

5-1-2013

## Simulated Annealing Approach To Flow Shop Scheduling

Sadhana Yellanki

University of Nevada, Las Vegas, yellanki@unlv.nevada.edu

Follow this and additional works at: <https://digitalscholarship.unlv.edu/thesesdissertations>



Part of the [Computer Sciences Commons](#), and the [Discrete Mathematics and Combinatorics Commons](#)

---

### Repository Citation

Yellanki, Sadhana, "Simulated Annealing Approach To Flow Shop Scheduling" (2013). *UNLV Theses, Dissertations, Professional Papers, and Capstones*. 1911.

<https://digitalscholarship.unlv.edu/thesesdissertations/1911>

This Thesis is protected by copyright and/or related rights. It has been brought to you by Digital Scholarship@UNLV with permission from the rights-holder(s). You are free to use this Thesis in any way that is permitted by the copyright and related rights legislation that applies to your use. For other uses you need to obtain permission from the rights-holder(s) directly, unless additional rights are indicated by a Creative Commons license in the record and/or on the work itself.

This Thesis has been accepted for inclusion in UNLV Theses, Dissertations, Professional Papers, and Capstones by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact [digitalscholarship@unlv.edu](mailto:digitalscholarship@unlv.edu).

SIMULATED ANNEALING APPROACH  
TO FLOW SHOP SCHEDULING

by

Sadhana Yellanki

Bachelor of Technology, Information Technology  
Jawaharlal Nehru Technological University, India  
2011

A thesis submitted in partial fulfillment  
of the requirements for the

**Master of Science in Computer Science**

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

**University of Nevada, Las Vegas  
May 2013**

© Sadhana Yellanki, 2013  
All Rights Reserved



## THE GRADUATE COLLEGE

We recommend the thesis prepared under our supervision by

Sadhana Yellanki

entitled

Simulated Annealing Approach to Flow Shop Scheduling

be accepted in partial fulfillment of the requirements for the degree of

**Master of Science in Computer Science**

School of Computer Science

Wolfgang Bein, Ph.D., Committee Chair

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

Ju-Yeon Jo, Ph.D., Committee Member

Venkatesan Muthukumar, Ph.D., Graduate College Representative

Thomas Piechota, Ph.D., Interim Vice President for Research &  
Dean of the Graduate College

**May 2013**

## ABSTRACT

### **Simulated Annealing Approach to Flow Shop Scheduling**

by

Sadhana Yellanki

Dr. Wolfgang Bein, Examination Committee Chair

Professor of Computer Science

University of Nevada, Las Vegas

Flow Shop Scheduling refers to the process of allotting various jobs to the machines given, such that every job starts to process on a machine  $n$  only after it has finished processing on machine  $n-1$ , with each job having  $n$  operations to be performed one per machine. To find a schedule that leads to the optimal utilization of resources, expects the schedule to finish in a minimum span of time, and also satisfy the optimality criterion set for the related scheduling problem is NP-Hard, if  $n > 2$ . In this thesis, we have developed an algorithm adopting a heuristic called Simulated Annealing, to act as a support to the Flow Shop Scheduling. This algorithm tries to deliver good/near optimal solutions to the given scheduling problem, in a reasonable time. We also carry out various tests to determine the behavior of the algorithm as well as to evaluate its effectiveness.

## **ACKNOWLEDGEMENTS**

I would like to express my sincere gratitude to my committee chair, Dr. Wolfgang Bein for his strong support and guidance throughout my thesis. It gives me great pleasure in acknowledging his valuable assistance.

I would also like to thank Dr. Ajoy K. Datta, Dr. Ju-Yeon Jo and Dr. Venkatesan Muthukumar for being a part of my committee. I convey my special thanks to the graduate coordinator, Dr. Ajoy K. Datta for all his support and advocacy. I extend my gratitude to the Computer Science department for funding my Master's degree.

I must offer my thanks to all my friends for being helpful all the time. Last but not the least important, I owe more thanks to my family for their immense love, support and encouragement.

**TABLE OF CONTENTS**

**ABSTRACT..... iii**

**ACKNOWLEDGEMENTS .....iv**

**TABLE OF CONTENTS ..... v**

**LIST OF TABLES ..... vii**

**LIST OF FIGURES ..... viii**

**CHAPTER 1 INTRODUCTION .....1**

    1.1 Outline..... 1

**CHAPTER 2 BACKGROUND AND LITERATURE .....2**

    2.1 Scheduling.....2

        2.1.1 Scheduling Problems .....2

        2.1.2 Terminology in Scheduling.....3

    2.2 Classes of Scheduling .....4

        2.2.1 Machine Environment.....4

        2.2.2 Job Characteristics .....8

        2.2.3 Optimality Criterion.....10

    2.3 Flow Shop Scheduling vs. Job Shop Scheduling.....12

        2.3.1 Example for Flow Shop Scheduling .....12

        2.3.2 Example for Job Shop Scheduling.....15

    2.4 Permutation Flow Shop Scheduling .....17

        2.4.1 PFSP for more than 2 machines.....18

        2.4.2 PFSP for 2 machines.....19

        2.4.3 Johnson’s Algorithm.....19

    2.5 Simulated Annealing.....21

        2.5.1 Acceptance Probability .....23

        2.5.2 Cooling Schedule .....23

        2.5.3 Simulated Annealing Algorithm.....24

|  |           |
|--|-----------|
| <b>CHAPTER 3 IMPLEMENTATION</b> .....            | <b>26</b> |
| 3.1 Problem Statement .....                      | 26        |
| 3.2 Approach followed towards the solution ..... | 26        |
| 3.3 Parameters of the Algorithm.....             | 27        |
| 3.3.1 Input .....                                | 27        |
| 3.3.2 Notations .....                            | 28        |
| 3.3.3 Assumptions.....                           | 30        |
| 3.3.4 Operations .....                           | 30        |
| 3.4 An Example .....                             | 31        |
| <b>CHAPTER 4 FINDINGS</b> .....                  | <b>36</b> |
| 4.1 A Classic Example.....                       | 36        |
| 4.2 Conducted Experiments .....                  | 37        |
| 4.2.1 Test1.....                                 | 38        |
| 4.2.2 Test2.....                                 | 40        |
| 4.3 Comparison with Taillard’s Instances .....   | 42        |
| 4.3.1 Results produced by the Algorithm .....    | 42        |
| 4.3.2 Conclusion .....                           | 45        |
| <b>CHAPTER 5 CONCLUSION AND FUTURE WORK.....</b> | <b>46</b> |
| <b>BIBLIOGRAPHY.....</b>                         | <b>47</b> |
| <b>VITA .....</b>                                | <b>49</b> |



## LIST OF TABLES

|  |    |
|--|----|
| 2.1 Example to demonstrate Flow Shop Scheduling .....          | 12 |
| 2.2 Example to demonstrate Job Shop Scheduling .....           | 15 |
| 2.3 Example to demonstrate Johnson's Algorithm .....           | 20 |
| 3.1 Example to demonstrate the behavior of our Algorithm ..... | 31 |
| 4.1 A Classic Example .....                                    | 36 |
| 4.2 Processing times for Test 1 and Test 2 .....               | 38 |
| 4.3 Results for Test 1 .....                                   | 39 |
| 4.4 Results for Test 2 .....                                   | 41 |
| 4.5 Instances of 5 jobs and 20 machines .....                  | 43 |
| 4.6 Instances of 10 jobs and 20 machines .....                 | 44 |
| 4.7 Instances of 20 jobs and 20 machines .....                 | 45 |

## LIST OF FIGURES

|  |    |
|--|----|
| 2.1 To demonstrate Flow Shop Scheduling .....              | 13 |
| 2.2 To demonstrate Job Shop Scheduling .....               | 15 |
| 2.3 Optimal Schedule.....                                  | 21 |
| 3.1 Input to the Algorithm .....                           | 32 |
| 3.2 Iterative Improvement.....                             | 33 |
| 4.1 Optimal Schedule for the Classic Example .....         | 37 |
| 4.2 Iterative Improvement by the Algorithm for Test 1..... | 40 |
| 4.3 Iterative Improvement by the Algorithm for Test 2..... | 42 |

# CHAPTER 1

## INTRODUCTION

Scheduling is a part of daily routine. We plan the daily activities with respect to time, such that we utilize our time in an optimal manner. In this thesis, we are going to discuss Flow Shop Scheduling Problem which deals with scheduling and we make an attempt to obtain improved solutions to the given problem. According to this problem, given a set of resources called as machines and a set of jobs which are required to be assigned to their resources, in an optimal manner [3]. It has to be ensured that the resources should be used efficiently. This problem seems to be simple, but turns out to be an NP-Hard problem, if the number of resources is more than 2. So, we develop an algorithm, and conduct various tests to verify how far it gives us good solutions to the problem considered.

### 1.1 Outline

Chapter 2 discusses in detail about the basics of scheduling, problems with scheduling and Permutation Flow Shop Scheduling. Chapter 3 discusses about the approach followed towards the solution, and the algorithm implemented to solve the problem considered. Chapter 4 demonstrates the behavior of the algorithm as well as lists out the results produced. Chapter 5 gives the conclusion and the improvements that can be made in the future.

## CHAPTER 2

### BACKGROUND AND LITERATURE

This chapter gives an insight about Scheduling, its problems and various other fundamental aspects of scheduling.

#### 2.1 Scheduling

Scheduling is a branch of science that deals with performing an effective arrangement of a given set of tasks, on the given resources such that the resources are allotted to them in an optimal manner.

Though this seems to be simple, there are various kinds of problems associated with scheduling. They have a history of 50 years, and hence it has a long way to go. Among the scheduling problems, some of them are a subset of the NP-Hard problems [1]. These problems are discussed in detail in the next section.

