5-1-2019

Permutation Flow Shop via Simulated Annealing and NEH

Pooja Bhatt
bhatt.pooza@gmail.com

PERMUTATION FLOW SHOP VIA SIMULATED ANNEALING AND NEH

By

Pooja Bhatt

Bachelor of Science (B.Sc.) in Computer Science and Information Technology(CSIT)
Tribhuwan University, Kathmandu, Nepal
2012

A thesis submitted in partial fulfillment of
the requirements for the

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 2019

UNLV | GRADUATE COLLEGE

Thesis Approval

The Graduate College
The University of Nevada, Las Vegas

April 11, 2019

This thesis prepared by

Pooja Bhatt

entitled

Permutation Flow Shop via Simulated Annealing and NEH

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.
Examination Committee Chair

Kathryn Hausbeck Korgan, Ph.D.
Graduate College Dean

Ajoy Datta, Ph.D.
Examination Committee Member

Laxmi Gewali, Ph.D.
Examination Committee Member

Henry Selvaraj, Ph.D.
Graduate College Faculty Representative

Abstract

Permutation Flow Shop Scheduling refers to the process of allocating operations of jobs to machines such that an operations starts to process on machine j only after the processing completes in j-1 machine. At a time a machine can process only one operation and similarly a job can have only one operation processed at a time. Finding a schedule that minimizes the overall completion times for Permutation Flow Shop problems is NP Hard if number of machines is greater than 2. So we concentrate on approaches with approximate solutions that are good enough for the problems. Heuristics is one way to find the approximate solutions for a problem.

For our thesis, we have used two heuristics - NEH and Simulated Annealing, both individually and in a combined form, to find the solutions for Permutation Flow Shop problems. We have compared NEH and Simulated Annealing algorithm based on result and execution time and also compared the combined algorithm with existing ones. Standard benchmarks are used to evaluate the performances of the implemented algorithm.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisor and committee chair, Dr.Wolfgang Bein for his guidance, suggestions and feedbacks throughout my thesis.
I am very grateful to my committee members Dr. Ajoy K. Datta, Dr. Laxmi Gewali and Dr. Henry Selvaraj for their willingness and time to serve on my committee.
I am indebted to my family without whom I wont be here. Their immense love and believe in me gives me strength and encouragement to overcome all the problems in pursuit of my dreams.
Finally, I would like to thank all my friends for their love, support, and guidance.

Pooja Bhatt

University of Nevada, Las Vegas
May 2019

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Scheduling is the process of managing tasks to utilize resources for an effective outcome. Making up a schedule is an important part in every ones daily life. Flow Shops are scheduling problem with a set of jobs and machines. A job has set of operations that needs to be processed in given machines. At a time, processing of an operations of a job can be done on only one machine. Similarly a machine is able to process only one job operations at one time. The scheduling problem is to specify order of the processing for each operation O_{ik} of Job J_i such that an operation O_{ik-1} in k-1 machine needs to be completed before starting processing on machine k. This problem seems simple but is NP- Hard if number of resources are greater than 2. The number of order that can be obtained from Flow Shop Scheduling problem is $(n!)^m$. Flow Shop problem can be represented by $n/m/F/C_{max}$ [RWCM67] or $F//C_{max}$ [RK79].

Flow Shop problems have history of more than 50 years of research. We have included some of the history of Flow shop referenced from Hejazi et.al. [HS05]. In classic Flow Shop described by Allahverdi et.al [TAA99], a job may wait on or between machines with infinite buffer. One variant of this problem is where job cannot form queues, e.g. Zero-buffer and no-wait Flow Shop problems. This means jobs J_i in machine k-1 cannot advance to machine k until machine k has completed the processing of jobs before in sequence. Abadi and Sriskandarajah [IAS96] described blocking Flow Shop problem where a machine can leave a machine only after next machine is free. Aldowaisan and Allahverdi [AA88] described no wait Flow Shop problem in which each job needs to be processed in line without delay from machine 1 to m. They also proposed Simulated Annealing and Genetic Algorithm for same problem [AA03].

Gangadharan and Rajendran [GR93] and Rock [Roc84] discussed problems based on three

machine no-wait flow shop NP- completeness. Hall and Sriskandarajah described the same in the survey [HS96] . Some earlier researchers - Stafford and Tseng [ST90], and Wismer [D.A72], called this No-Intermediate Queues(NIQ) Flow Shop problem.

Lee et al. [CLL93] and Potts et al. [CPZ95] initiated the concurrency concept in a flow shop environment. Koulamas and Kyparisis [KK04] extended the concept of concurrency with makespan criterion by introducing new shop. Guinet [Gui00] studied job shop problems. Nagar et al. [ANH96] proposed branch and bound by combining with genetic algorithm to solve two machine Flow Shop problem. Similarly Neppalli et al. [VNG96] applied genetic algorithms to solve the same problem. To solve n=m=F=C_max, Rajendran and Gangadharan [GR94] used Simulated Annealing.

A Special case of the Flow Shop problem is Permutation Flow Shop problem. In Permutation Flow Shop, the job order is the same for all machines, that reduces the number of sequences to n!. Due to many real world application and decreased number of sequences, a lot of research is now focused on Permutation Flow Shop problems. With two machines, Johnson's algorithm [Joh54] solves the problem. However, when the machine size is greater than 2; the problem is NP-Hard [MGR76]. Exact algorithms, heuristics, and meta heuristics, are mostly used to find solutions for Permutation Flow Shop problems ([RR05], [JXG14]). Exact algorithms find the optimal solution, but are computationally expensive as the problem size gets bigger. Branch and bound [CSC02], Dynamic Programming are examples of exact algorithms. For some, the solution doesn't even exists. On the other hand, Heuristics and meta heuristics are cost effective and feasible. However, the solution obtained from Heuristics is not always optimal. Heuristics provide an approximate solution which are still a good solution for the problems.

The heuristics can be divided as Constructive heuristics and Improvement heuristics. A non reversible sequence is made for Constructive heuristics. An improvement heuristics or descent method is an iterative method that starts with any sequences and attempts to improve the value of objective functions by modifying the sequence. Only improved sequences that decrease the cost are accepted and even constructive sequences can be used for improvement. Several Constructive heuristics([HCS70], [Dan77], [MN83], [Pal65]) are proposed and the number of heuristics are increasing every year. Dannenbring [Dan77] proposes an improvement method in which adjacent job interchanges are attempted. Similarly, Simulated Annealing is also a randomized improvement method but it also accepts non improved sequences with some probability [IOR89].

For our thesis, we are using two heuristics- NEH and Simulated Annealing. Also we have com-

bined both algorithms for the improvements of the solution. Comparisons are done between both algorithms and with the combined ones using standard benchmarks.

1.1 Outline

Chapter 2 discusses in detail about Scheduling: Terms used in Scheduling, Classes of Scheduling, Flow Shop Problems and Permutation Flow Shop problems. Chapter 3 discuss about Problem Statement, Simulated Annealing Algorithm and its Implementation. Chapter 4 is about NEH algorithm, its variations and combination of NEH and Simulated Annealing algorithm. Chapter 5 lists out the results with algorithm implemented in Chapter 3 and Chapter 4. It also also contains results of the comparisons between the two algorithms. Chapter 6 gives the conclusion and probable future improvements.

Chapter 2

Background and Shop Problems

This chapter includes some of the fundamental concepts of Scheduling.

2.1 Scheduling

Scheduling is the branch of science that deals with the allocating tasks to the available resources such that the resources are utilized in optimal manner. Suppose we have m resources $M_1, M_2, M_3, ..., M_m$ called machines and n jobs $J_1, J_2, ..., J_n$. Allocation of the operations of n jobs on the m machines such that at a time a job is processed by only one machine and a machine process only one operation of a job is called Scheduling [Bru04].

2.1.1 Terms in Scheduling

Job

A job is group of tasks or operations that needs to be processed by machines. Each operation has processing time corresponding to each machine. Throughout the thesis,a job is represented by J_i and operations by O_{ij} where i denotes the job and j denotes machine.

Processing Time

The time taken by an operation of a job to complete processing in a machine is called Processing time. For an operation O_{ij}, processing time is represented as p_{ij} and is always positive number.

Idle Time

The time where a machine does nothing is Idle time for the machine i.e the time where a machine doesn't have any job for processing.

Makespan

Makespan for a schedule is the time duration between the start of first job and completion of last job. It is the time where all machines have completed processing of all the given jobs and is denoted as C_{max}.

The optimal makespan is the one that is minimum among all the possible schedules for the given problem.

2.2 Classes of Scheduling

Scheduling problem can be classified in different categories based on the attributes associated with jobs and machines. According to Peter Brucker [Bru04], scheduling problems can be defined with three fields namely $\alpha|\beta|\gamma$. The first symbol α defines the machine environment, β defines the job characteristics and γ represents the optimality. All the terms and definition included in this chapter are adapted from Peter Brucker's book named "Scheduling Algorithms".

2.2.1 Machine Environment

Based on the serving purpose, different types of the machines are used for Scheduling environment. It is denoted by the string α which contains $\alpha_1\alpha_2$, each with own meaning. α_1 can have any of {o, P, Q, R, PMPM, QMPM, G, X, O, J, F} values and α_2 represent number of machines. If machine number is fixed for any problem, then it is represented as $\alpha_2 = y$, where y is a random positive number. α_2=o means that the number of machines present is arbitrary. o denotes void(empty) symbol. If α_1=o, then $\alpha = \alpha_2$.
Each symbols in α_1 implies different meaning when used alone or with combinations. We have different cases for the symbol and each case explains how the operations are performed in a schedule.

Case 1 $\alpha_1 = o$, then each job has only one operation and needs to be processed in dedicated machine. Hence this is called dedicated Machine Environment.

Case 2 $\alpha_1 \in \{P, Q, R\}$, then jobs run on parallel machines. Any job in our given schedule can process on any of the available machine $M_1, M_2, ..., M_m$. The job are said to run on parallel machines but a machine can process only one job at a time and a job can be processed on more than one machine. This case can be subdivided into three new parts based on the symbol taken into considerations.

- $\alpha_1 = P$

In this case the job processing is done on Identical Parallel machine and processing time of a job is same on every machine $M_1, M_2, ..., M_m$. Here, $p_{ij} = p_i$ for all machines M_j.

- $\alpha_1 = Q$

  In this case, the processing is on Uniform Parallel Machines with speed associated with each machine. For all jobs in machine j, the speed of the machine s_j is uniform . Suppose if p_{ij} is the processing time of job J_i in machine j, then $p_{ij} = p_i/s_j$.

- $\alpha_1 = R$

  In this case, the processing is on unrelated machine with speed associated with each jobs instead of machine. The processing p_{ij} time for a job J_i in machine M_j with speed s_{ij} is given by, $p_{ij} = p_i/s_{ij}$.

Case 3 $\alpha_1 = PMPM$ or α_1=QMPM represents multipurpose machines. PMPM in the scheduling means each job J_i have same speed across all the machines but different jobs can have different speed. QMPM refers to the machines where all jobs in a machine have same speed. For PMPM, the speed of one job may vary from another where in QMPM, a machine will process all jobs with same speed and speed of a machine may differ from one another.

