimum. Let us see if there is any neighbor with a better local optima. All the neighbors $S_2 = N(S_1)$ seem to have a poor solution than S_1. Figure 8.7 represents S_2 a neighbor of S_1 .

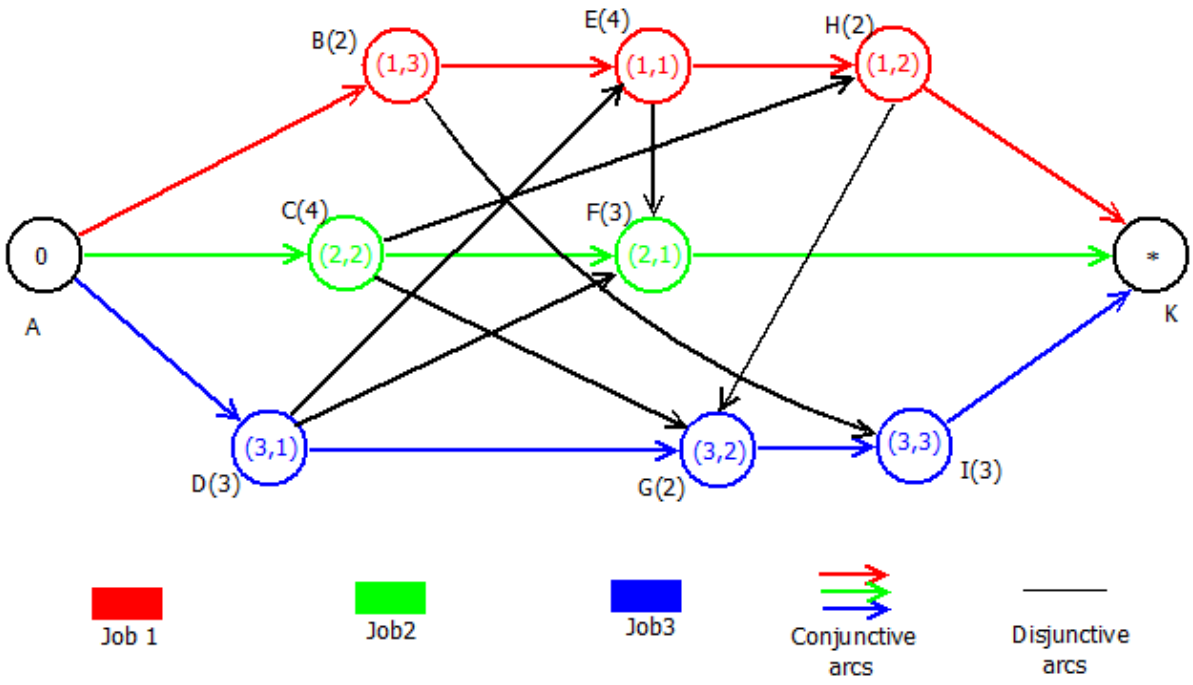


Figure 8.7 Disjunctive Graph representing a neighbor of S_1 .

From figure 8.7,the longest path is A→B→E→H→G→I→K which makes it to a sum of 13 which is much worse than our current local optima. Likewise, all the neighbors of S_1 are searched for a better solution .If none of them turn out to be better than our local optimum, the algorithm stops. Here S_1 in figure 8.6 gives our optimal solution .

8.6 Simulated Annealing

Simulated annealing is similar to the local neighborhood search algorithm except that it also accepts a neighbor with a worse solution (with some probability).It can be defined as a generic heuristic for a global optimization problem where we start from an initial state (some feasible solution) and iteratively improvise it to find a better near optimal solution .This algorithm overcomes the loop holes of the local search algorithm by considering a

wider space than the local optima. The name of this algorithm comes from the process of annealing in metal works.

Annealing is a metallurgical process involving heating and cooling of a material to alter its physical properties and reduce their defects. During the cooling process, the material tends to retain its newly obtained properties but at higher temperatures, the material tends to accept changes made to it. In simulated annealing, we follow the same process. Here , we use a temperature variable to simulate the annealing heating process. Initially, this variable is set to a high value and eventually this value falls down as the algorithm runs. As long as this variable has a higher value, the algorithm will be allowed to accept worse solutions than our current solution at a higher frequency rate . This allows our algorithm to look beyond the local optima range during the early stages of the algorithm. As the temperature variable value drops, so is the probability of accepting worse solutions which gradually results in the algorithm focusing on a specific area of the search space in which hopefully, a near optimum solution can be found. This process of gradual cooling is what makes the simulated annealing algorithm effective at finding near optimum solutions in large problems where we may have numerous local optimums.