##### 2.1.1 Scheduling Problems

According to Peter Brucker [2], a Scheduling problem can be defined as a problem, where given a set of  $m$  machines  $M_j (j=1,2,..,m)$  or  $M_j \in \{M_1, M_2, \dots, M_m\}$  and  $n$  jobs  $J_i (i=1,2,..,n)$  or  $J_i \in \{J_1, J_2, \dots, J_n\}$  and the task of scheduling is to find a schedule i.e., the sequence to be followed by the jobs when they run on a machine.

A job will start processing on machine  $n$  only after it finishes its processing on the machine  $n-1$ . Also, all the jobs visit the machines in the same order. It should be ensured that no job processes on more than machine at the same instance of time. In such case, if each job has  $m$  number of operations associated with it, it is termed as Flow Shop Scheduling.

A parameter is chosen for every Flow Shop problem considered, which has to be minimized by the schedule chosen. This parameter is called as the Optimality Criterion.

In Flow Shop, obtaining an optimal schedule which achieves the optimality criterion set for the problem may not happen in a reasonable time, when the problem includes more than 2 machines. Under such circumstances, we call it as an NP-Hard problem. Apart from the case mentioned above, there are some special cases where Flow Shop Scheduling is non NP-Hard, and they are discussed in the later sections.

### **2.1.2 Terminology in Scheduling**

#### **Job**

A Job is as a set of tasks also called as operations, where each operation has a time interval corresponding to every particular machine on which it has to process. Each job can be represented as  $J_i$  and each operation as  $O_{ij}$ , where  $i$  denotes the job and  $j$  denotes the machine. This notation is followed throughout the document.

#### **Processing time**

Processing time can be defined as the amount of time for which, an operation of a job processes on a machine. It is represented as  $p_{ij}$  for an operation  $O_{ij}$ . Always,

$$p_{ij} \geq 0$$

#### **Idle time**

Idle time can be defined as the interval of time for which the machine is idle, without any job being processed on it.

## **Makespan**

The time at which the last job in the schedule is completed on the last machine is referred to as the Makespan of the schedule and is denoted as  $C_{max}$ . It is the time by which all the jobs in the schedule complete all their operations [23].

The makespan of a schedule whose value is minimal among the makespans of all the possible schedules for the given problem is the optimal makespan.

## **Completion time**

The amount of time taken by a job  $J_i$  to complete its processing on the last machine is termed as the Completion time of job  $J_i$  denoted as  $C_i$ .

The sum of completion times  $\sum C_i$  of a schedule whose value is minimal among the  $\sum C_i$  values of all the possible schedules for the given problem is the optimal  $\sum C_i$ .

## **2.2 Classes of Scheduling**

Based on the attributes associated with the jobs and the machines, Scheduling is classified into various categories. According to Peter Brucker [2], the different classes are represented in the form of a  $\alpha|\beta|\gamma$  notation where,  $\alpha$  denotes the machine environment,  $\beta$  denotes the job characteristics and  $\gamma$  denotes the optimality criterion for the schedule.

### **2.2.1 Machine Environment**

Different types of machines are available that can be used in the scheduling environment, with each of them serving their own purpose. The type of the machine is based on the properties associated with it, and all together constitute to the machine environment. It is denoted by a string  $\alpha$ , and this string has two parts  $\alpha_1, \alpha_2$

each with a specific meaning. In the string  $\alpha$ ,  $\alpha_2$  belongs to the number of machines used in scheduling.

$\alpha_1$  belongs to the set  $\{o, P, Q, R, PMPM, QMPM, G, X, O, J, F\}$  with each element in the set, associated with a different functionality. Let us go through all the cases. Each case is concerned with the way in which, the operations of a job are performed in a schedule. Each operation  $O_{ij}$  is associated with a set of machines represented as  $\mu_{ij} \subseteq \{M_1, \dots, M_m\}$  where  $m$  denotes the number of machines.  $O_{ij}$  may be processed on any of the machines that belong to  $\mu_{ij}$ . This property varies upon the type of machines chosen [2].

**Case 1**  $\alpha_1=o$ , we have  $\alpha = \alpha_2$ . It is because,  $o$  represents an empty symbol. Here, each job has only one operation, and it is processed only on a particular machine and hence it is called a Dedicated machine environment.

**Case 2**  $\alpha_1 \in \{P, Q, R\}$ , then the jobs are said to run on Parallel machines i.e., in contrast to the above case where a job is dedicated to only to a particular machine to process, here a job can have more than one operation, and each job can be processed on all the machines in the set  $M_1$  to  $M_m$ . We subdivide this case into three parts based on whether it belongs to  $P$  or  $Q$  or  $R$  [3].

- If  $\alpha_1 = P$ , then scheduling is performed on Identical Parallel Machines where the processing times of a job are identical for all the machines in  $\mu_{ij}$  i.e., a job processes on every machine for the same amount of time [21].

Here, we have,  $p_{ij} = p_i$  for each element in  $\mu_{ij}$ .

- If  $\alpha_1 = Q$ , then scheduling is performed on Uniform Parallel Machines. Here, each machine is associated with a specific speed, and the speed of machine is

uniform for all the jobs processing on the same machine. In other terms, if  $p_{ij}$  is the processing time of job  $J_i$  and  $s_j$  is the speed of a machine  $j$  then, the amount of time this job processes on this machine is  $p_i/s_j$ , and it is same for all the jobs on machine  $M_j$  [21].

- If  $\alpha_I = R$ , then the scheduling is said to be performed on Unrelated Parallel Machines. In contrast to the above scenario, where every machine is associated with a specific speed, here every job is associated with a specific speed for each machine. In other terms, if  $p_{ij}$  is the processing time for a job  $J_i$  on machine  $M_j$ , then the value of  $p_{ij}$  is equal to  $p_i/s_{ij}$  where,  $s_{ij}$  is the speed associated for a particular job on a particular machine.

**Case 3**  $\alpha_I = PMPM$  or  $QMPM$ , then the machines are called the Multipurpose Machines [4]. *PMPM* refers to the machines where each job has an identical speed on all the machines in the scheduling. The speed of one job may vary from another. *QMPM* refers to the machines that have same speed associated for all the jobs being performed on the same machine. This speed might vary from machine to machine.

**Case 4**  $\alpha_I \in \{G, X, O, J, F\}$ , then the scenario would be having a set of operations for every job, where each operation is performed on a specific machine.

- If  $\alpha_I = G$ , it represents General Flow Shop. In this case, there would be an order to be followed by the operations in every job, for processing on a machine. This refers to a precedence relationship among the operations. According to this relationship, when each operation is associated with a certain priority, the one with the higher priority processes first than, the one with the lower priority. When certain conditions are altered in General Flow



Shop, it will result in other kinds of scheduling problems like Open Shop, Job Shop, Flow Shop and Mixed Shop.

- If  $\alpha_I = J$ , it denotes Job Shop. In this case, for each job having a set of operations represented as  $O_{ij}$ , there is a relationship maintained between the operations which is called as Precedence. Precedence relationship can be denoted by the  $\rightarrow$  symbol. If we have  $O_{i1} \rightarrow O_{i2}$ , then  $O_{i2}$  has to start processing, only after  $O_{i1}$  is finished.

This relationship is continued in between all the operations  $O_{ik}$  where  $k$  denotes number of operations in job  $J_i$  [5]. Also, in this case, each job follows its own route of visiting the machines.

- If  $\alpha_I = F$ , we call it as Flow Shop. This is closely similar to Job Shop and the only difference here from the above case is that, the number of operations in a job is equal to the number of machines i.e. we have  $k=m$ . We have one operation processed per machine, and so when all the operations of a job are put together, they cover all the machines in the set  $\mu_{ij}$  [6]. In this case, all the jobs maintain the same machine sequence while visiting the machines, in the process of Scheduling. The job sequence may vary from machine to machine.

In Flow shop, if the job sequence remains the same for all the machines, it is called as a Permutation Flow Shop. Job Shop and Flow Shop problems are discussed in detail in Section 2.3.

- If  $\alpha_I = O$ , where  $O$  denotes Open Flow Shop, this is again related to Flow Shop but with a special condition associated with it. In this case, we do not

have any priorities assigned to the operations of a job, and so there is no order to be followed in between the operations of a job when it does its processing [7].

### 2.2.2 Job Characteristics

This attribute refers to the types of characteristics with which a job is associated. It is represented by  $\beta$ , and it is a set containing six characteristics which are denoted as  $\beta \in \{\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6\}$ . Each element of the set represents a different property. So, this can be divided into six cases.

**Case 1**  $\beta = \beta_1$ , then we say the jobs are preempted.  $\beta_1$  represents preemption and so,  $\beta_1 = pmtn$ . Preemption refers to a scenario, where the jobs scheduled are allowed to undergo interrupts. A particular start time and finish time are associated with every task that is/wants to be a part of the schedule. If preemption property exists for the jobs, only then  $\beta_1$  is present in the set  $\beta$ .

**Case 2**  $\beta = \beta_2$ , then we say that the jobs have a precedence relationship among them and  $\beta_2 = prec$ . As mentioned earlier, it refers to a relation where jobs are given some priorities, and based on which the job with higher priority gets scheduled first before the one with the lower priority. This precedence relationship will be maintained throughout the schedule. There are various other ways in which the jobs can be arranged. Some of them are Intree, Outtree, Chains and sp-graphs.

- If the structure is a tree with every node of the tree having 0 or 1 outgoing node connections, then it is an Intree. Otherwise, if every node in the tree has 0 or 1 incoming node connections then, it is an Outtree. We represent it as  $\beta_2 = Intree$  or  $\beta_2 = Outtree$ , based on the type of the tree. If the structure is

a Chain i.e., a tree in which every node is connected to only 0 or 1 other nodes, then we have  $\beta_2 = chain$ .