Case 4 $\alpha_1 in \{G, X, O, J, F\}$, then it is multi operational model. Each job J_i has set of operations O_{ij} that needs to be processed on dedicated set $\mu_{ij} \subseteq \{M_1, M_2, ..., M_m\}$. These are called shop problems.

- $\alpha_1 = G$ represents General Flow Shop. The processing time p_{ij} of each job is given and each job consists a set of operations O_{ij}. Precedence needs to be followed by operations in every job. A job can be processed by only one machine at a time. Similarly a machine can only process one job at a time. After altering certain conditions, the problem can be altered to some other shop problem like below.

- $\alpha_1 = J$ represents Job Shop. For Job shop all the operations O_{ik} of job J_i need to follow predefined precedence for processing in any machine M_j. The precedence is of the form $O_{ik-1} \rightarrow O_{ik}$ i.e an operation O_{ik-1} must be completed before O_{ik}. Each job follows its own route of visiting the machine

- $\alpha_1 = F$ represents Flow Shop. This is similar to Job Shop but with number of operations in a job equals to the number of machines. Each operation is processed on one machine. While visiting the machine, all jobs have same machine sequence but the job sequence in each machine can be different.

For a schedule, if the job sequence for all machine is same along with machine sequence, then it is Permutation Flow Shop. We use the notation F-perm for Permutation Flow Shop.

- $\alpha_1 = O$, represents Open Shop. This is similar to Flow Shop in that the number of operations in each job J_i is equal to number of Machines m. But the difference from Flow Shop is, there is no precedence relationship between operations. Thus job order as well as machine order is needed for Open Shop.

- $\alpha_1 = X$, represents Mixed Shop. Mixed Shop sometimes behaves as Open Shop and some times as Job Shop. As in Job Shop, some of the jobs follow specified machine order while some jobs doesn't need to follow any order as in Open Shop.

2.2.2 Job Characteristics

This refers to the characteristics associated with job and is represented by β. $\beta \in \{\beta_1, \beta_2.\beta_3, \beta_4, \beta_5, \beta_6\}$, each with their own property. Following are six cases, one with each symbol.
Case 1 $\beta = \beta_1$, represents preemption is allowed for a schedule. Preemption means any job can be removed or paused for a while and later started on same or different machine. $\beta_1 = pmtn$ when preemption is allowed for a schedule.

Case 2 $\beta = \beta_2$ represents precedence relationship exists between jobs in a schedule. The jobs and operations need to follow the precedence defined if β_2 exists in the set β. A job J_{i-1} must be completed before J_i and needs to be followed throughout the schedule.

Case 3 $\beta = \beta_3$ represents release date associated with each job. Release date is the time when the processing starts for a schedule. If release date is included for a schedule then $\beta_3 = r_i$.

Case 4 $\beta = \beta_4$ is associated with restriction on processing time p_{ij}. When processing time for all jobs p_i=1 then it is unit processing requirement. If operation of all job have value k then $p_i = k$.

Case 5 $\beta = \beta_5$ is associated with deadline for the job and is represented as $\beta_5 = d_i$. The deadline for a job is the time by which the job needs to be completed in that machine.

Case 6 $\beta = \beta_6$ is associated with processing jobs in batches. Batching means grouping of jobs that needs to be proceeded together without any setup before all jobs processing is completed.

2.2.3 Optimality Criterion

γ represents Optimality Criterion. Optimality criterion is associated with cost function. The parameter may vary from one to another. Some of the optimality criterion that can be considered are:

Case 1: One optimality criterion is to minimize the total completion time of the schedule. This is called Bottleneck Objective Function.

Case 2: Another optimality criterion is to minimize overall sum of completion time of all jobs for a schedule. This is called Sum Objective Function.

Apart from above two, other parameter can be used for optimality criterion. Below are some of them:

Lateness

Lateness is the time duration by which a job is late in completing the processing of the schedule. C_i is the time of completion of Job J_i with deadline d_i, then Lateness L_i is obtained as a difference of Completion time and deadline. This can be written as:

$$L_i = C_i - d_i$$

Earliness

Earliness can be defined as the time duration by which a schedule is completed before estimated completion time. This is just opposite of Lateness. Lateness finds how late the job is where earliness calculates how early the job completes. For C_i completion time and d_i deadline, Earliness E_i is the difference of deadline and Completion and is 0 if the difference is negative. Mathematically this can be represented as:

$$E_i = d_i - C_i$$

Tardiness

Tardiness is Lateness if L_i is non-negative i.e the difference of Completion time and duration should be positive value for Tardiness. For job J_i, Tardiness T_i is calculated as:

$$T_i = C_i - d_i, if C_i - d_i > 0$$

$$T_i = 0, if C_i - d_i \leq 0$$

Deviation

Deviation are of two types;

- **Squared Deviation** is the square of the lateness values and is represented by S_i.

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

- **Absolute Deviation** is absolute value of lateness and is represented by D_i.

$$D_i = |L_i| = |C_i - d_i|$$

Unit Penalty

This is associated with lateness. If lateness is positive value, then penalty is 1. Otherwise it's 0. Unit penalty is denoted by U_i.

2.3 Flow Shop Problems

We have already discussed Flow Shop while explaining Machine Environment properties. This section includes Flow Shop in detail with an example and then Permutation Flow Shop problem with example in next section.

We have n jobs $J_1, J_2, ..., J_n$ and m machines $M_1, M_2, .., M_m$. Each job have m operation $O_{i1}, O_{i2}, ..., O_{ik}$. Operation O_{ik} for a job J_i needs to be processed on k^{th} machine. Flow Shop problem can be defined as scheduling problem where precedence is maintained between each operation i.e O_{i1} needs to be completed in machine 1 before starting operation O_{i2} on second machine for job J_i. This sequence is followed by all jobs in the schedule. For each job the first operation will start processing from machine 1 and m^{th} operation will be on m^{th} machine. The job sequence can be different for each machine.

The goal is to minimize the makespan of a schedule. The total sequences for Flow Shop with n job and m machine is $(n!)^m$. Finding the optimal schedule that minimizes makespan we need to check all $(n!)^m$, which is hard as the job size and machine size got bigger and bigger. The problem is NP- Hard if machine size is greater than 2. Therefore we concentrate on Permutation Flow Shop Problems.

2.3.1 Example

Consider example with two machine and 3 jobs in 2.1 with the processing time given for all jobs. In Flow Shop, the machine sequence should be same but the job sequence can be different in each machine. For this example, the machine sequence is M_1M_2 and job sequence is $J_1J_2J_3$ on M_1 and $J_2J_3J_1$ on M_2. Gantt chart is used for visual representation of jobs over different machines. Gantt charts are of two types - Machine Oriented Gantt Chart and Job Oriented Gantt chart. The

	J1	J2	J3
M1	3	1	4
M2	2	3	5

Table 2.1: Example with 3 jobs 2 machine

horizontal axis represent the time frames in both charts. The vertical axis represent machine in Machine Oriented Gantt chart while the axis represent Job for Job Oriented Gantt Chart. We will use Machine oriented Gantt chart for this example and further examples in this Thesis.
The Gantt chart for problem in table 2.1 with schedule $J_1J_2J_3$ on M_1 and $J_2J_3J_1$ on M_2 is represented in figure 2.1

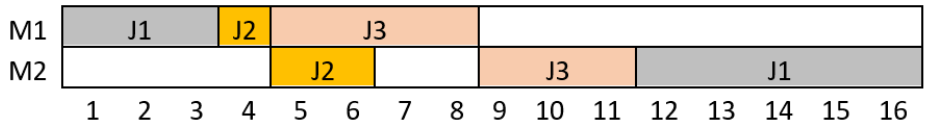

Figure 2.1: Example of Flow Shop scheduling

Flow Shop Scheduling process includes below steps.

1. The schedule for Machine M_1 is $J_1J_2J_3$. At time t=0, J_1 starts on machine M_1. No jobs can start at t=0 on machine M_2 since the execution for J_2 is not completed in M_1. J_1 runs for three units on M_1.

2. J_2 starts at t=3 and runs for 1 unit on M_1. At t=4, J_2 starts on machine M_2 and runs for 2 units.

3. J_3 starts at t=4 on M_1 and run till t=8. From t=6 to t=8, M_2 remain idle. At t=9, J_3 starts processing on M_2 for 3 units.

4. All jobs have completed processing on M_1 so, M_1 remain idle from t=8. Also M_2 remains idle from t=6 to t=8 until J_3 completes processing on M_1.J_3 completes at t=11 and then J_1 starts processing on M_2. J_1 runs for 5 units.

2.4 Permutation Flow Shop problem(PFSP)

Permutation Flow Shop is Flow Shop problem with restriction on order of jobs across machines. The job sequence can be different across machines for Flow Shop but for Permutation Flow Shop problem, the sequence of jobs need to be same across all machine. The sequence or permutation is

the schedule for the problem.

Different schedules are obtained by exchanging the jobs order. Each permutation is denoted by π. With n jobs, the number of permutation is n! as oppose to Flow Shop where the number of sequences is $(n!)^m$. For $m > 2$, the problem is NP Hard but if the processing time of all jobs is same then the problem is not NP Hard. All the permutation will have same sequence no matter which permutation is selected.

The objective of the algorithms implemented in this thesis, is to find approximate result that is near to optimal solutions for Permutation Flow Shop problem. The optimization is obtained by minimizing the makespan i.e. C_{max}. We have used two heuristics to obtain a solution for the problems.

2.4.1 Example

Below is the example for demonstration of PFSP with five jobs and four machines. The processing time of each job is given.

For five jobs, we will have 5! =120 different sequences.