8.6.1 Acceptance Probability Function

In simulated annealing, we know that the algorithm accepts worse solutions once in a while. But when and on which basis does the algorithm accept the worse solution which ultimately makes it go beyond the local optima.

Firstly, We check if the neighboring solution is better than our current solution. If yes, the neighboring solution becomes the current solution unconditionally. If not , then the

algorithm decides to accept it or not based on certain factors like how worse is the neighboring solution than the current solution and what is the current value of the temperature variable.

Let us denote the current state as S and the neighboring state be S'. Let C be the cost function and the cost of the state S is given as C(S) and the neighboring state S' as C(S').Here we are trying to minimize the cost(as in make span in a shop problem). We denote the temperature variable with T. The probability function is denoted by P_t at an instance when T = t and can be given as

$$P_t = \begin{cases} 1 & \text{if } C(S') \leq C(S) \\ exp(\, C(S) - C(S') / t \,) & \text{if } C(S') > C(S) \end{cases}$$

At higher temperatures , the algorithm is more likely to accept worse solutions . When the neighbor has a relatively higher cost and the difference between the neighbor cost and the current cost is small and the temperature variable has a higher value, the probability of accepting that neighbor is more.

8.6.2 Pseudo Code

The following pseudo code represents the implementation of the simulated annealing algorithm as discussed so far. Let us consider we are starting the algorithm from the feasible state S_0.For each round of the iterative process, the temperature variable is reduced by some amount defined by the user, but should reach T = 0 by the end of the time available. Metropolis loop is the essential characteristic of the simulated annealing algorithm. The

statement N(S) implies that a random feasible neighbor of S is selected. Here the function random( ) is a random number generator between [0.1) .

Pseudo code : Simulate Annealing

---

1. $S := S_0$; $C = C(S)$ // Initializing variables state S and cost C to initial state.
2. $S_b := S$; $C_b = C$ //Best state S_b and best cost C_b initialized .
3. $T := T_0$; // T_0 implies the initial temperature reading.
4. While(STOP) //Until stopping criteria is met.
5. Repeat (K) // K is a pre chosen
6. S' : = N(S) ; // Select a Neighbor of S
7. C' : = C(S') // Cost of the selected neighbor.
8. If C' < C THEN // If neighbor solution better than current.
   $S_b = S'$; $C_b = C$; // update best state and cost.
9. IF P_t > random( ) THEN // If probability greater than random
   S = S' ; C = C' // Change current state and cost
10. END Repeat;
11. T : = Update(T) ; // Temperature updated for every iteration.
12. END While;
13. RETURN S_b and C_b. // Return best solution .

CHAPTER 9

CONCLUSION

In this thesis, we have given an introduction to Scheduling and discussed different algorithms to solve the shop scheduling problems. Basically, scheduling has a significant role in many disciplines . It is nothing but decision making and deals with resource allocation to tasks with an objective to optimize a given optimality criterion. The concept of scheduling is significant in areas such as Manufacturing , Business, Engineering, Management, Computing, Industries, construction ,etc.

We have investigated various methods to solve shop scheduling problems in this thesis. It is very apparent how challenging and hard these problems can be .Research and study has been going on some of these problems for more than 50 years. In addition, shop problems have their application across different fields . We have discussed different shop problems which have been solved in polynomial time. But there are many more and in fact most of the shop problems still have not been solved and are considered to be NP-Hard problems.

BIBILIOGRAPHY

[1] Peter Brucker, *Scheduling Algorithms,* Fifth edition, Chapter 1,6,Springer Publication, March 2007.

[2] Sheldon B. Akers Jr. and Joyce Friedman. *A non-numerical approach to production scheduling problems*. Journal of the Operations Research Society of America.Vol.3,No. 4,pp 429-442,1955.

[3] P.Brucker, *An efficient algorithm for the job-shop problem with two jobs*, Springer-Verlag, Computing Volume 40,Issue 4 ,pp 353-359,1988.

[4] Yuri N. Sotskov, Peter Brucker, Svetlana A. Kravchenko, *On the complexity of two machine job shop scheduling with regular objective functions*, Vol 19,Issue 1,pp. 5-10,1997.