- If it is a sp-graph or Series-Parallel graph [8], then  $\beta_2 = sp-graph$ . This structure is a Graph say  $G_i = (V_i, A_i)$  where  $i \in \{1,2\}$ ,  $V$  represents the vertices and  $A$  represents the arcs, where it can have either a single vertex called as a Base Graph or it can be a mix of 2 different graphs.

If it is a composition of two graphs, and they are joined such that, the vertices and arcs of one graph are joined to the vertices and arcs of the other, then we call it as Series Composition. The other possible way of joining the two graphs is not only joining the vertices and arcs as before, but also linking the leaf nodes of one graph to the source nodes of another graph and we call it as Parallel Composition [2].

If the jobs maintain one among the structures mentioned above, only then we have  $\beta_2$  present in the set  $\beta$ .

**Case 3**  $\beta = \beta_3$ , then we say that a release date  $r_i$  is associated with every job, which represents the time at which a job is released i.e., the time at which a job's first operation is ready to be processed.  $r_i$  denotes the release date of a job  $i$ . If we have a release dates associated with jobs, only then,  $\beta_3$  is present in the set  $\beta$ .

**Case 4**  $\beta = \beta_4$ , then we say that the jobs are linked with processing times restriction i.e., if we have all the processing times of the operations of every job equal to 1 i.e.  $p_i = 1$  then, we may call it as a unit processing requirement. The restriction can also be of the form  $p_i = k$  where  $k$  is a constant value. If this restriction property exists for the jobs, only then  $\beta_4$  is present in the set  $\beta$ .

**Case 5**  $\beta = \beta_5$ , then we say that all the jobs are associated with a specific deadline  $d_i$ , which represents the time by which that particular job has to finish its processing on a machine. We represent it as  $\beta_5 = d_i$ . If deadlines exist for the jobs, only then  $\beta_5$  is present in the set  $\beta$ .

**Case 6**  $\beta = \beta_6$ , then we say that the jobs are processes as batches. Batching can be defined as a set of jobs which are grouped together to be scheduled back to back without any set up time in between them.

There are two kinds of batching problems. If the length of the batch is considered to be the maximum time, among the set of processing times of the jobs in a batch, it is called as p-batching. Otherwise, if the length of the batch is calculated as, the sum of the processing times of all the jobs in a batch, it is called as s-batching. We have  $\beta_6 = p\text{-batch}$  or  $\beta_6 = s\text{-batch}$  depending upon the type of the batching problem considered. If batching property exists for the jobs, only then  $\beta_6$  is present in the set  $\beta$ .

### 2.2.3 Optimality Criterion

There will be a cost function associated with every scheduling problem, and the goal would be to reduce the cost of the schedule associated with, minimizing the optimality criterion of the problem considered. This parameter may vary from one problem to another, and it is represented by  $\gamma$ . Some cases with some of the possible optimality criteria that could be considered are as follows.

**Case 1** This case refers to minimizing the total time taken by the last job to finish, on the last machine in the schedule. The function associated with this parameter is called as a Bottleneck objective function.

**Case 2** This case refers to minimizing the sum of completion times of all the jobs in a schedule. The function associated with this parameter is called as a Sum objective function.

There can be many other parameters which can be used in an objective function. Some of them are related to the parameters like Lateness, Earliness, Tardiness, Deviations and Penalties [9].

### **Lateness**

The amount of time by which a job is late in finishing its processing is termed as Lateness. If  $C_i$  denotes the completion time of a job  $J_i$ , then the difference between the  $C_i$  value and the due date  $d_i$  is calculated as the Lateness.  $L_i$  represents lateness of a job  $J_i$ . So we have,

$$L_i = C_i - d_i$$

### **Earliness**

The amount of time by which, a job is early in finishing its processing is termed as Earliness. If  $C_i$  denotes the completion time of processing of a job  $J_i$ , then the difference between  $d_i$  and  $C_i$  is calculated as the Earliness.  $E_i$  represents earliness of a job  $J_i$ . We also have,

$$E_i = d_i - C_i, \text{ if } E_i > 0$$

$$E_i = 0, \text{ if } E_i \leq 0$$

### **Tardiness**

Tardiness is same as lateness except for the condition that, it is true only when lateness is positive, otherwise tardiness is 0. It is represented as  $T_i$  for a job  $J_i$ .

## Deviations

There are two kinds of deviations. A deviation, which is the square of the lateness value, is stated as *Squared deviation*  $S_i$ .

If the deviation is calculated as the absolute value of lateness, it is termed as *Absolute deviation* denoted as  $D_i$ .

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

$$D_i = |C_i - d_i|$$

## Unit Penalty

This value pertains to the value of lateness. If the lateness value of a particular job  $J_i$  is greater than 0, then the penalty associated with this job is 1 otherwise it is 0. Unit penalty is represented by  $U_i$  for a job  $J_i$ .

### 2.3 Flow Shop vs. Job Shop Scheduling

We have discussed both the Flow Shop Scheduling and the Job Shop Scheduling along with their properties. Let us go through an example to demonstrate the differences between these two scheduling problems [10].

#### 2.3.1 Example for Flow Shop Scheduling Problem

In this example, consider the schedule as  $J_1-J_2$  on  $M_1$ , and  $J_2-J_1$  on  $M_2$  for the example presented in the Table 2.1.

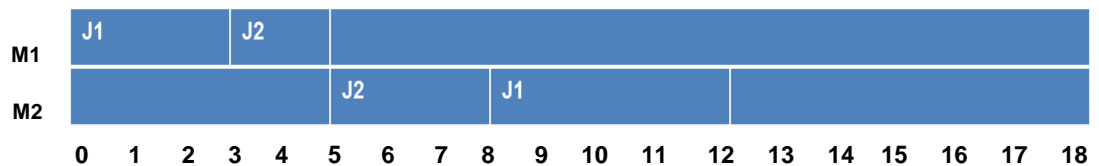
|    | M1 | M2 |
|----|----|----|
| J1 | 3  | 4  |
| J2 | 2  | 3  |

**Table 2.1 Example to demonstrate Flow Shop Scheduling**

In Flow Shop Scheduling, the job schedule may vary from machine to machine but, the machine sequence followed by the all the jobs should remain the same. In this example, the machine sequence followed is  $M_1-M_2$  and the set up times of the jobs are included in their respective processing times.

The way the jobs are organized on different machines over time is represented using a Gantt chart. Gantt chart provides the visual representation of this arrangement. There are two kinds of Gantt charts namely, Machine-Oriented Gantt chart and Job-Oriented Gantt chart. The horizontal axis of the chart represents the time frame and this common in both the charts. If the vertical axis represents the machines, it is a Machine-Oriented Gantt chart, and if it represents the jobs, it is a Job-Oriented Gantt chart.

Let us consider a Machine-Oriented Gantt Chart for our schedule. So, the Gantt chart for the concerned problem is represented in Figure 2.1.



**Figure 2.1 To demonstrate Flow Shop Scheduling**

Following steps are used to demonstrate the Flow Shop Scheduling procedure.

1. At time  $t=0$ , on the machine  $M_1$ ,  $J_1$  is ready for execution, and  $J_2$  executes only after  $J_1$  is done.  $J_1$  runs for 3 time units on  $M_1$ .

2. At  $t=3$ ,  $J_2$  runs on  $M_1$  for 2 units of time, and now it starts its execution on  $M_2$ , and runs for 3 time units.  $J_1$  waits for  $J_2$  to complete on  $M_2$  because; the job sequence on  $M_2$  is  $J_2-J_1$ .
3. At  $t=8$ ,  $J_2$  is done, and  $J_1$  starts running on  $M_2$  for 4 time units and will finish at time  $t=12$ .

### **Calculating $C_{max}$**

This procedure remains same for all kinds of scheduling problems. For the example mentioned in Table 2.1, the makespan value can be calculated as follows.

From Figure 2.1, it can be observed that, the last job to be schedule is  $J_1$ , and it does its last operation on the last machine  $M_2$ . As defined earlier, the finishing time of the last operation of last job  $J_1$ , is the makespan of the schedule. The time taken by  $J_1$  to complete all its operations is 12 units.

So, for this schedule, we have  $C_{max} = 12$ .

### **Calculating $\sum C_i$**

The procedure to calculate the  $\sum C_i$  can be described using the same example and referring to the Figure 2.1.

To obtain the  $\sum C_i$  value, calculate the completion times of every job on the last machine, i.e. the time by a job has completed all the operations on all the machines.

Completion time of  $J_1 = 12$

Completion time of  $J_2 = 8$

Average completion time =  $\sum C_i$  / number of jobs

In this example, we have  $\sum C_i$  / njobs =  $(20/2) = 10$



### 2.3.2 Example for Job Shop Scheduling Problem

Let us consider an example presented in Table 2.2 to gain a brief understanding of Job Shop Scheduling problem.

As said earlier, in Job Shop Scheduling, the order of visiting the machines may differ from one job to another.

|    |       |       |
|----|-------|-------|
| J1 | M1(5) | M2(8) |
| J2 | M2(4) | M1(2) |
| J3 | M1(6) | M2(6) |

**Table 2.2 Example to demonstrate Job Shop Scheduling**

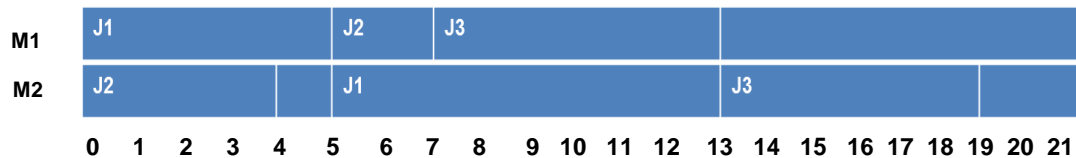
It can be noticed from the Table 2.2 that, the machine sequence is varying from one job to another.