	J0	J1	J2	J3	J4
M0	10	8	4	12	5
M1	2	8	7	10	4
M2	6	12	4	2	8
M3	4	5	7	10	11

Table 2.2: Example for Permutation Flow Shop

Calculating Makespan(C_{max})

The procedure for calculating C_{max} is same for all scheduling problem. As defined earlier, makespan of a schedule is the finishing time of last operation of last job on last Machine. The sequence that make the minimum makespan of all possible sequence is the optimal sequence and the C_{max} for the sequence is optimal solution. The sequence that makes the maximum of all possible sequence for the given problem is worst sequence and makespan value is the worst solution. Makespan for some of sequences obtained from problem in Table 2.2:

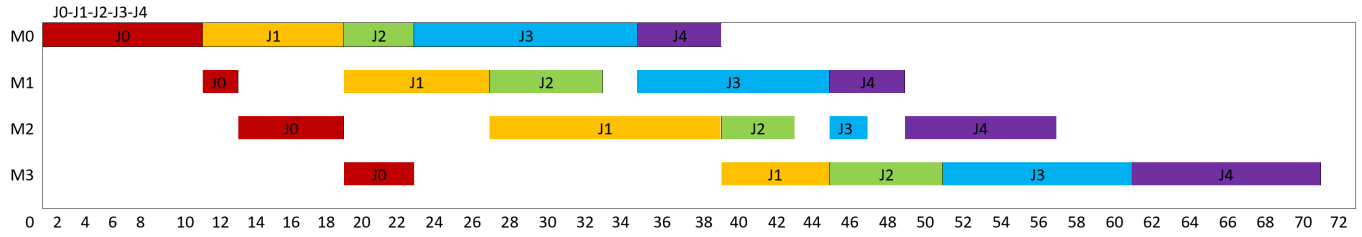

Figure 2.2: PFSP with schedule J0-J1-J2-J3-J4

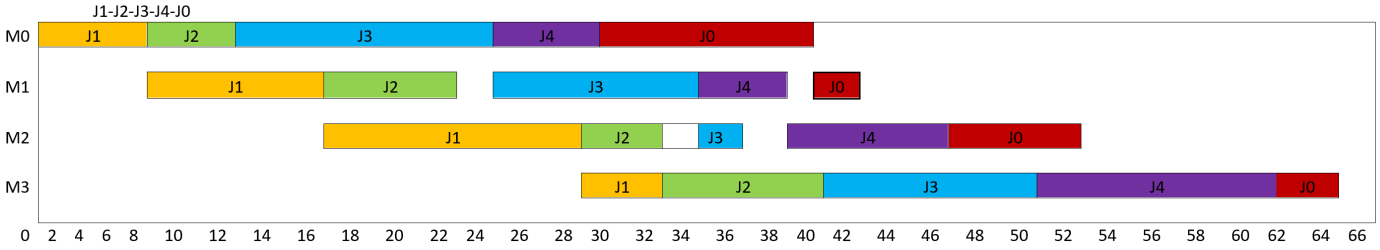

Figure 2.3: PFSP with schedule J1-J2-J3-J4-J0

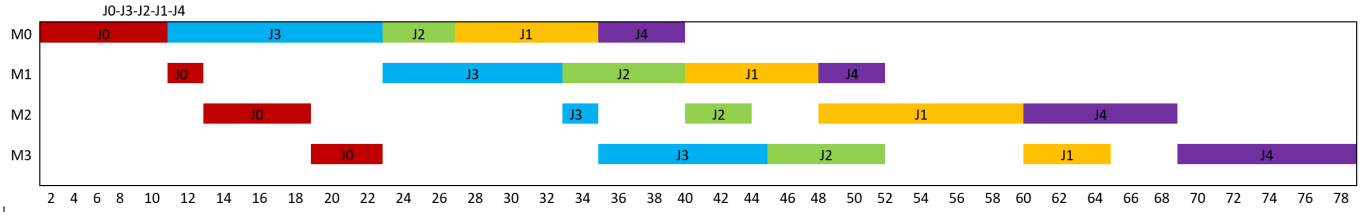

Figure 2.4: PFSP worst schedule J0-J3-J2-J1-J4

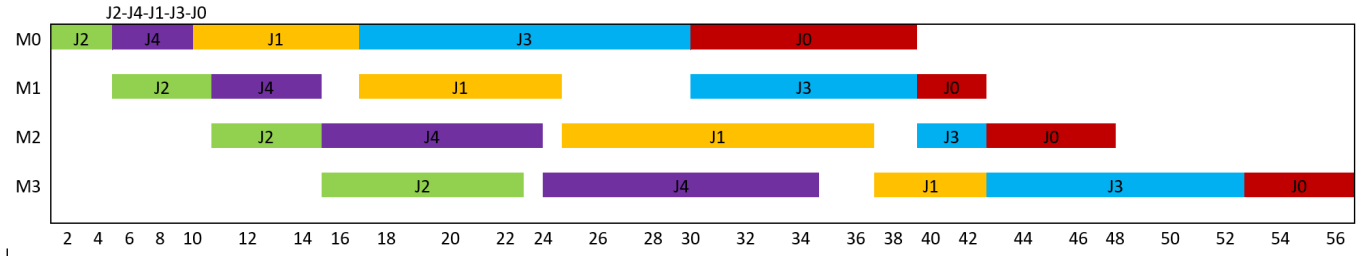

Figure 2.5: PFSP best schedule J2-J4-J1-J3-J0

Above figure 2.2, 2.3, 2.4 and 2.5 are the gantt chart showing the processing of the problem in Table 2.2 for four different schedules. The completion time of of last job J4 in 2.2 is 71. Hence the makespan(C_{max}) for schedule $J_0J_1J_2J_3J_4$ is 71. Similarly, the C_{max} in figure 2.3, 2.4 and 2.5 is 65, 78 and 56 respectively. The best solution for the problem is 56 which is obtained with sequence

$J_2J_4J_1J_3J_0$ as in figure 2.5. The worst solution for the problem is 78 with sequence $J_0J_4J_1J_3J_0$ and is shown in figure 2.4

Chapter 3

Simulated Annealing

3.1 Introduction

Simulated Annealing is one of the heuristics for Permutation Flow Shop problem. The name originates from its use in understanding metals behavior as they heat and cool. This includes heating the metal at extreme high temperature and afterward cooling gradually until it get its minimum energy state which is most regular possible configuration for the metal. If the metal is cooled too rapidly, it will end up in useless form since the heating process will move the atoms randomly changing the internal structure. This analogy can be used with job scheduling with state of the solid as feasible solutions and final form of the solid, as optimal solution. The concept was first introduced by Metropolis et al.[NM53] and then further studied by Kirkpatrick et al. [SK83] and Fetterolf and Anandalingam [PCF91] for discrete optimization models[AS96].

Simulated Annealing depends on the randomization techniques. It additional joins several perspective of iterative improvement algorithms known as neighborhood search or local search. Neighborhood search implies that there's only one thing that contrasts between the old solution and the new solution. At each iterations, two solutions are generated [H.S11] - current solution and new solution. If the solution improves, it is always accepted while some worse solutions within acceptance probability are also accepted occasionally to reach global minima by escaping local minima. The initial solution is chosen at random and the probability of accepting worse solutions depends on acceptance probability and cooling temperature.

3.1.1 Acceptance probability

Simulated Annealing occasionally accepts worse solutions within an acceptance probability. The acceptance probability depends on the temperature used and how worse the solution is. If the

temperature is high, the algorithm is likely to accept all the solution while the acceptance probability decreases with decrease in temperature and with freezing temperature, only improved solutions are accepted. The acceptance probability AP is,

$$AP = e^{-(\frac{\Delta}{T})} \tag{3.1}$$

where Δ is difference of the new solution and old solution and T is the current temperature. If value from equation 3.1 $> Random[0, 1]$, then new solution is accepted otherwise it is rejected.

3.1.2 Temperature

Temperature plays an essential role in getting way from local minima. The algorithm is kept running against different temperature to find the solution. At first, the algorithm is run at very high temperature which allows the algorithm to search in an large solution space. The temperature is then decreased gradually also called cooling rate which narrows down the solution space to allow only the improving solution.The stopping temperature also plays an importance for finding good solution. If we stop at very high temperature, we may not get a good solution and depending on problems, the stopping temperature can be different.
Simulated Annealing works in following way:

1. Set a very high temperature.
2. Select a random sequence and find the solution for the random sequence.
3. Find a new solution with neighbor sequence and compare with old solution.
4. If new solution is better accept the sequence.
5. Else calculate Acceptance probability(AP) using equation 3.1
   (a) If AP$>$ $Random[0, 1]$, accept the new solution and sequence
   (b) Else reject the solution
6. Repeat the step from 2 to 5
7. Decrease the temperature and repeat all the above steps until desired solution is obtained

The algorithm is also described with a flow chart.

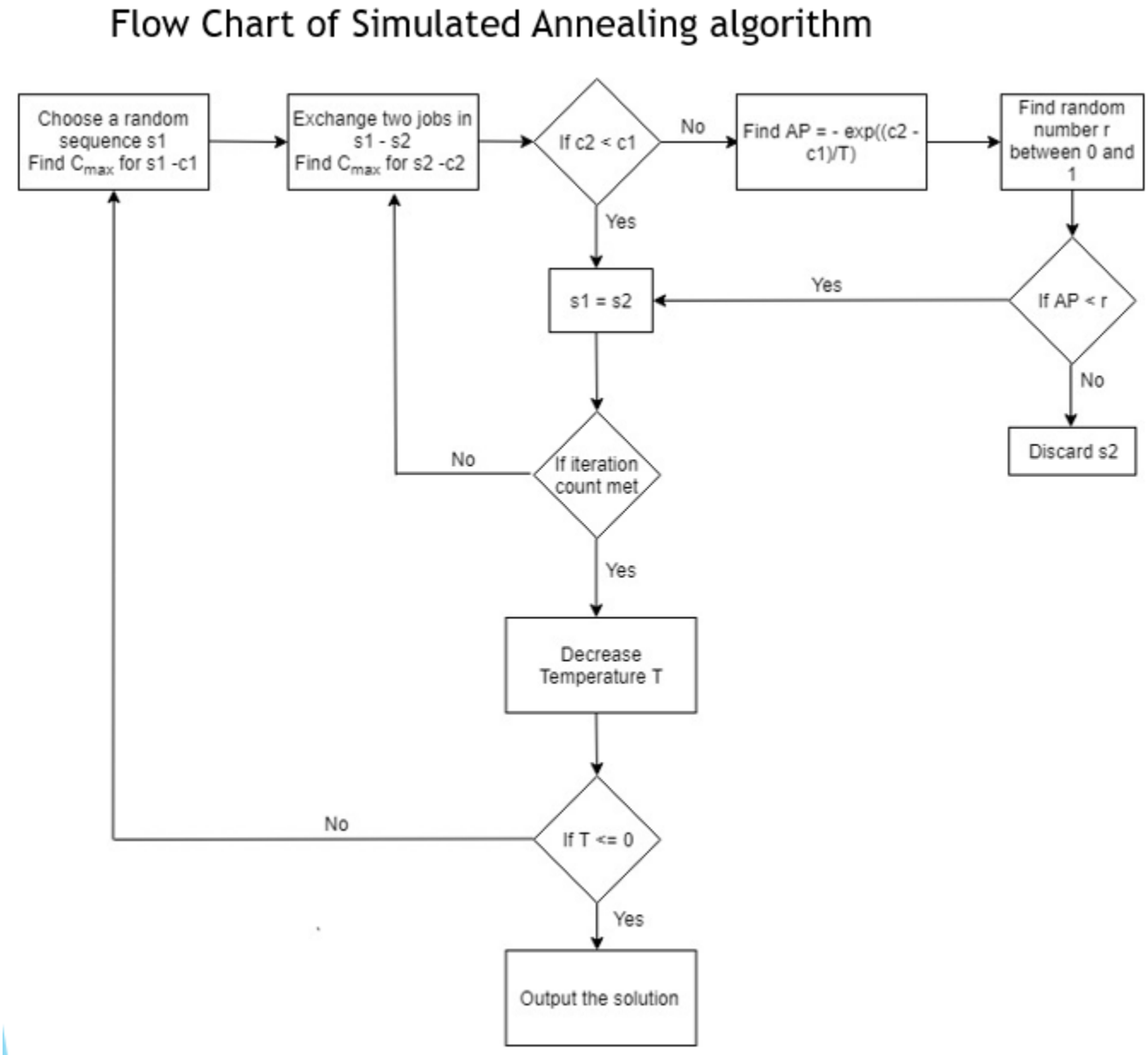

Figure 3.1: Flow chart for Simulated Annealing

3.2 Implementation

Simulated annealing is one of the heuristic to find approximate solution for Permutation Flow Shop problem. Simulated Annealing doesnt necessarily find the optimal solution, but it tries to obtain good result. Simulated annealing receives processing time of n jobs in m machine as input and gives best possible solution. The algorithm run continuously for a set of iterations to find the solution

with varying range of the temperature. The initial temperature, the cooling rate and stopping criteria play an important role to find a good solution. Initial temperature should be high to allow enough sequences to escape local minima and better the cooling rate and stopping criteria, better will be the result. If the algorithm is run for i iteration and t temperature, then total number of iterations run for Simulated Annealing is i*t.