[5] Yu. N. Sotskov , N.V. Shakhlevich ,*NP-hardness of shop scheduling problems with three jobs*, Discrete Applied Mathematics,Vol.59 Issue 3,pp. 237–266,1995.

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

[7] N.V. Shakhlevich, Yu.N.Sotskov, *Scheduling two jobs with fixed and non fixed routes,* Computing, Vol 52,Issue 1 ,pp.17-30,1994.

[8] V.A. Strusevich, *Two-machine Super-shop Scheduling Problem* , Journal of Operational Research Society , Vol. 42(6), pp. 479-492,1991.

[9] Natalia Shakhlevich, Yuri Sotskov, Frank Werner, *Shop Scheduling Problems with Fixed and Non-Fixed Machine Order of the Job*, Annals of Operations Research, 92:281-304,1999.

[10] Sartaj Sahni, Teofilo Gonzalez, *Open Shop Scheduling to Minimize Finish Time*, Journal of the Association for Computing Machinery, Vol.23,No 4,pp.665-679,October 1976.

[11] Emile Aarts, Jan Karel Lenstra, *Local Search In Combinatorial Optimization*, John Wiley & Sons Ltd, Chapter 1,4, September 1998.

[12] Michael R. Garey, David S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*, W.H. Freeman, 1979.

[13] Michael L. Pinedo, *Scheduling Theory Algorithms and Systems*, Fourth Edition, Chapter 2, Springer Publication,2012.

[14] Zsuzsanna Vaik, *On scheduling problems with parallel multi-purpose machines*, EGRES Technical Report. 2005-02, 2005.

[15] Joseph Y-T. Leung, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapter 6, CRC Press, 2004.

[16] Rutenbar, R.A., *Simulated annealing algorithms: an overview*, Circuits and Devices Magazine, *IEEE* , vol.5, No.1, pp.19-21, Jan. 1989.

[17] Peter Brucker, *Job-Shop Scheduling Problem*, Springer US,pp.1782-1788,2009.

[18] Rutenbar , R.A., *Simulated annealing algorithms: an overview*, Circuits and Devices Magazine, *IEEE* , vol.5, no.1, pp.19-21, Jan. 1989.

[19] Michael W. Trosset, *What is Simulated Annealing?*, Optimization and Engineering, Vol 2, Issue 2, pp.201-213, 2001.

[20] Scott Kirkpatrick, *Optimization by simulated annealing: Quantitative studies*, Journal of Statistics, Vol 34, Issue 5-6,pp.975-986,1984.

[21] Peter J.M. van Laarhoven, Emile H.L.Aarts, *Simulated Annealing : Theory and Applications*, Springer Netherlands, Volume 37, pp.7-15, 1987.

[22] Toefilo Gonzalez and Sartaz Sahni, *Flow shop and Job Shop Schedules: Complexity and Approximation*, University of Minnesota, Operations Research, Vol.26, February 1978.

[23] J.K.Lenstra, A.H.G.Rinnooy Kan, *Computational Complexity of Discrete Optimization*, Annals of Discrete Mathematics, Vol 4,pp. 121-140,1979.

[24] Philippe Fortemps , *On the Disjunctive Graph For Project Scheduling* , Foundations of Computing and Decision Sciences , Vol 22 , pp.195-210,1997.

[25] A.V. Aho, *The Design And Analysis of Computer Algorithms. Addison-Wesley, Reading*, MA, 1974.

[26] M.R.Garey, D.S.Johnson, R.Sethi, *The complexity of Flow shop and Job shop Scheduling* ,Vol.1,No.2,pp.117-129,1976.

[27] T. Gonzalez, *A note on open shop preemptive schedules*, Transactions on Computers, Vol 28,pp.782-786, 1979.

[28] Peter Brucker and A. Kramer, *Shop scheduling problems with multiprocessor tasks on dedicated processors*, European Journal of Operational Research, Vol 90,pp.212-226,1996.

VITA

Graduate College

University of Nevada, Las Vegas

Megha Sairam Darapuneni

Degrees

Bachelor of Technology in Information Technology, 2012

GITAM University, INDIA.

Master of Science in Computer Science, 2014

University of Nevada Las Vegas.

Thesis Title

A Taxonomy of polynomially solvable shop problems with a limited number of machines or jobs .

Thesis Examination Committee

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

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

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

Graduate College Representative, Dr. Venkatesan Muthukumar, Ph.D.