Machine sequence for  $J_1$  is  $M_1-M_2$ .

Machine sequence for  $J_2$  is  $M_2-M_1$ .

Machine sequence for  $J_3$  is  $M_1-M_2$ .

Now, let us design a Gantt chart for this problem so as to analyze this method of scheduling.



**Figure 2.2 To demonstrate Job Shop Scheduling**

Following steps guide the construction of Gantt chart in Job Shop Scheduling.

1. At time  $t=0$ , we have jobs  $J_1$  and  $J_3$  competing for  $M_1$ . According to the Shortest Processing Time rule [22], when there are two different jobs waiting for the same machine at the same time, then choose the job with the shortest processing time among them. So, according to this, we chose  $J_1$  to be scheduled on  $M_1$ .
2. Also at  $t=0$ , we have  $J_2$  waiting to be processed on  $M_2$ , and as it is the only job waiting to be processes on this machine at this point of time, it is right away scheduled.
3. At  $t=4$ ,  $J_2$  finishes it's processing on  $M_2$  and starts waiting for  $M_1$ .
4. At  $t=5$ , we have jobs  $J_3$  and  $J_2$  waiting to process on  $M_1$ . According to the Shortest Processing Time rule mentioned before, we choose  $J_2$  because, on  $M_1$ , its processing time is less than that of  $J_3$ .
5. Also at  $t=5$ , we  $J_1$  completes its task on  $M_1$ , it jumps to  $M_2$  for its task on this machine. At this point of time,  $M_2$  is free and so,  $J_1$  is right away scheduled, and it processes on this machine for  $p_{12}$  time units.
6. At  $t=7$ ,  $J_2$  finishes its operation on  $M_1$ , and it is done with all its operations. So, the Completion time of  $J_2=7$ .
7. At  $t=7$ ,  $M_1$  becomes free, and we have a job  $J_3$  waiting to be scheduled on  $M_1$ . It is now scheduled on  $M_1$ , and will process for  $p_{31}$  time units.
8. At  $t=13$ ,  $J_1$  finishes its operation on  $M_2$ , and it is done with all its tasks. So, the Completion time of  $J_1=13$ .

9. At  $t=13$ ,  $J_3$  finishes its operation on  $M_1$ , and it jumps to  $M_2$  for its next operation for  $p_{32}$  time units. At  $t=19$ , it finishes all its tasks. So, the Completion time of  $J_3=19$ .

Finally, the  $C_{max}$  and  $\sum C_i$  values for the above schedule are as follows

$$C_{max} = 19.$$

$$\sum C_i = (13+7+19)/3 = 13.$$

## 2.4 Permutation Flow Shop Scheduling Problem

The objective of the algorithm implemented in this thesis, is to focus on the problem of optimizing the results for Permutation Flow Shop Scheduling (PFSP). For more than 2 machines, it is an NP-Hard problem. Through this algorithm, we try to obtain a near optimal solution to this problem. Before going into the details, we will discuss the basic definition of PFSP [11].

Permutation Flow Shop Scheduling is a special case of Flow Shop Scheduling. In a Flow Shop Scheduling problem, the job sequence might vary from machine to machine, and this is called a Regular Flow Shop Problem. PFSP is closely related to Flow Shop with an additional condition associated with it i.e., the job schedule remains same for all the machines.

In this kind of scheduling, various permutations are performed with the given jobs to obtain different schedules. Each permutation is denoted by  $\pi$ . If the given number of jobs is  $n$ , then the possible number of permutations with them is  $n!$  [12].

Whether Flow Shop Scheduling problems are NP-Hard or not depends upon the number of machines used in the scheduling. They are said to be NP-Hard only if the number of machines  $m$  is more than 2 [13]. If the number of machines are less than

or equal to 2, it is solvable in polynomial time. In a special case, where all the processing times of all the jobs are same, it is not an NP-Hard problem irrespective of the number of machines.

#### **2.4.1 PFSP for more than 2 machines**

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

- (i) The job sequence on the first two machines is the same.*
- (ii) The job sequence on the last two machines is the same.*

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

In PFSP, obtaining the optimal job sequence by performing all the possible permutations among the jobs, and checking which schedule gives the optimal results is not possible in a reasonable time if  $m > 2$ .

The algorithm we have implemented, tries to reach a near optimal solution for the given Flow Shop problem, by adopting a heuristic called Simulated Annealing. This algorithm is discussed in Chapter 3, and the concept of Simulated Annealing is discussed later in Section 2.5.

According to the  $\alpha|\beta|\gamma$  – notation, the algorithm mentioned above is designed to solve a problem which can be represented as  $Fm|prmu|C_{max}$ , if makespan is the optimality criterion for the schedule. If we consider the optimality criterion as minimizing the sum of completion times, then the problem can be denoted as  $Fm|prmu|\sum C_i$  [14]. In this notation,  $F$  represents flow shop problem  $m$  represents

the number of machines and  $prmu$  is an abbreviation representing the job constraints on the permutation of the jobs.

The above two problems are strongly NP-Hard for  $m \geq 3$ .

#### **2.4.2 PFSP for 2 machines**

PFSP is not NP-Hard for 2 machines, and also for the special case mentioned earlier i.e., when the processing times of all the jobs are equal. When we have identical processing times for all the jobs on all the machines, it is non NP-Hard, because whatever may be the job sequence, all the schedules come up with the same results, and every schedule is an optimal solution.

According to Theorem 3.6 [2], *“For the flow-shop problem  $F2||\sum C_i$  there exists an optimal schedule in which both machines process the jobs in the same order”*.

There are some algorithms existing to solve and produce an optimal solution to PFSP for 2 machines, and one among them is the Johnson’s Algorithm. It produces an optimal solution for the problem  $F2||C_{max}$ . This algorithm is explained in the next section.

#### **2.4.3 Johnson’s Algorithm**

Johnson’s algorithm gives an optimal solution to the  $F2||C_{max}$  problem, and here all the jobs are scheduled in the same order for both machines [15]. In the algorithm below,  $T$  represents the final schedule which is same on both the machines.

If a job  $i$  is found with least processing time on machine  $j$ , it is either concatenated with the string  $L$  or string  $R$ , if  $j$  is from  $M_1$  or  $M_2$  respectively.  $L$  and  $R$  hold null values initially and produce the final schedule  $T$ , when they are concatenated. This is the algorithm presented in Algorithm 2.1.

---

**Algorithm 2.1 Johnson's Algorithm**

---

1. Consider two strings  $L$  and  $R$  and let  $S$  be the set of jobs.
  2. Find the minimum processing time among all the jobs say  $p_{ij}$  and  $i \in S$ .
  3. If  $j = 1$ , add  $i$  to the tail of  $L$ .
  4. Else if,  $j = 2$ , add  $i$  to the front of  $R$ .
  5. Eliminate job  $i$  from the set  $S$ .
  6. If  $S \neq \text{empty}$ , go to (2).
  7. Concatenate  $L$  and  $R$  and assign it to  $T$ .
  8. Stop.
- 

**Example**

Initially let  $X = \{1...i...n\}$  be the set of all jobs.

The example in Table 2.2 is used to show how Johnson's algorithm works for a set of 4 jobs on 2 machines. The optimal schedule is presented in Figure 2.3.

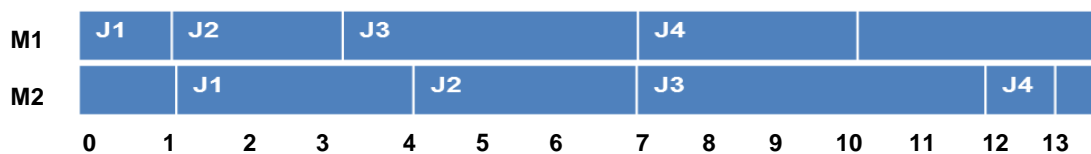
| $i \backslash j$ | $p_{i1}$ | $p_{i2}$ |
|------------------|----------|----------|
| 1                | 1        | 3        |
| 2                | 2        | 3        |
| 3                | 4        | 5        |
| 4                | 3        | 1        |

**Table 2.3 Example to demonstrate Johnson's Algorithm**

1. Let  $X = \{J_1, J_2, J_3, J_4\}$ . At the initial step, the minimum processing time found is 1, and this value exists twice in the schedule, as we have both  $p_{11}=1$  and also  $p_{42}=1$ .  
We can consider any one job among them, and let us choose  $p_{11}$ . So, concatenate  $J_1$  to string  $L$ , and eliminate  $J_1$  from  $X$ . So,  $L = "J_1"$ .
2. Now, consider  $p_{42}$  and concatenate  $J_4$  to  $R$ , and then eliminate it from  $X$ .  
So,  $R = "J_4"$ .
3. The next minimum value found is  $p_{21} = 2$ . Concatenate  $J_2$  to  $L$ , and then eliminate it from  $X$ . So,  $L = "J_1 J_2"$ .
4. The next minimum value found is  $p_{31} = 3$ . Concatenate  $J_3$  to  $L$  and then eliminate it from  $X$ . So,  $L = "J_1 J_2 J_3"$ .

So finally, the schedule is the concatenation of  $L$  and  $R$  and here we have the sequence as " $J_1 - J_2 - J_3 - J_4$ " is the optimal schedule in this example.

The Machine-Oriented Gantt chart for this example can be represented as below in Figure 2.3.



**Figure 2.3 Optimal Schedule**

## 2.5 Simulated Annealing

Simulated annealing is a heuristic which could be adopted by Combinatorial Optimization Problems like Flow Shop Scheduling.