In our implementation for Simulated Annealing, the rate at which temperature T is decreased is 0.8% and algorithm stops when temperature reaches 0. This algorithm iterates over certain number of iterations for each temperature T. The next iteration is obtained by interchanging the neighbors. Suppose initial sequence is 12345 then the next sequence is obtained by interchanging two neighbors 1 and 2 and the sequence would be 21345. At a time only two jobs would be changed, and the solution will be the smallest C_{max} value among all the temperature.

3.2.1 Terms used in the Algorithm

T- temperature
Tmax Initial temperature
Smax A high number that will be greater than largest value of the problem.
oldSequence Accepted Sequence
oldVal- C_{max} obtained from oldSequence.
newSequence new Sequence obtained with exchanging with neighbor
newVal- C_{max} for newSequence
noOfIter no of Iteration
bestSolution Best C_{max} value among all the temperature.
bestSequence - Sequence that return bestSolution

3.2.2 Algorithm

3.2.3 Assumptions

The initial value for oldSequence is {1,2, 3, n}. The algorithm maintains job sequence in an array and new sequence is created by interchanging positions of two jobs. Duplicate sequences are avoided within the same temperature. The rate for temperature decrease is 80%.

Algorithm 1 Implementation of SA algorithm

```
function SA()
    T= Tmax , bestSolution = Smax
    while (T > 0) do
        while (iterations > noOfIter) do
            oldSequence = A random sequence , find oldVal for oldSequence.
            Find newVal from newSequence.
            if (newSequence is not duplicate) then
                Compare oldVal and newVal
                if (newVal < oldVal) then
                    oldSequence = newSequence
                    oldVal = newVal
                else
                    Δ = newVal - oldVal
                    p = Random[0, 1]
                    if ( p <= e^-(Δ/T)) then
                        oldSequence = newSequence
                        oldVal = newVal
    T= 0.8*T
    if (bestSolution > oldVal) then
        bestSolution = oldVal
        bestSequence = oldSequence
```

	J0	J1	J2	J3	J4
M0	10	8	4	12	5
M1	2	8	7	10	4
M2	6	12	4	2	8
M3	4	5	7	10	11

Table 3.1: Example with 5 jobs and 4 machines

3.2.4 Example

Table 3.1 is a Flow Shop problem with five jobs and four machines. The processing time of each jobs in different machine is given as input to Algorithm 1. The initial temperature is set to 100, and best solution is set to 1000. Algorithm 1 is run for 6 iteration for each temperature. The temperature is decreased by 80%. Below is the screenshot showing how algorithm works for two temperatures.

The initial sequence for each temperature is 12345 which is the oldSequence as well as the newSequence for the algorithm. The sequence represents job sequence in current iteration and C_{max} is calculated for each iteration. At each iteration oldVal and newVal are compared; if newVal is smaller, newSequence is accepted; otherwise Acceptance probability is calculated for accepting

```
The processing time for jobs in different machine:
   J1 J2 J3 J4 J5
   ------------
M1|10 8 4 12 5
M2|2 8 7 10 4
M3|6 12 4 2 8
M4|4 5 7 10 11
********************************************************************************

Temperature:100

Sequence[1 2 3 4 5 ] Cmax:71
Sequence[2 1 3 4 5 ]Cmax:67
Sequence[3 1 2 4 5 ]Cmax:68   Random Number: 41 and Calculated probability:99
Sequence[1 3 2 4 5 ]Cmax:68   Random Number: 85 and Calculated probability:100
Sequence[2 3 1 4 5 ]Cmax:67
Sequence[3 2 1 4 5 ]Cmax:67   Random Number: 72 and Calculated probability:100
Best Value:67
################################################################################
Temperature:80

Sequence[1 2 3 4 5 ] Cmax:71
Sequence[2 1 3 4 5 ]Cmax:67
Sequence[3 1 2 4 5 ]Cmax:68   Random Number: 38 and Calculated probability:98
Sequence[1 3 2 4 5 ]Cmax:68   Random Number: 80 and Calculated probability:100
Sequence[2 3 1 4 5 ]Cmax:67
Sequence[3 2 1 4 5 ]Cmax:67   Random Number: 69 and Calculated probability:100
Best Value:67
################################################################################
```

Figure 3.2: Screenshot demonstrating how SA works

or rejecting the sequence. The acceptance probability is given as:

$$AP = e^{(-\Delta/T)},\ where\ \Delta = \text{newVal}\ \ \text{oldVal}$$

Iteration 1:

Job Sequence J_1 J_2 J_3 $J_4 J_5$ and C_{max}: 71. As this is first sequence, this is accepted, and we obtain next sequence by exchanging jobs at position 1 and 2.

Iteration 2:

Job Sequence J_2 J_1 J_3 $J_4 J_5$ and C_{max}: 67. As C_{max} for this sequence is less than the first sequence, this is accepted. Now the oldSequence is 2,1,3,4,5 and oldVal is 67. Position 1 and 2 are swapped but since this generates the same sequence as iteration 1, it is discarded, and next swapping is done

between position 1 and 3.

Iteration 3. Job Sequence $J_3\ J_1\ J_2\ J_4 J_5$ and C_{max}:68. As C_{max} of this sequence is greater than last sequence, we calculate acceptance probability P for this sequence and compares with random value between 0 and 100 is generated by program.

$\Delta = 68 - 67 = 1,\ T = 100$

$AP\ =\ e^{-(\frac{\Delta}{T})}\ =\ 0.99$

AP is then converted to whole number by multiplying with 100. Random value generated by the program is 41 which is less than AP. Hence sequence is accepted. The next sequence is obtained by interchanging two number in the sequence $J_3\ J_1\ J_2\ J_4 J_5$.

Iteration 4:

Job Sequence $J_1\ J_3\ J_2\ J_4 J_5$ and C_{max}:68. The C_{max} of this sequence is same as old sequence. The value of P is 100 and Random number generated is 85. Hence this sequence is accepted. The next sequence is obtained by exchanging two number in the sequence.

Iteration 5:

Job Sequence $J_2\ J_3\ J_1\ J_4 J_5$ and C_{max}:67. The C_{max} of this sequence is smaller than the old sequence. Hence this is accepted.

Iteration 6: Job Sequence $J_3\ J_2\ J_1\ J_4 J_5$ is obtained by exchanging first two jobs from Iteration 5 and C_max:67.This is the last iteration for the temperature so bestSolution is the C_{max} value obtained at last iteration, which is 67.

The next temperature is 80 which is 80% of the initial temperature. Again same process is repeated and the algorithm is run for 6 iterations for this temperature. After the 6^{th} iteration, the existing best solution is compared with C_{max} value of the 6^{th} iteration. If the C_{max} is better, then C_{max} will be the new bestSolution. This process is repeated until temperature reaches 0 and the best solution obtained is the best Value obtained from the algorithm.

The algorithm occasionally accepts worse value to escape local minima. In our case it has accepted all the sequences since Acceptance Probability is really high with a difference not greater than 1. Since, the algorithm runs only for 6 iterations, the sequences with both temperature is the same. But if we change the iteration count, the sequence would be different, with high temperature accepting almost all sequences and lower temperature accepting only the improving sequence.

Chapter 4

NEH

4.1 Introduction

NEH named after Nawaz, Encore and Ham [MN83] is one of the best constructive method for Permutation Flow Shop Problem [JMF04]. NEH consists of two steps. First, it find the initial sequence using some criteria and for second steps involve making final sequences by taking partial sequences from initial sequence and then reordering sequences based on makespan. Below is the algorithm for NEH:

1. Find sum of processing time of operations of jobs and then sort based on the decreasing order of the sum. This is the initial order.
2. Iteratively add jobs to partial sequence from initial order for which the makespan is minimum. This can be further divided in two steps
   (a) Choose first two jobs and find the sequence for these two jobs that minimizes the makespan.
   (b) For k=3 to n, insert k_{th} job to the sequence without altering the relative position of k-1 jobs and minimizing the partial makespan.

Assume first k-1 jobs are already sequenced. We need to add k^th job from initial sequence to sequenced jobs such that the relative position of k-1 jobs is not altered and also makespan is minimum. For k^th jobs, we can place at k different places. The best of these sequence is kept as input for next iteration. This sequence is repeated until all jobs have been ordered. The total no of the sequences that need to be generated for the NEH is $[\frac{n(n+1)}{2} - 1]$. The working process of the NEH algorithm is explained with example in section 4.1.1.

4.1.1 Example

	J0	J1	J2	J3	J4
M0	10	8	4	12	5
M1	2	8	7	10	4
M2	6	12	4	2	8
M3	4	5	7	10	11

Table 4.1: Example with 5 jobs and 4 machines

There are 5 jobs J_0, J_1, J_2, J_3, J_4 and 4 machines M_0, M_1, M_2, M_3 and each block represent the processing time of jobs in that machine. We have calculated optimal solution for the problem using brute force algorithm which is 56. We have implemented NEH for the above problem. Below is the process for finding solution with NEH for above problem.

The first step involves finding sum of jobs across all the machines.

$J_0 = 10 + 2 + 6 + 4 = 22$

$J_1 = 8 + 8 + 12 + 5 = 33$

$J_2 = 4 + 7 + 4 + 7 = 18$

$J_3 = 12 + 10 + 2 + 10 = 34$

$J_4 = 5 + 4 + 8 + 11 = 28$

The next step involves sorting the job in descending order of the sum. i.e. J_3, J_1, J_4, J_0, J_2. Choose first two jobs and find the sequence J_3 J_1 or J_1 J_3 which minimizes the makespan.