According to Rob A. Rutenbar [16], the basic idea of this technique has been inspired from a real time example, the process of annealing. Annealing can be defined as a process of heating a metal until it reaches a critical temperature i.e., the temperature at which the metal starts melting. As soon as it happens, the metal is set to cool down step by step, by exposing it to various temperatures. This is done to improve some of the characteristics of that metal.

Analogous to the above example, Simulated Annealing is referred to as a generic heuristic, applied to a problem by starting with a random solution in a solution space. It continues to search in this solution space for a better solution at its each iteration, and every time when a new solution is found, it will be accepted or rejected based upon the fulfillment of a criterion adopted by the problem, by the new solution.

These iterations are continued to be performed until the algorithm reaches a terminating point [17]. Simulated Annealing is just a heuristic or technique that can be applied to the combinatorial optimization problems. It is just a strategy which could help in obtaining a good solution which may be near optimal, in a feasible time. They cannot guarantee to produce an optimal solution.

This procedure is continued until it comes out with a solution, which is closer to the near optimal solution. There are some parameters which are a part of this heuristic, and they are the Cooling schedule and the Acceptance probability.

The concept of Acceptance probability [18] is explained in the Section 2.5.1 and the purpose of the Cooling schedule [19] is described in Section 2.5.2.



### **2.5.1 Acceptance probability**

The criterion used by the Simulated Annealing technique, when it has to decide whether or not to accept a newly found solution, that it will come across at its every new iteration during the search, is the Acceptance probability.

The formula used for calculating the acceptable probability is

$$P = e^{-\Delta E/T}$$

$\Delta E$  refers to the change of energy, which can be a gain or loss resulted in moving from one solution to another during the iteration.

$T$  refers to the current temperature value, it is a cooling constant which is one among the set of values belonging to the cooling schedule.

If there is a loss of energy, a new probability value is generated at current iteration. It is checked whether this value belongs to the range of the Acceptable probability for the corresponding  $T$  value. If it is the range, the solution is accepted. Else, it is rejected.

### **2.5.2 Cooling Schedule**

Cooling schedule is the crucial part of this technique. The cooling schedule is a set of values called cooling constants, analogous to the set of the varying temperatures use to cool down the metal periodically, in the annealing process.

The values for the cooling schedule are inputted in such a way that, they go down periodically. If the values in the cooling schedule initially are very high, they allow the algorithm to start with a search for the solution, in a larger solution space.

Later on for the next values, its value keeps going down i.e., it narrows down the solution space to restrict and refine the searching process in further iterations [19].

So, the higher the cooling constant value, the larger is the solution space. As, the cooling constant keeps decreasing, the solution space shrinks gradually. It tries to lead the algorithm into a productive area in the space, where the scope of obtaining an optimal solution is high.

### **2.5.3 Simulated Annealing Algorithm**

As said earlier, Simulated Annealing is a technique which starts with a random solution and does an iterative improvement in a step by step process.

It tries to achieve a near optimal solution for the given problem. This algorithm is explained in Algorithm 3.2 [16].

#### **Notations**

Following are the notations used in this algorithm

$i$  – Initial solution and is chosen randomly at the beginning

$n$  – New solution

$\Delta E$  – Change of energy in moving from one solution to another

$P$  – Acceptance probability value

$p$  – Probability generated in current iteration

$T$  – Current temperature or a current cooling constant

$nitr$  – number of iterations and  $nitr > 0$

---

**Algorithm 2.2 Simulated Annealing**

---

1. Start with a random solution  $i$
  2. For each value of  $T$
  3. Loop until  $nitr > 0$
  4. Perform a new iteration and get  $n$
  5. Check if  $\Delta E$  is positive
  6. If true, go to(7)
  7. Accept  $n$  and decrement  $nitr$  and go to (3)
  8. Else, calculate  $P = 1/e^{(\Delta E/T)}$  and generate  $p$
  9. Check if  $p \in P$ , if true, go to (7)
  10. Else, discard  $n$ , decrement  $nitr$  and go to (3)
  11. End loop.
-

## CHAPTER 3

### IMPLEMENTATION

In this chapter, the first section describes about the Problem statement, the second section gives a brief view about the approach followed towards the solution and the rest of the chapter includes the in detailed discussion about the algorithm implemented.

#### 3.1 Problem Statement

Some of the Flow Shop Scheduling problems are NP-Hard problems, and PFSP is one among them. PFSP is a scheduling problem, where we are given a set of machines  $M_j$  for  $j = 1, 2, \dots, m$  and a set of jobs  $J_i$  for  $i = 1, 2, \dots, n$ , and a schedule is obtained by applying permutations to the given set of jobs such that, the schedule satisfies the optimality criterion. And, obtaining the optimal job sequence in polynomial time is NP-Hard for  $m > 2$ .

We have implemented an algorithm, which focuses on improvising the solutions for this NP-Hard problem, and tries to come up with a near optimal solution in a reasonable time.

#### 3.2 Approach followed towards the solution

The heuristic used here is Simulated Annealing which acts as a background technique to support PFSP. It helps to gain good and improved results to the given problem. Simulated Annealing is just a strategy and it doesn't guarantee the optimal solutions.

The near optimal solution here we refer to is a near optimal schedule. It is the order in which the jobs are scheduled on every machine such that the schedule

satisfies the optimality criterion. This algorithm tries to minimize the optimality criterion chosen for the problem, and it may refer to the makespan value  $C_{max}$  of the schedule or the sum of completion times  $\sum C_i$  of the schedule.

### **3.3 Parameters of the Algorithm**

#### **3.3.1 Input**

The algorithm receives a set of inputs, and tries to give in the best possible solutions. The basic input is the number of machines and the number of jobs to be scheduled and their processing times. The critical part of the algorithm is to obtain the best sequence of the jobs i.e., to find a best schedule. This algorithm does the iterations in a continuous manner and so, there should be a terminating point for the algorithm, to know when it has to stop. So, the input set also includes the number of iterations to perform. The algorithm has to always perform a check at its each new iteration, to verify that it is not exceeding the given number of iterations.

The rest of the input data belongs to the cooling schedule according to which, the algorithm alters its search area for the solutions throughout the iterations performed. The better the values of the cooling constants in the schedule, the better will be the output. If the values of these constants are high, then the search space is wide and vice versa. This input plays a key role as the behavior of the algorithm is grounded on this data. If it is a set of  $k$  values and  $l$  is the number of iterations, the algorithm would perform is  $l*k$  iterations in total because, it does  $l$  iterations for each value in the set.

The final input is the optimality criterion. The choice is to opt for one among, the makespan  $C_{max}$  or the sum of completion times  $\sum C_i$ , to be minimized by the algorithm.

### 3.3.2 Notations

Following are the notations used in the implementation of the algorithm.

$N_j$  – Number of jobs given

$N_m$  – Number of machines given

$N_{itr}$  – Number of iterations

$Ch$  – M, if  $C_{max}$  is the optimality criterion

– S, if  $\sum C_i$  is the optimality criterion

$Old\_seq$  – Job sequence of previous accepted solution

$Best\_seq$  – Best job sequence till the current iteration

$New\_seq$  – New job sequence

$\Delta mk_1 = C_{max}$  of  $New\_seq$  -  $C_{max}$  of  $Old\_seq$

$\Delta mk_2 = C_{max}$  of  $New\_seq$  -  $C_{max}$  of  $Best\_seq$

$\Delta \sum C_i 1 = \sum C_i$  of  $New\_seq$  -  $\sum C_i$  of  $Old\_seq$

$\Delta \sum C_i 2 = \sum C_i$  of  $New\_seq$  -  $\sum C_i$  of  $Best\_seq$

$Aprob$  – Generated Acceptance Probability

$Rprob$  – Random probability generated at current iteration

$N\_Sch$  – Number of values in the cooling schedule

$Cc$  – Set of cooling constants

$T$  - Current value from  $Cc$  at current iteration.

$T_{itr} = (N_{itr} * N_{sch})$

---

**Algorithm 3.1**

---

1. Choose  $Ch = M$  or  $S$
  2. For each value of  $T$  in  $Cc$
  3. Check  $N_{itr} > 0$ , if true go to (4), else go to (19)
  4. Start with  $Old\_seq$  and  $Best\_seq$
  5. Calculate  $C_{max}$  or  $\sum C_i$  of  $Best\_seq$
  6. Calculate  $C_{max}$  or  $\sum C_i$  of  $Old\_seq$
  7. Perform permutation on  $Old\_seq$  and generate  $New\_seq$
  8. Calculate  $C_{max}$  or  $\sum C_i$  of  $New\_seq$
  9. Calculate  $\Delta mk_1$  or  $\Delta \sum C_i 1$
  10. Calculate  $\Delta mk_2$  or  $\Delta \sum C_i 2$
  11. If (9)  $\leq 0$  and (10)  $\leq 0$  then,  
 $Old\_seq = New\_seq; Best\_seq = New\_seq; T_{itr}--;$  Go to (3)
  12. Else if, (9)  $\leq 0$  and (10)  $> 0$  then,  
 $Old\_seq = New\_seq; T_{itr}--;$  Go to (3)
  13. Else if (6)  $> 0$  and (7)  $> 0$  then, generate  $Rprob$
  14. Calculate  $Aprob = 1/e^{(\Delta mk_1/T)}$ , if  $Ch=M$ .
  15. Else,  $Aprob = 1/e^{(\Delta \sum C_i 1/T)}$
  16. If  $Rprob \in Aprob$ , then solution is *escaped* from being discarded.  
 $Old\_seq = New\_seq; T_{itr}--;$  Go to (3)
  17. Else, solution is *discarded*.  
Perform  $T_{itr}--;$  Go to (3)
  18. Stop
-

### 3.3.3 Assumptions

There are some basic assumptions made by the algorithm, which correspond to the initial values of some of the parameters used in the algorithm. They are mentioned as follows

1. *Old\_seq* has the initial value as  $\{1, \dots, Nj\}$  at the first iteration of the algorithm, and later on it varies in the following iterations according to the conditions satisfied.
2. *Best\_seq* has the initial value as  $\{1, \dots, Nj\}$  at the first iteration of the algorithm and later on it varies if it finds a much better schedule in the followed iterations.

### 3.3.4 Operations

**Perform Iteration** operation refers to performing iteration on the job sequence input. This algorithm maintains the job sequence in an array. Two random positions of the array are picked and swapped, to come up with a new schedule i.e., *New\_seq* in a new iteration. Simple swapping strategy is adopted here, in order to produce a new permutation from the given jobs.

**Generate Aprob** operation refers to the process of generating the acceptance probability which calls for the decision, whether or not to accept *New\_seq* generated in the latest iteration. If the criterion which the algorithm strives to optimize is not improved with the *New\_seq*, then *Aprob* is generated. Otherwise, if it gives a sign of improvement in the solution, then in such cases, the *New\_seq* is right away accepted.



If  $Ch = M$  and  $\Delta mk_l \geq 0$ ,

$$Aprob = 1/e^{(\Delta mk_l/T)}$$

If  $Ch=S$  and  $\Delta \sum C_i l \geq 0$ ,

$$Aprob = 1/e^{(\Delta \sum C_i l/T)}$$

**Generate Rprob** operation is performed to generate a random probability value  $Rprob$ . It is generated whenever  $Aprob$  is generated. In such cases it is checked whether or not  $Rprob$  belongs to  $Aprob$ . Only if it is true, the  $New\_seq$  is considered for further iterations in the algorithm. Otherwise it is discarded.

### 3.4 An Example

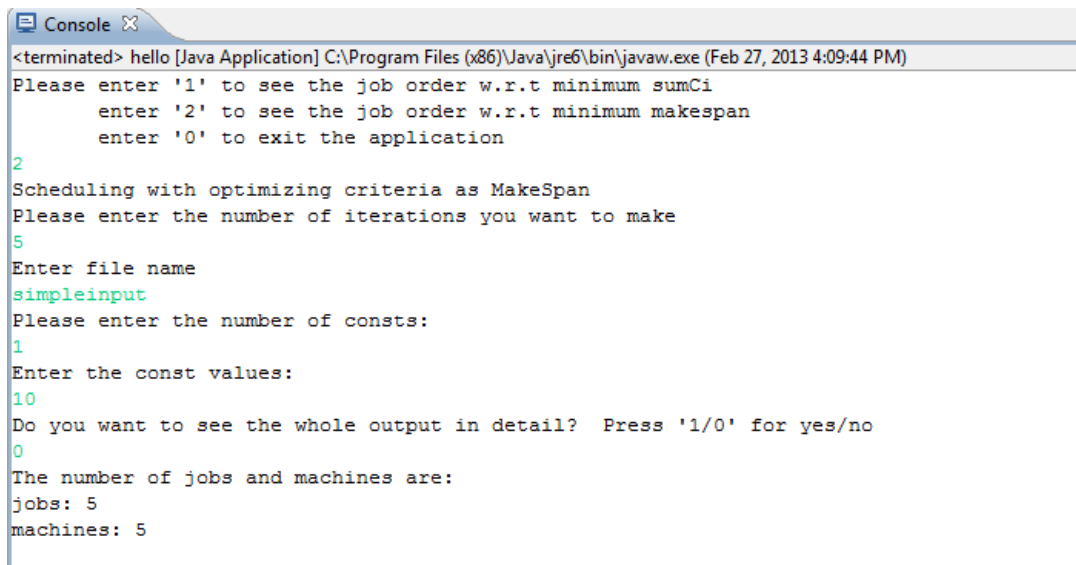
Consider a small example in Table 3.1 which is intended to reflect working behavior of the algorithm.

|    | M1 | M2 | M3 | M4 | M5 |
|----|----|----|----|----|----|
| J1 | 5  | 2  | 3  | 1  | 4  |
| J2 | 1  | 4  | 5  | 6  | 7  |
| J3 | 2  | 7  | 6  | 5  | 1  |
| J4 | 1  | 2  | 3  | 4  | 5  |
| J5 | 5  | 4  | 3  | 2  | 1  |

**Table 3.1 Example to demonstrate the behavior of our Algorithm**

Let us consider the problem of scheduling 3 jobs say  $J_1, J_2, J_3$  to be scheduled on the 3 machines say  $M_1, M_2, M_3$ .

The following screens would give a brief view about the working procedure of the algorithm for the given input. They would display the iterative improvement shown by the algorithm, over the iterations performed.



```
<terminated> hello [Java Application] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (Feb 27, 2013 4:09:44 PM)
Please enter '1' to see the job order w.r.t minimum sumCi
        enter '2' to see the job order w.r.t minimum makespan
        enter '0' to exit the application
2
Scheduling with optimizing criteria as MakeSpan
Please enter the number of iterations you want to make
5
Enter file name
simpleinput
Please enter the number of consts:
1
Enter the const values:
10
Do you want to see the whole output in detail? Press '1/0' for yes/no
0
The number of jobs and machines are:
jobs: 5
machines: 5
```

**Figure 3.1 Input to the Algorithm**

The first step is to choose the optimality criterion, which makes the algorithm to work in the preferred direction i.e., whether to optimize  $C_{max}$  or  $\sum C_i$  accordingly. In this example,  $C_{max}$  is chosen as the parameter to optimize. The number of machines, number of jobs and their processing times are given in the form of an input file.

```

1 : MS: 39 , Best: 39
    1 2 3 4 5

2 : MS: 31 , BEST: 31
CURRENT ITERATION : 4 2 3 1 5
CURRENT SOL :      4 2 3 1 5

3 : MS: 33 , BEST: 31 ,escaped!!  RANDOM NUMBER :490
CURRENT ITERATION : 4 2 5 1 3
CURRENT SOL :      4 2 5 1 3

4 : MS: 43 , BEST: 31 ,Discarded!  RANDOM NUMBER :144
CURRENT ITERATION : 3 2 5 1 4
CURRENT SOL :      4 2 5 1 3

5 : MS: 43 , BEST: 31 ,Discarded!  RANDOM NUMBER :848
CURRENT ITERATION : 3 2 5 1 4
CURRENT SOL :      4 2 5 1 3

FINAL RESULTS:

printing the current best order w.r.t makespan and best makespan value

4 2 3 1 5  and Current Best makespan is: 31

```

**Figure 3.2 : Iterative Improvement**

As the number of iterations to be done is 5, we have 5 iterations to be performed per epoch, where epochs refer to the number of values in the cooling schedule. Here we have only one epoch and hence the total number of iterations is 5.

In the output section, the variables  $MS$  represents the  $C_{max}$  at the current iteration,  $Best$  represents the best  $C_{max}$  value the current iteration,  $Current Iteration$  represents the Job Sequence at the current iteration.

In every iteration,

$\Delta mk_1 =$  Difference between the current makespan and the makespan of the last accepted sequence.

If  $\Delta mk_1 \leq 0$ , it indicates an improvement in the solution.

$\Delta mk_2 =$  Difference between the current makespan and the best makespan value maintained till the current iteration.

If  $\Delta mk_2 \leq 0$ , it indicates an improvement in the new solution.

1. At iteration 1,

Job Sequence:  $J_1-J_2-J_3-J_4-J_5$ ,  $MS=39$ ,  $Best=39$

As this is the initial solution, it is accepted and we perform next permutation for the next iteration on this job sequence.

2. At iteration 2,

Job Sequence:  $J_4-J_2-J_3-J_1-J_5$ ,  $MS = 31$

Condition Satisfied:  $\Delta mk_1 \leq 0$

Condition Satisfied:  $\Delta mk_2 \leq 0$

So we make,  $Best = MS$ . Hence,  $Best = 31$

We accept the solution, and perform the next iteration on this schedule.

3. At iteration 3,

Job Sequence:  $J_4-J_2-J_5-J_1-J_3$ ,  $MS = 33$

Condition Failed:  $\Delta mk_1 \leq 0$

Condition Failed:  $\Delta mk_2 \leq 0$

So, generate  $Aprob$  and  $Rprob$  and check if  $Rprob \in Aprob$ . In this case,

Condition Satisfied:  $Rprob \in Aprob$

So, the schedule generated in this iteration is said to be *ESCAPED* from being discarded, and the next iteration is performed on this schedule.

4. At iteration 4,

Job Sequence:  $J_3-J_2-J_5-J_1-J_4$ ,  $MS = 43$

Condition Failed:  $\Delta mk_1 \leq 0$

Condition Failed:  $\Delta mk_2 \leq 0$

So, generate  $Aprob$  and  $Rprob$  and check if  $Rprob \in Aprob$ . In this case,

Condition Failed:  $Rprob \in Aprob$

So, the schedule generated in this iteration is said to be *DISCARDED*, and the next iteration is performed on the last accepted schedule which is here, the schedule from Iteration 3.

5. At iteration 5,

Job Sequence:  $J_3-J_2-J_5-J_1-J_4$ ,  $MS = 43$

Condition Failed:  $\Delta mk_1 \leq 0$

Condition Failed:  $\Delta mk_2 \leq 0$

So, we generate *Aprob* and *Rprob* and check if  $Rprob \in Aprob$ . In this case,

Condition Failed:  $Rprob \in Aprob$

So, the schedule generated in this iteration is said to be discarded, and the next iteration is performed on the last accepted schedule which is here, the schedule from Iteration 3.

As the algorithm has performed the given number of iterations, it terminates and gives the minimal value it could acquire in 5 iterations. The minimum possible value the algorithm could get, after performing the given number of iterations is displayed in the form of final results. The final results also include the job sequence for which it acquired the minimal value.