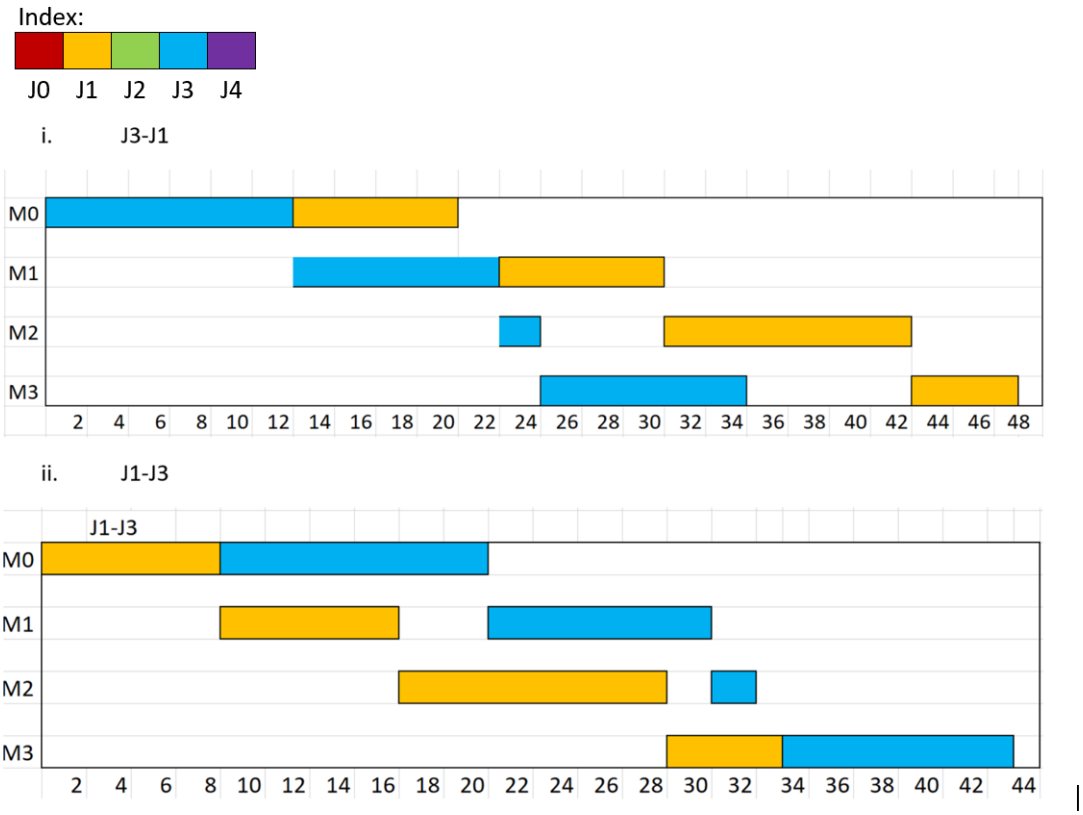

Figure 4.1: Makespan with two jobs

For two jobs J_1, J_3, J_1J_3 sequence has minimum makespan, so J_1J_3 job sequence is selected.

Now, adding third highest job i.e. J_4 to sequence J_1J_3 by keeping the position of $J_1 and J_3$ with respect to each other constant. J_4 will be added to the position that minimizes makespan for these three jobs.

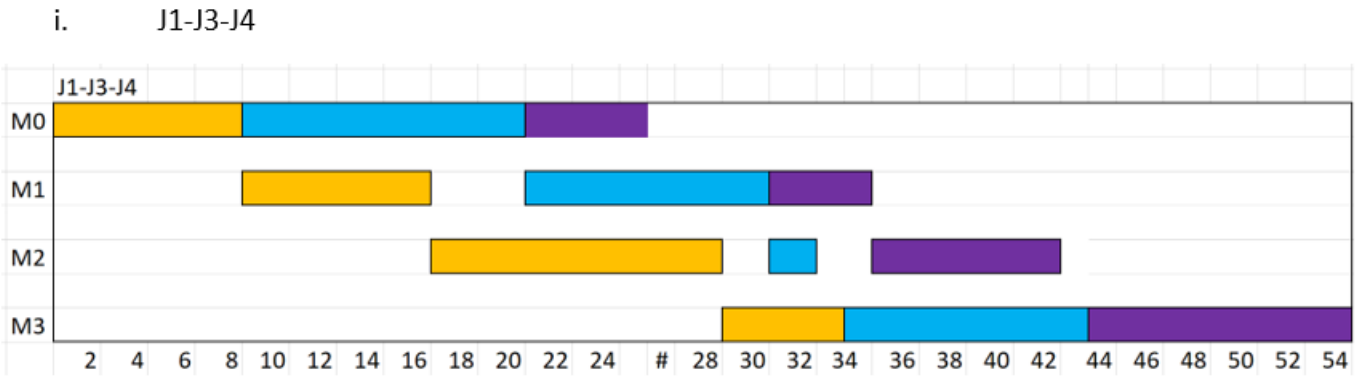

Figure 4.2: Makespan with three jobs - 1^{st} sequence

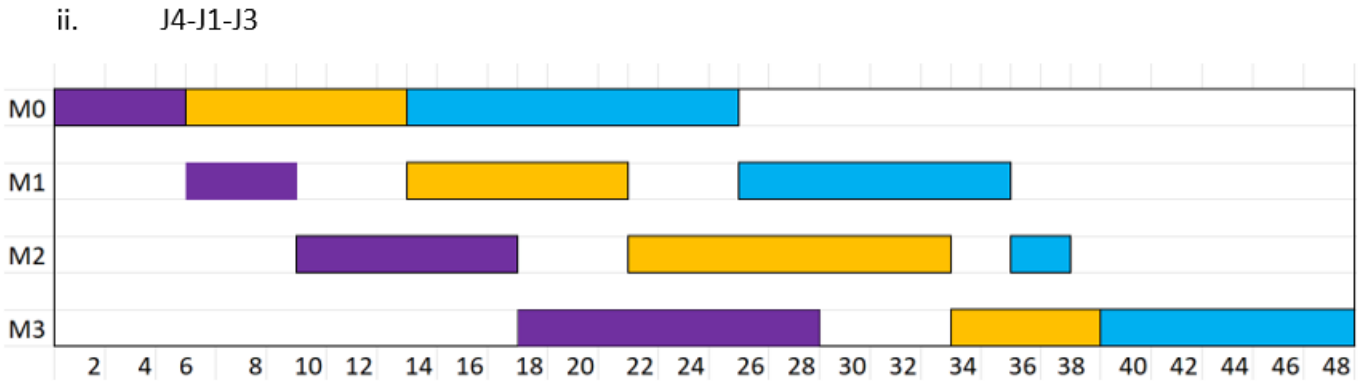

Figure 4.3: Makespan with three jobs - 2^{nd} sequence

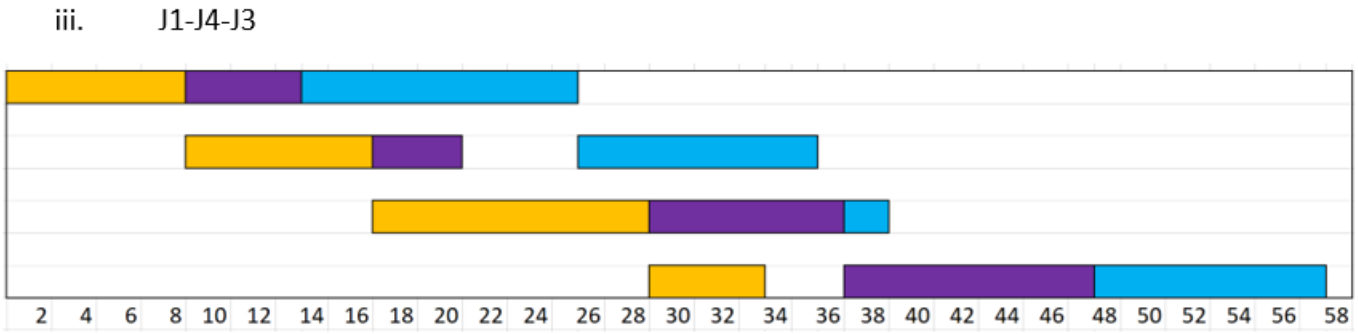

Figure 4.4: Makespan with three jobs - 3^{rd} sequence

Among the sequence $J_1J_3J_4$, $J_1J_4J_3$ and $J_1J_3J_4$; $J_4J_1J_3$ has lowest makespan. Hence $J_4J_1J_3$ sequence is selected. Now adding fourth highest job J_0 in sequence $J_4J_1J_3$.