So, from this experiment, we can conclude that this algorithm has tried to optimize the  $C_{max}$  value from 39 to 31 following an iterative procedure, for the improvement in the chosen criteria in the given number of iterations. And, the same strategy is followed if the optimality criteria chosen as  $\sum C_i$ .

## CHAPTER 4

### FINDINGS

This chapter gives a detailed view about the results obtained by the Algorithm 3.1 and also gives an insight about the effectiveness of the implementation.

Many experiments have been performed to test the Algorithm 3.1 and it has been concluded that, the more the algorithm iterates, the higher are the chances of obtaining much enhanced solution. We do not know what the optimal solution for the given scheduling problem is, if it is NP-Hard. In order to know how far the algorithm is performing well, there is a need to find that how close is the solution generated by the implemented algorithm, when compared with the optimal solution.

So, we consider a benchmark example for which the ideal schedule is already known, and use this to test the performance of the Algorithm 3.1. So, a standard example is presented in the first section. The second section explains the outcome of the algorithm for varied inputs and the last section compares the results of this algorithm with the standard results acquired by Taillard's instances.

#### 4.1 A Classic Example

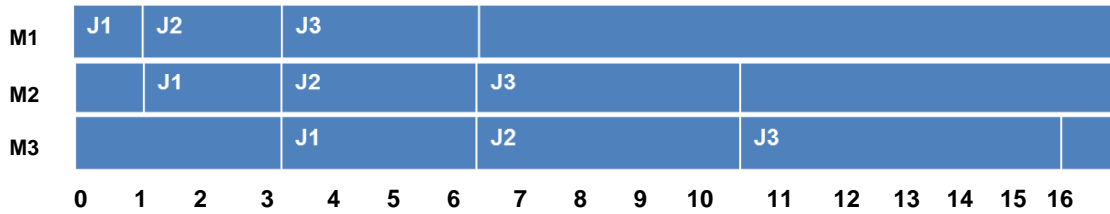
As said before, we need a standard example where we are aware of the optimal schedule, for the given processing times.

|    | M1 | M2 | M3 |
|----|----|----|----|
| J1 | 1  | 2  | 3  |
| J2 | 2  | 3  | 4  |
| J3 | 3  | 4  | 5  |

**Table 4.1 A Classic Example**

So, we considered an example Table 4.1, with 3 jobs and 3 machines, and for which we know the optimal schedule. This schedule is the optimal solution if the optimality criterion is  $C_{max}$  or  $\sum C_i$ .

So, the optimal schedule for this problem is  $J_1-J_2-J_3$ . This schedule has zero idle time. We can represent it in the form of a Gantt chart as shown in Figure 4.1.



**Figure 4.1 Optimal Schedule for the Classic Example**

## 4.2 Conducted Experiments

In order to test the optimality of the solution produced by our algorithm, we need to consider a bigger, and another standard example of the same pattern for the processing times as followed in Table 4.1.

|            | M1 | M2 | M3 | M4 | M5 | M6 | M7 | M8 | M9 | M10 |
|------------|----|----|----|----|----|----|----|----|----|-----|
| <b>J1</b>  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15  |
| <b>J2</b>  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11  |
| <b>J3</b>  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12  |
| <b>J4</b>  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10  |
| <b>J5</b>  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13  |
| <b>J6</b>  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14  |
| <b>J7</b>  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16  |
| <b>J8</b>  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17  |
| <b>J9</b>  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18  |
| <b>J10</b> | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19  |

**Table 4.2 Processing times for Test 1 and Test 2**

This example is presented with 10 jobs and 10 machines under Table 4.2. , and it is used as input for both the tests in Section 4.3.1 and Section 4.3.2. In the Output section of these tests, the outcome of the algorithm for its every 1000<sup>th</sup> iteration is tabulated. In Table 4.3 AS and BEST represent average and best  $\sum C_i$  respectively. In Table 4.4 MS represents  $C_{max}$  value and BEST represents the best  $C_{max}$  value.

#### **4.2.1 Test 1**

##### **Input**

Optimality Criterion =  $\sum C_i$ .

Number of constants in the cooling schedule = 5

Cooling constants = {500, 450, 300, 150, 50}

Number of iterations = 1000



## Output

The number of jobs and machines are:

Jobs: 10

Machines: 10

| Iteration# | Cooling Constant | AS  | BEST |
|------------|------------------|-----|------|
| 1000       | 500              | 206 | 130  |
| 2000       | 450              | 200 | 130  |
| 3000       | 300              | 172 | 130  |
| 4000       | 150              | 133 | 125  |
| 5000       | 50               | 160 | 125  |

**Table 4.3 Results for Test 1**

Final Results:

Printing the Best order w.r.t sumCi and Best sumCi value

$J_4 - J_2 - J_3 - J_5 - J_6 - J_1 - J_7 - J_8 - J_{10} - J_9$

Best Average SumCi is: 118

### Efficiency:

Efficiency of the algorithm for the given input is calculated based on the optimal solution of the problem.

Optimal Schedule:  $J_4 - J_2 - J_3 - J_5 - J_6 - J_1 - J_7 - J_8 - J_9 - J_{10}$

The optimal value of  $\sum C_i$  is 116

The final output of the algorithm is 118

## Graph for Iterative Improvement

A graph has been plotted for easy understanding of the improvement shown by the algorithm stage by stage.

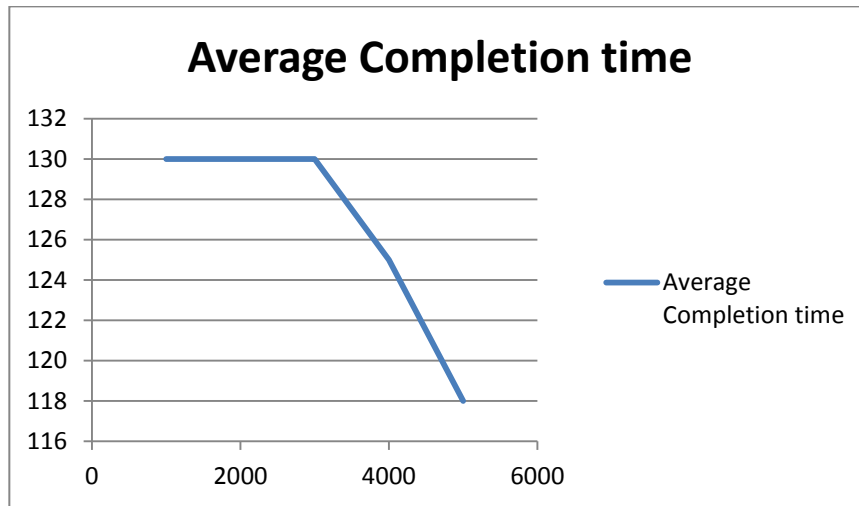


Figure 4.2 Iterative Improvements by the Algorithm for Test 1

### 4.2.2 Test 2

#### Input

Optimality Criterion =  $C_{max}$

Number of constants in the cooling schedule = 5

Cooling constants = {500, 450, 300, 150, 50}

Number of iterations = 1000

#### Output

The number of jobs and machines are:

Jobs: 10

Machines: 10

| Iteration# | Cooling Constant | MS  | BEST |
|------------|------------------|-----|------|
| 1000       | 500              | 235 | 235  |
| 2000       | 450              | 271 | 208  |
| 3000       | 300              | 244 | 208  |
| 4000       | 150              | 262 | 208  |
| 5000       | 50               | 271 | 199  |

**Table 4.4 Results for Test 2**

Final Results:

Printing the Best order w.r.t makespan and Best makespan value

$J_2 - J_3 - J_4 - J_5 - J_1 - J_7 - J_8 - J_9 - J_{10} - J_6$

Best makespan is: 199

**Efficiency:**

Efficiency of the algorithm for the given input is calculated based on the optimal solution of the problem.

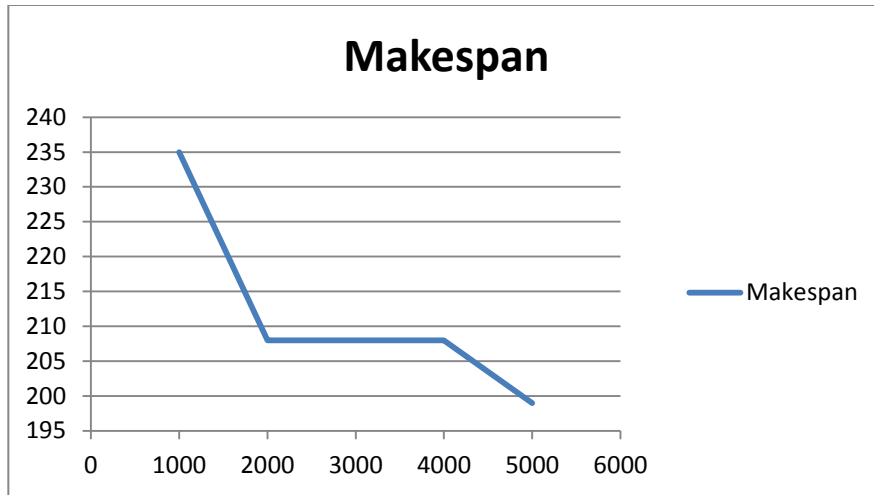
Optimal Job Order:  $J_7 - J_9 - J_1 - J_3 - J_6 - J_5 - J_{10} - J_4 - J_2 - J_8$

The optimal value of  $C_{\max}$  is 190.

The final output of the algorithm is 199.

**Graph for Iterative Improvement**

A graph has been plotted for easy understanding of improvement shown by the algorithm.