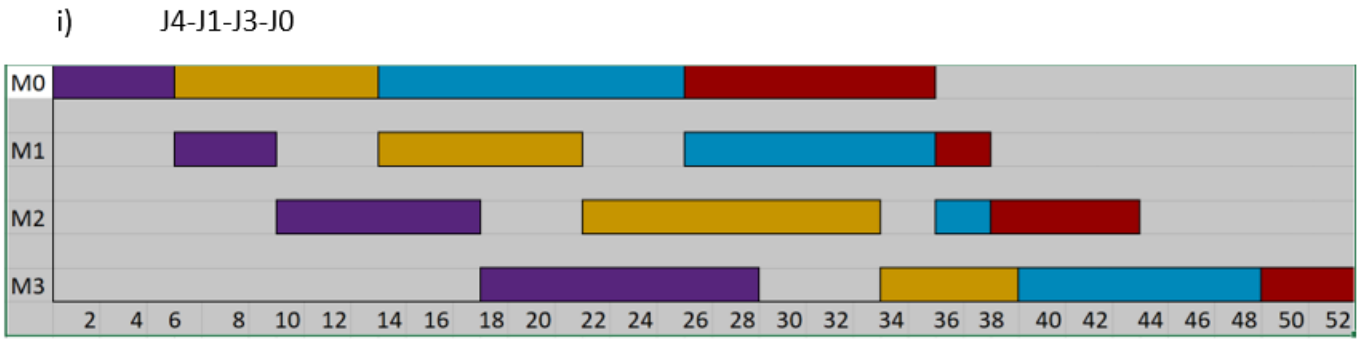

Figure 4.5: Makespan with four jobs - 1^{st} sequence

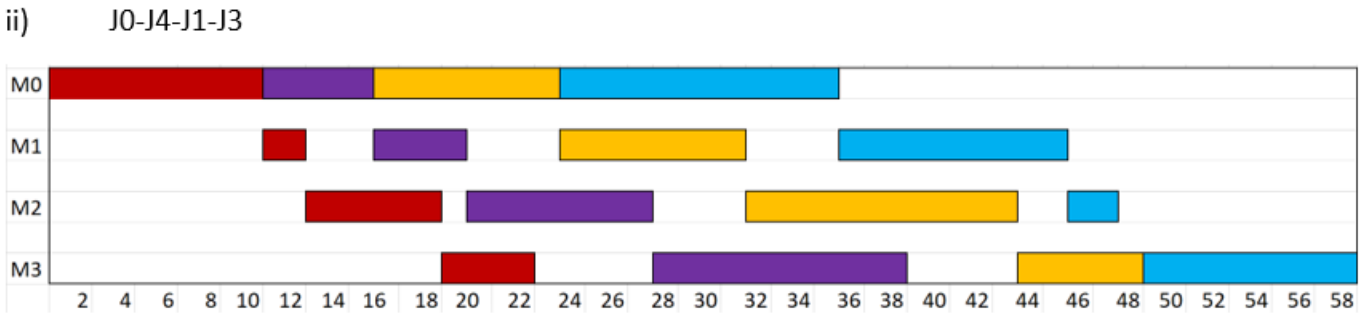

Figure 4.6: Makespan with four jobs - 2^{nd} sequence

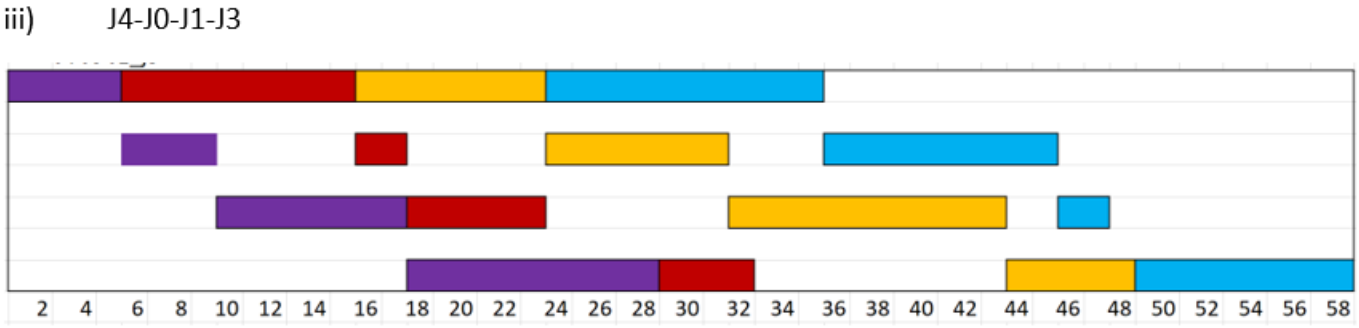

Figure 4.7: Makespan with four jobs - 3^{rd} sequence

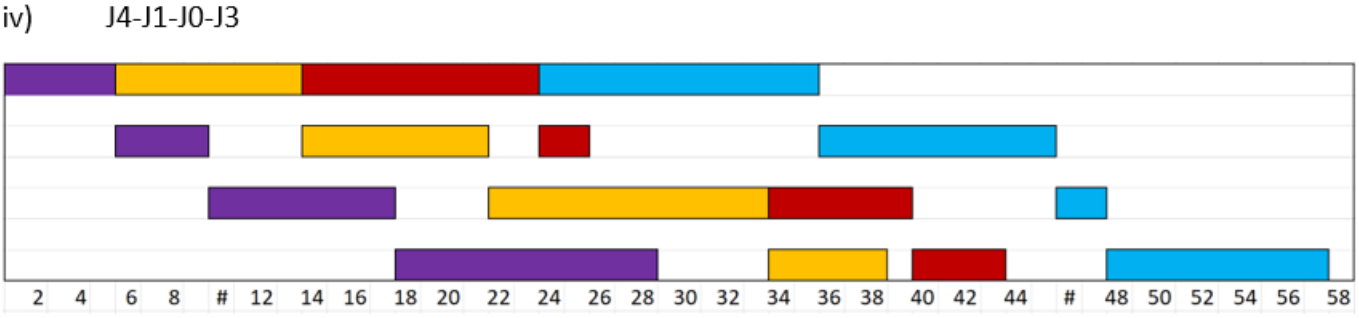

Figure 4.8: Makespan with four jobs - 4^{th} sequence

With four jobs, $J_4J_1J_3J_0$ has lowest makespan. So accepting sequence $J_4J_1J_3J_0$ for four jobs. Adding last job J_2 to find the final sequence and solution.

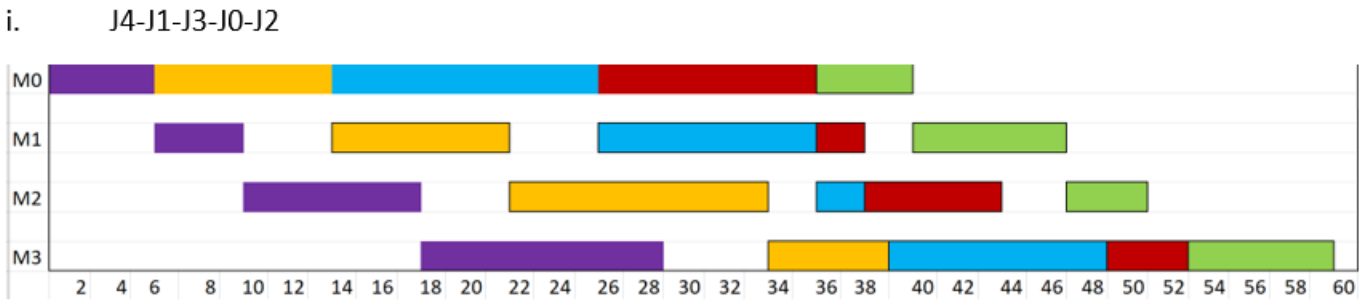

Figure 4.9: Makespan with five jobs - 1^{st} sequence

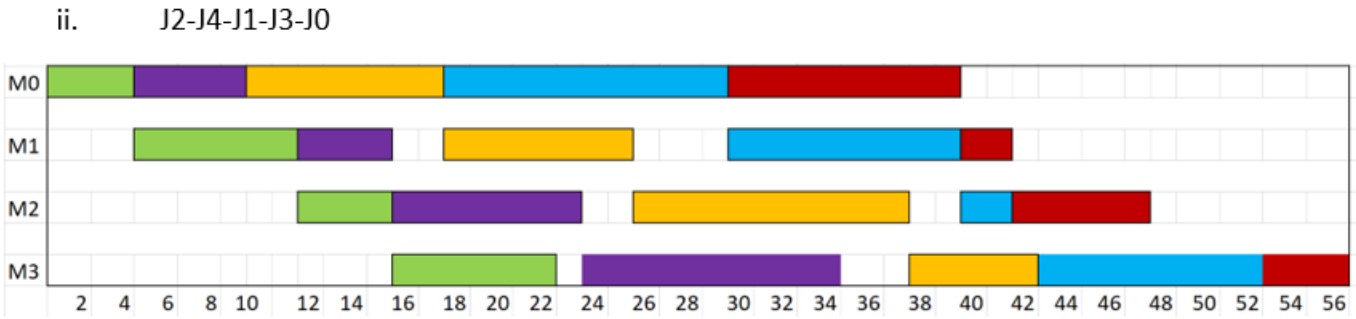

Figure 4.10: Makespan with five jobs - 2^{nd} sequence

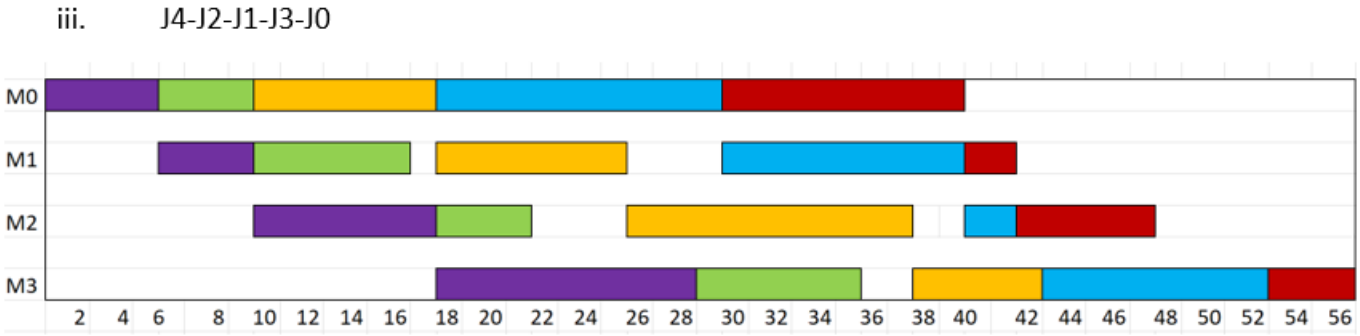

Figure 4.11: Makespan with five jobs - 3^{rd} sequence

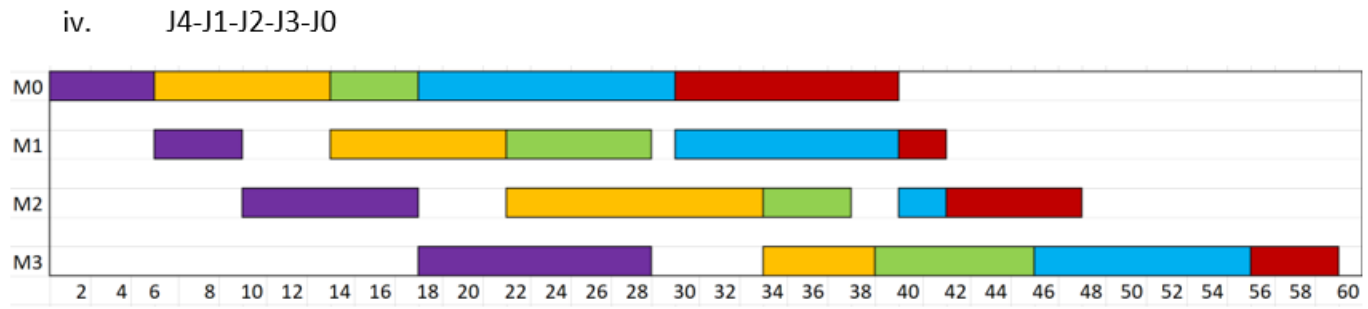

Figure 4.12: Makespan with five jobs - 4^{th} sequence

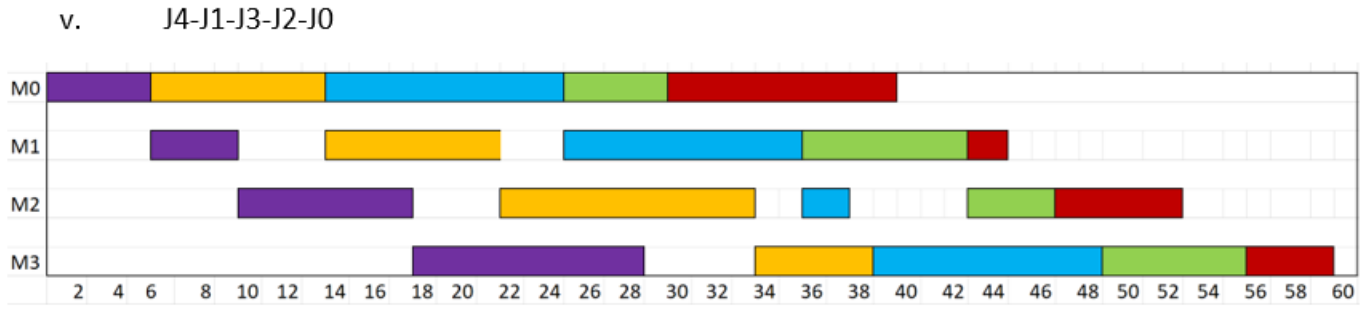

Figure 4.13: Makespan with five jobs - 5^{th} sequence

$J_2J_4J_1J_3J_0$ and $J_4J_2J_1J_3J_0$ have same C_{max} value of 56 which is optimal value. Choosing either sequence as final sequence for NEH and C_{max} for the final sequence is the solution of the problem.This is the working process of NEH. It starts with two jobs, find sequence with minimum makespan from two jobs which is input for next step. Iteratively adds new jobs to the ordered sequence and finally return the sequence with shortest makespan. The value with the final sequence is the solution for this problem

4.2 Variations of NEH

Taillard [A.B16] studied the complexity and quality of different heuristics and conclude that NEH algorithm is better for problem of job size 9 to 50 jobs. NEH performance mainly depends on the initial order according to which jobs are ordered for insertion. A number of variation were developed after initial release of the NEH algorithm by modifying the initial sequence and sorting procedure. Dong et. al. [XD08] develop a heuristic NEH-D based on Deviation that work better than NEH with same time complexity. Similarly Kalczynski and Kamburowski [PK08] developed a tie-break rule that improved the performance of NEH and was named NEH-KK. The algorithm

which works better for small problems but took more time for tie breaking rule during insertion. Framinan et. al. [JF03] construct 177 different initial orders and evaluated the performance. In this thesis we are comparing NEH result with various variation of NEH in terms of accuracy. All of the variation in this thesis has these two steps.

1. Find initial sequence based on some criterion.

2. Iteratively insert jobs to partial sequence from initial sequence.

We have used following variation of NEH

1. First variation for NEH is same as original NEH. It first find the sum of processing time of jobs. But it sorts in increasing order of jobs instead of decreasing order [JF03]. The second step is same.

2. Second variation involve calculating standard deviation for jobs in different machine [XD08]. Standard deviation is calculated for each job and then jobs are sorted on the decreasing order of standard deviation. The insertion process is same as original NEH.

3. Third form of NEH is calculated same as second but we just sort the jobs based on increasing order of the deviation.

4. Fourth NEH variation is calculated by using sum of average and standard Deviation [LSW11].The standard deviation and average of jobs is calculated for all jobs which are added. And then jobs are sorted on descending order of the sum.This is the initial sequence. The second step is same as original NEH.

4.3 NEH and Simulated Annealing combined

Modification are done to improve the results from Simulated Annealing and NEH heuristics. We changed the initial sequence for both the algorithm. At first we use the solution sequence of NEH in Simulated Annealing. Then second modification was done by using Simulated Annealing bestSequence as NEH initial sequence. The detailed process is explained in subsection 4.3.1 for using NEH final sequence in Simulated Annealing and 4.3.2 for using Simulated Annealing bestSequence in NEH.

4.3.1 Using NEH result in Simulated Annealing

1. Find the resulting sequence for given problem using NEH algorithm.

2. Use that sequence as initial oldSequence in Simulated Annealing algorithm.

3. Run the Simulated Annealing algorithm and find the solution.

4.3.2 Using Simulated Annealing result in NEH

1. Find the bestSequence using Simulated Annealing algorithm.

2. Skip first step of NEH algorithm. Instead hard code the sequence from step 1.

3. Run the NEH algorithm and find the solution.

Chapter 5

Results

Many Permutation Flow Shop problems are used to test the performance and optimality of Simulated Algorithm, NEH algorithm, variations of NEH and combination of Simulated Annealing and NEH. To test the performance of the algorithm, result obtained from the algorithms needs to be compared with Optimal Solution for the problem. For NP Hard problems, optimal solutions are not always known. For this thesis we are using the problems for which optimal solution are known. We have used VFR instances by Eva Vallada, Roben Ruiz, Jose M. Framinan [EV15] and Taillard instances [E.T93]. All the coding is done in C++ using Intel Core i7 processor and 8 GB RAM.

5.1 Performance testing parameters

Execution Time

It is the time taken by the implemented algorithms to find the final solution in the machine.

Optimal Solution

Optimal Solution is the minimum C_{max} that can be achieved from a given Permutation Flow Shop problems. For n job problem, there are n! total sequences in Permutation Flow Shop. After getting C_{max} for all the sequences, the minimum C_{max} is optimal solution and the sequence for which we get optimal value is Best Sequence.

Approximate Ratio

$$ApproximateRatio = \frac{Solution}{Optimal\ Solution}$$

Solution is the final C_{max} value returned from the algorithm implemented. Approximate ratio is used to analyze the performance of algorithm mentioned in chapter 3 and chapter 4 and is the

percentage deviation from the optimal value. Algorithms implemented in this thesis are heuristics to find approximate solution. Approximate ratio calculates the percentage of increase from optimal solution.

5.2 VRF Instances

Vallada et. al. [EV15] created benchmark for Flow Shop scheduling problems with 480 instances; 240 large instances and 240 small instances. Benchmark generation and analysis took around six year of CPU time effort. Extensive generations and computational experiments were done to demonstrate that the propose benchmark is harder than most of the instances proposed before.

Small instances are set of 240 with machine size as {5,10,15,20} and job size as {10, 20, 30, 40, 50, 60}. Similarly large instances are set of 240 with machine size as {20,40,60} and job size {100, 200, 300, 400, 500, 600, 700, 800}. The lower bound and upper bound of all the instances are provided and a website is also available to record all the instances, solutions and bounds.

The upper bound provided for the problem is Optimal Solution for Permutation Flow Shop problem which we have checked by using Brute Force algorithm. For this thesis we are only using Optimal solution for comparison of our result. We have selected 20 instances, 4 of them are from Large Instance set and remaining from Small Instances. The results is divided into four section. First section involve results using Simulated Annealing algorithm implemented in chapter 3. Second section include result with NEH and all the variation of NEH in chapter 4. In the third section we have have done comparison of Simulated Annealing and NEH side by side based on Execution time and Approximate Ratio. The last section includes the result from the combined algorithm in section 4.3 of Chapter 4.
The results are obtained based on the absolute C_{max} value as well as Approximate Ratio.

5.2.1 Observations

Simulated Annealing results with different iterations

The first column in all the tables are the Instance name with job size and machine size. E.g. For VFR20_10_10_Gap, first number after VFR is job size, the second number is machine size and third number represents instance number. Second column in the Table 5.1 represents the optimal solution for the problem. Third column, fourth and fifth column in Table 5.1 represent the absolute value using Simulated Annealing algorithm with 500, 1000 and 2500 iterations(noOfIter) respectively.

The second, third and fourth column in Table 5.2 shows the approximate ratio of the S.A with different iterations.The value 1.09 for first instance means the absolute value deviates the optimal solution by 9%.

Instance	**OptimalVal**	**500iters**	**1000iters**	**2500iters**
VFR10_10_1_Gap	1097	1199	1173	1165
VFR10_10_10_Gap	1099	1164	1149	1150
VFR10_15_1_Gap	1307	1423	1414	1414
VFR10_15_10_Gap	1461	1567	1566	1566
VFR20_10_10_Gap	1489	1628	1623	1619
VFR20_10_2_Gap	1525	1777	1764	1706
VFR20_15_1_Gap	1936	2191	2179	2131
VFR20_20_10_Gap	2199	2543	2544	2503
VFR30_5_1_Gap	1805	2080	1993	1976
VFR30_20_10_Gap	2805	3375	3372	3299
VFR40_20_9_Gap	3335	4333	4159	4165
VFR40_20_10_Gap	3122	3897	3871	3774
VFR50_10_10_Gap	3056	3811	3811	3701
VFR50_20_10_Gap	3769	4865	4755	4677
VFR60_5_1_Gap	3350	3948	3948	3948
VFR60_20_1_Gap	4163	5236	5168	5082
VFR100_20_10_Gap	6145	7619	7502	7475
VFR200_40_10_Gap	13228	16706	16474	16165
VFR300_20_10_Gap	16899	20111	19951	19906
VFR400_20_6_Gap	21214	25062	24856	24778

Table 5.1: Absolute value using Simulated Annealing algorithm with different number of iterations

Instance	500iters	1000iters	2500iters
VFR10_10_1_Gap	1.093	1.069	1.062
VFR10_10_10_Gap	1.059	1.045	1.046
VFR10_15_1_Gap	1.089	1.082	1.082
VFR10_15_10_Gap	1.073	1.072	1.072
VFR20_10_10_Gap	1.093	1.090	1.087
VFR20_10_2_Gap	1.165	1.157	1.119
VFR20_15_1_Gap	1.132	1.126	1.101
VFR20_20_10_Gap	1.156	1.157	1.138
VFR30_5_1_Gap	1.152	1.104	1.095
VFR30_20_10_Gap	1.203	1.202	1.176
VFR40_20_9_Gap	1.299	1.247	1.249
VFR40_20_10_Gap	1.248	1.240	1.209
VFR50_10_10_Gap	1.247	1.247	1.211
VFR50_20_10_Gap	1.291	1.262	1.241
VFR60_5_1_Gap	1.179	1.179	1.179
VFR60_20_1_Gap	1.258	1.241	1.221
VFR100_20_10_Gap	1.240	1.221	1.216
VFR200_40_10_Gap	1.263	1.245	1.222
VFR300_20_10_Gap	1.190	1.181	1.178
VFR400_20_6_Gap	1.181	1.172	1.168

Table 5.2: Approximate ratio of Simulated Annealing with different iterations

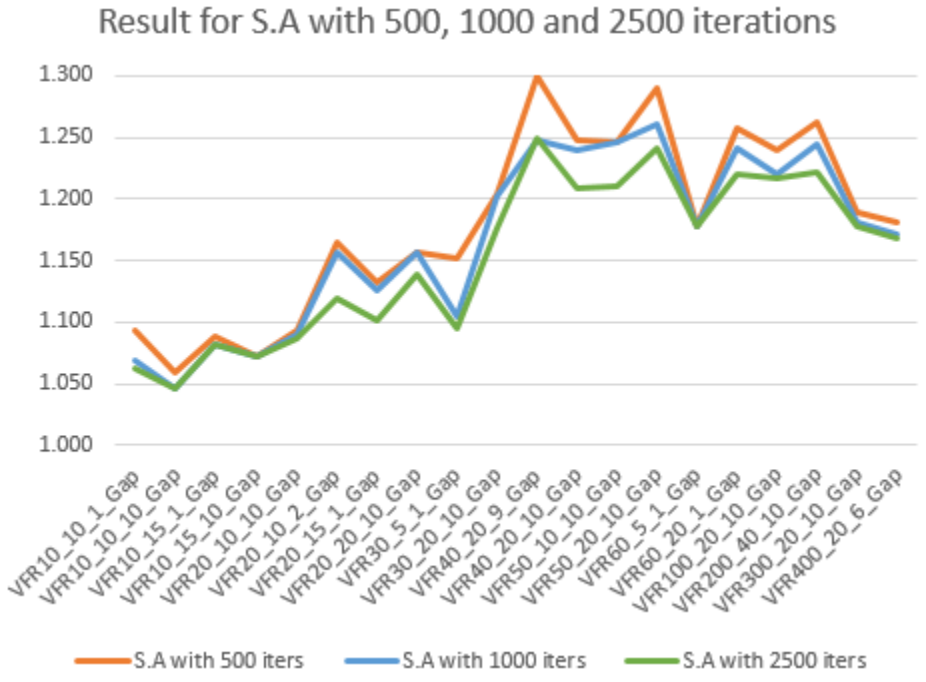

Figure 5.1: Approximate ratio for Simulated Annealing by changing the number of iterations

From the results obtained, it can be observed as the problem size gets bigger, the solution becomes worse. For Simulated Annealing, the number of neighbor observed is the iteration count and is always same no matter how big the problem size is; so it might never meet the optimal solution. But as we increase the number of iterations from 500 to 2500, the result got better and better as Simulated Annealing is able to search larger solution space. We have plotted the result in figure 5.1 for visual representation. The lowest deviation among three is with Iteration count 2500, then with 1000 and Simulated Annealing with iterations count 500 deviates the most from the optimal solution. Even though Simulated Annealing with iteration count 2500 beats the other two, the value is far from the optimal result.