**Figure 4.3 Iterative Improvements by the Algorithm for Test 2**

### 4.3 Comparison with Taillard's Instances

Taillard is a great researcher who has performed various experiments on shop problems and he has presented a set of standard results produced for a set of standard input instances called Taillard Instances [20].

A comparison has been made between the results produced by Taillard's experiment and the results of Algorithm 3.1. Taillard's experiments focus on minimizing the  $C_{max}$  value of a schedule.

#### 4.3.1 Results produced by the Algorithm

In Table 4.5, 4.6, 4.7, Column 1 represents the Instance number, Column 2 and Column 3 represent the upper bound and lower bound of Taillard's experiment respectively and the Results of our algorithm in Column 4.

## 5 jobs and 20 machines

| Instance | Taillard's |      | Results |
|----------|------------|------|---------|
|          | UB         | LB   |         |
| 1        | 1278       | 1232 | 1265    |
| 2        | 1359       | 1290 | 1237    |
| 3        | 1081       | 1073 | 1133    |
| 4        | 1293       | 1268 | 1449    |
| 5        | 1236       | 1198 | 1342    |
| 6        | 1195       | 1180 | 1254    |
| 7        | 1239       | 1226 | 1221    |
| 8        | 1206       | 1170 | 1331    |
| 9        | 1230       | 1206 | 1339    |
| 10       | 1108       | 1082 | 1231    |

**Table 4.5 Instances of 5 jobs and 20 machines**

Table 4.5 includes the results of our algorithm for the Taillard's instances of 5 jobs and 20 machines. Each instance represented in Column 1 is a data set of 5 jobs and 20 machines.

## 10 jobs and 20 machines

| <b>Instance</b> | <b>Taillard's<br/>UB</b> | <b>Taillard's<br/>LB</b> | <b>Results</b> |
|-----------------|--------------------------|--------------------------|----------------|
| 1               | 1582                     | 1448                     | 1595           |
| 2               | 1659                     | 1479                     | 1731           |
| 3               | 1496                     | 1407                     | 1574           |
| 4               | 1378                     | 1308                     | 1409           |
| 5               | 1419                     | 1325                     | 1445           |
| 6               | 1397                     | 1290                     | 1395           |
| 7               | 1484                     | 1388                     | 1515           |
| 8               | 1538                     | 1363                     | 1551           |
| 9               | 1593                     | 1472                     | 1570           |
| 10              | 1591                     | 1356                     | 1707           |

**Table 4.6 Instances of 10 jobs and 20 machines**

Table 4.6 includes the results of our algorithm for all the instances used by Taillard's for 10 jobs and 20 machines. Each instance represented in Column 1 is a data set of 10 jobs and 20 machines.

## 20 jobs and 20 machines

| <b>Instance</b> | <b>Taillard's<br/>UB</b> | <b>Taillard's<br/>LB</b> | <b>Results</b> |
|-----------------|--------------------------|--------------------------|----------------|
| 1               | 2297                     | 1911                     | 2269           |
| 2               | 2100                     | 1711                     | 2164           |
| 3               | 2326                     | 1844                     | 2202           |
| 4               | 2223                     | 1810                     | 2210           |
| 5               | 2291                     | 1899                     | 2283           |
| 6               | 2226                     | 1875                     | 2215           |
| 7               | 2273                     | 1875                     | 2302           |
| 8               | 2200                     | 1880                     | 2183           |
| 9               | 2237                     | 1840                     | 2291           |
| 10              | 2178                     | 1900                     | 2156           |

**Table 4.7 Instances of 20 jobs and 20 machines**

Similarly, Table 4.7 includes the results of our algorithm for all the instances used by Taillard for 20 jobs and 20 machines. Each instance represented in Column 1 is a data set of 20 jobs and 20 machines.

### 4.3.2 Conclusion

After testing the algorithm and comparing with the Taillard's instances, it was observed that, the algorithm we have implemented has performed well for some instances and needed improvement for some instances.

## CHAPTER 5

### CONCLUSION AND FUTURE WORK

In this thesis, we have implemented an algorithm, mentioned in Chapter 3 which was an attempt to provide improvement in the solutions to a scheduling problems which is NP-Hard. In Chapter 4, we have discussed about all the results produced by Algorithm 3.1 and tested its efficiency using a standard example. We have also compared the results of our algorithm with the standard results produced by the experiments conducted by Taillard.

After all these tests, it can be concluded that our algorithm has performed well in many cases and its performance depends on the inputs given to the algorithm. The input here we refer to is the cooling schedule input to the algorithm. We discussed the importance of this input parameter in Chapter 2 in Section 2.5.2. It has a crucial role in directing the algorithm towards the optimal solution. Also, when compared our results with Taillard's results, it was noticed that our algorithm has exhibited a good behavior towards most of the instances, and needed improvement for some instances.

Further improvements can be made in our algorithm so as to bring out much more enhanced solutions, which can be possible by altering and improving the cooling schedule and neighborhood of the algorithm.



## BIBLIOGRAPHY

- [1] Cormen, Thomas H., et al. *Introduction to algorithms*, Third Edition, Chapter 34. MIT press, 2001.
- [2] Brucker, Peter. *Scheduling algorithms*, Chapter 1, 6. Springer, 2007.
- [3] Pinedo, Michael L. *Scheduling: theory, algorithms, and systems*, Fourth Edition, Chapter 2. Springer, 2012.
- [4] Vaik, Zsuzsanna. *On scheduling problems with parallel multi-purpose machines*. Technical Reports, Egervary Research Group on Combinatorial Optimization, Hungary, [www.cs.elte.hu/egres/tr/egres-05-02.pdf](http://www.cs.elte.hu/egres/tr/egres-05-02.pdf), 2005.
- [5] Zalzal, Ali MS, and Peter J. Fleming, eds. *Genetic algorithms in engineering systems*. Vol. 55. Iet, 1997.
- [6] Panneerselvam, R. *Production and operations management*, Second Edition, Chapter 14, PHI Learning Pvt. Ltd., 2006.
- [7] Leung, Joseph YT, ed. *Handbook of scheduling: algorithms, models, and performance analysis*, Vol. 1, Chapter 6. Chapman & Hall/CRC, 2004.
- [8] Schoenmakers, L. A. M. *A new algorithm for the recognition of series parallel graphs*. Department of Algorithmics and Architecture, CWI, 1995.
- [9] Zmaranda, Doina, and Gianina Gabor. "Issues on Optimality Criteria Applied in Real-Time Scheduling." *Int. J. of Computers, Communications and Control* 3 (2008): 536-540.
- [10] Gonzalez, Teofilo, and Sartaj Sahni. "Flowshop and jobshop schedules: complexity and approximation." *Operations Research* 26.1 (1978): 36-52.
- [11] Gao, Jian, and Rong Chen. "An NEH-based heuristic algorithm for distributed permutation flowshop scheduling problems." *Scientific Research and Essays* 6.14 (2011): 3094-3100.
- [12] Samia kouki, Samia kouki, Mohamed Jemni Mohamed Jemni, and Talel Ladhari Talel Ladhari. "Solving the Permutation Flow Shop Problem with Makespan Criterion using Grids." *International Journal of Grid and Distributed Computing* 4.2 (2011): 53-64.

- [13] Sotskov, Yu N., and N. V. Shakhlevich. "NP-hardness of shop-scheduling problems with three jobs." *Discrete Applied Mathematics* 59.3 (1995): 237-266.
- [14] Semančo, Pavol, and Vladimír Modrák. "Hybrid GA-based improvement heuristic with makespan criterion for flow-shop scheduling problems." *ENTERprise Information Systems* (2011): 11-18.
- [15] Adusumilli, Kumar, Doina Bein, and Wolfgang Bein. "A Genetic Algorithm for the Two Machine Flow Shop Problem." *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*. IEEE, 2008.
- [16] Rutenbar, Rob A. "Simulated annealing algorithms: An overview." *Circuits and Devices Magazine, IEEE* 5.1 (1989): 19-26.
- [17] Henderson, Darrall, Sheldon Jacobson, and Alan Johnson. "The theory and practice of simulated annealing." *Handbook of metaheuristics* (2003): 287-319.
- [18] Anily, S., and A. Federgruen. "Simulated annealing methods with general acceptance probabilities." *Journal of Applied Probability* (1987): 657-667.
- [19] Hajek, Bruce. "Cooling schedules for optimal annealing." *Mathematics of operations research* 13.2 (1988): 311-329.
- [20] Taillard's Instances  
<http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/flowshop.dir/>
- [21] Alcan, Pelin, and Huseyin Basligil. "An Application with Non-identical Parallel Machines using Genetic Algorithm with the Help of Fuzzy Logic." *Lecture Notes in Engineering and Computer Science* 2191 (2011).
- [22] Oral, Muhittin, and J-L. Malouin. "Evaluation of the shortest processing time scheduling rule with truncation process." *AIIE Transactions* 5.4 (1973): 357-365.
- [23] Rajkumar, R., et al. "A Bi-Criteria Approach to the M-machine Flowshop Scheduling Problem." *Studies in Informatics and Control* 18.2 (2009): 127-136.

**VITA**  
Graduate College  
University of Nevada, Las Vegas

Sadhana Yellanki

Degrees:

Bachelor of Technology in Information Technology, 2011  
Jawaharlal Nehru Technological University  
Master of Science in Computer Science, 2013  
University of Nevada Las Vegas

Thesis Title: Simulated Annealing Approach to Flow Shop Scheduling

Thesis Examination Committee:

Chair Person, Dr. Wolfgang Bein, Ph.D.  
Committee Member, Dr. Ajoy K. Datta, Ph.D.  
Committee Member, Dr. Ju-Yeon Jo, Ph.D.  
Graduate College Representative, Dr. Venkatesan Muthukumar, Ph.D.