Results from NEH and its Variation

Instance	**SumDes**	**SumAsc**	**StdDes**	**StdAsc**	**Std+Avg**
VFR10_10_1_Gap	1.118	1.081	1.118	1.081	1.118
VFR10_10_10_Gap	1.086	1.138	1.086	1.096	1.067
VFR10_15_1_Gap	1.082	1.118	1.082	1.121	1.082
VFR10_15_10_Gap	1.075	1.086	1.075	1.089	1.078
VFR20_10_10_Gap	1.117	1.179	1.117	1.125	1.086
VFR20_10_2_Gap	1.079	1.082	1.079	1.095	1.077
VFR20_15_1_Gap	1.111	1.106	1.111	1.133	1.089
VFR20_20_10_Gap	1.128	1.160	1.128	1.146	1.129
VFR30_5_1_Gap	1.016	1.022	1.016	1.028	1.017
VFR30_20_10_Gap	1.107	1.183	1.107	1.164	1.111
VFR40_20_9_Gap	1.103	1.139	1.103	1.147	1.113
VFR40_20_10_Gap	1.116	1.142	1.116	1.135	1.120
VFR50_10_10_Gap	1.067	1.062	1.067	1.098	1.053
VFR50_20_10_Gap	1.115	1.145	1.115	1.127	1.099
VFR60_5_1_Gap	1.003	1.037	1.003	1.027	1.003
VFR60_20_1_Gap	1.123	1.131	1.123	1.122	1.102
VFR100_20_10_Gap	1.094	1.104	1.094	1.104	1.091
VFR200_40_10_Gap	1.113	1.124	1.113	1.116	1.119
VFR300_20_10_Gap	1.028	1.042	1.030	1.039	1.028
VFR400_20_6_Gap	1.038	1.051	1.035	1.049	1.038

Table 5.3: Approximate ratio obtained using various variation of NEH

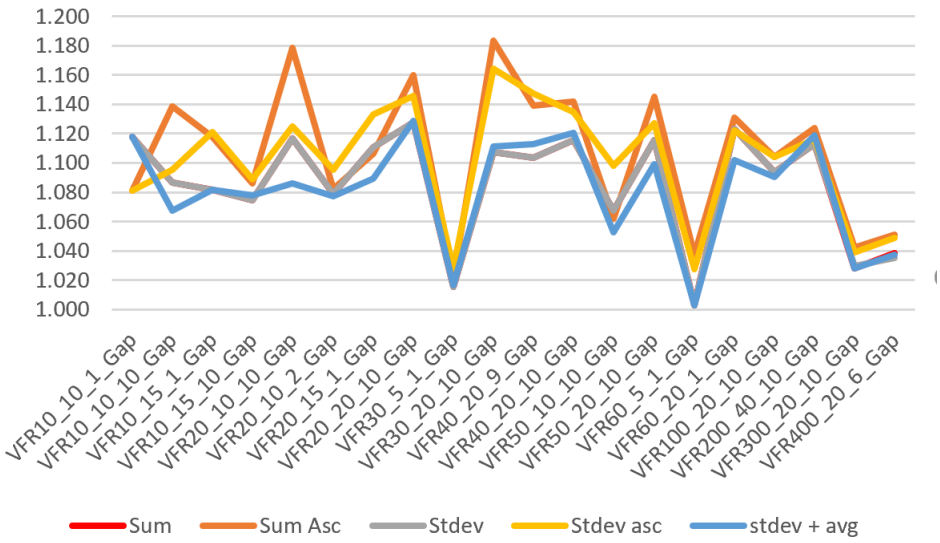

Figure 5.2: NEH and its variation

Table 5.3 shows the approximate ratio for NEH and its variation with VFR instances. The second column in Table represents NEH result. Third, fourth, fifth and sixth column shows the result with NEH variation 1, 2, 3, 4 respectively from Chapter 4.

Comparing NEH and its various variation based on approximate ratio, variations for which initial sequence is obtained by sorting in ascending order i.e. variation 1 and variation 3 works poorly compared to the initial sequence obtained by sorting in descending order. The results are plotted in figure 5.2. In the figure as well as Table 5.3, NEH and NEH variation 2(StdDesc) have almost same values and are overlapping. Also, the result from NEH variation 4, obtained as sum of Standard Deviation and Average performed better than original NEH for most of the cases. Hence, while comparing NEH and all its variation for the given problem, variation 4 performed the best for VFR instances.

Simulated Annealing and NEH results

In this section, we are comparing performance of Simulated Annealing and NEH algorithm based on approximate ratio as well as execution time. The second column in Table 5.5 shows the approximate ratio from Simulated Annealing algorithm with 500 iterations, third column is approximate ratio for NEH, fifth and sixth column shows the Execution time of Simulated Annealing and NEH algorithm respectively. The execution time is in milliseconds.

Instance	SA	NEH	SA time(milisecs)	NEH time(milisecs)
VFR10_10_1_Gap	1.093	1.118	26826	22
VFR10_10_10_Gap	1.059	1.086	25332	23
VFR10_15_1_Gap	1.089	1.082	25622	33
VFR10_15_10_Gap	1.073	1.075	36661	35
VFR20_10_10_Gap	1.093	1.117	72762	114
VFR20_10_2_Gap	1.165	1.079	45181	122
VFR20_15_1_Gap	1.132	1.111	55881	139
VFR20_20_10_Gap	1.156	1.128	57727	187
VFR30_5_1_Gap	1.152	1.016	70825	184
VFR30_20_10_Gap	1.203	1.107	62092	446
VFR40_20_9_Gap	1.299	1.103	87480	781
VFR40_20_10_Gap	1.248	1.116	98672	760
VFR50_10_10_Gap	1.247	1.067	93922	817
VFR50_20_10_Gap	1.291	1.115	104816	1309
VFR60_5_1_Gap	1.179	1.003	127997	880
VFR60_20_1_Gap	1.258	1.123	135068	2092
VFR100_20_10_Gap	1.240	1.094	202314	7527
VFR200_40_10_Gap	1.263	1.113	374608	152823
VFR300_20_10_Gap	1.190	1.028	426026	315313
VFR400_20_6_Gap	1.181	1.038	543329	351587

Table 5.4: NEH and Simulated Annealing with approximate ratio and completion time

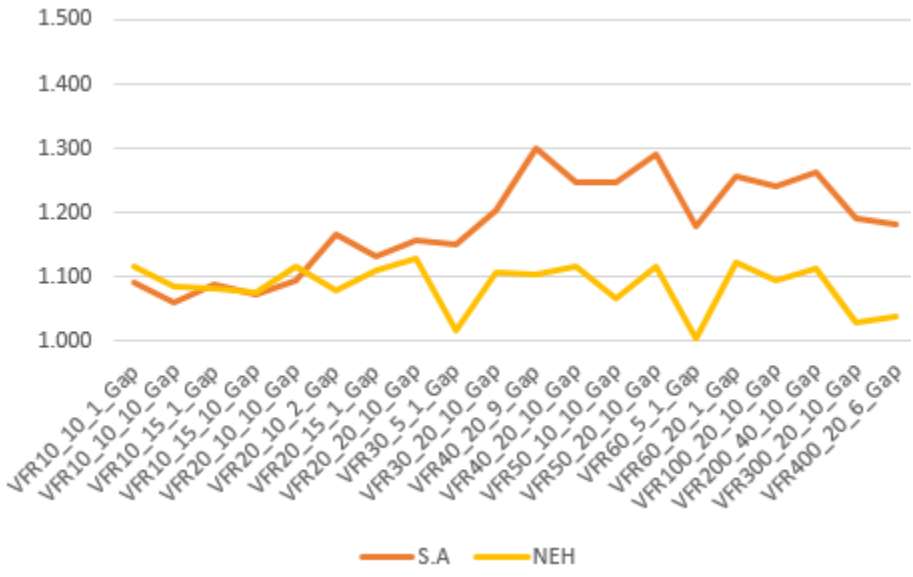

Figure 5.3: Approximate ratio with Simulated Annealing and NEH

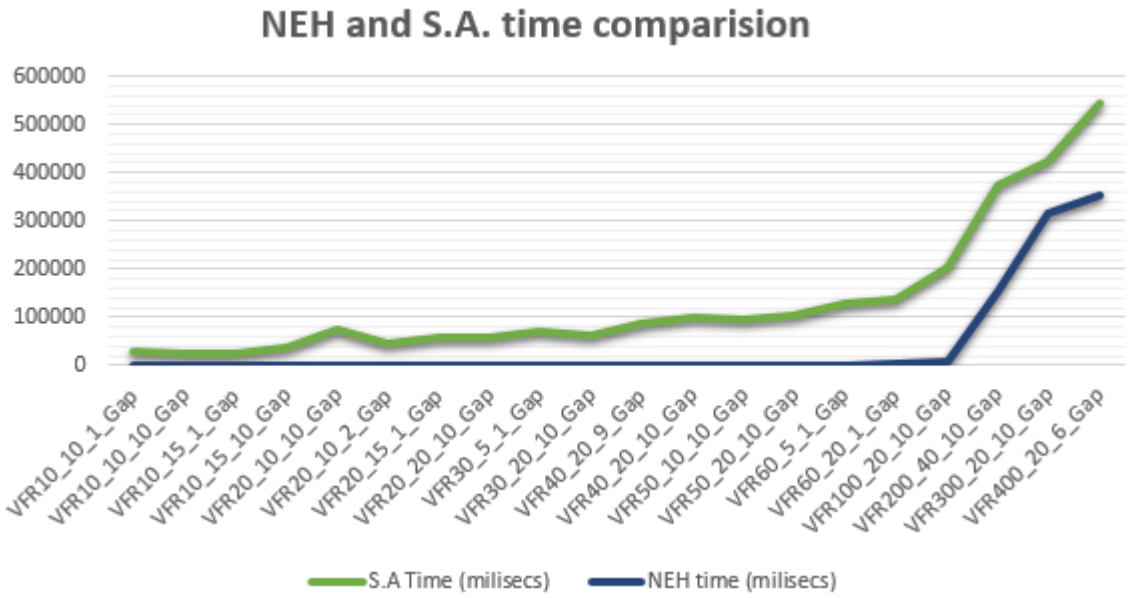

Figure 5.4: Time taken by NEH and Annealing for VFR Instances

Observing the result based on the approximate ratio, NEH algorithm performs much better than Simulated Annealing in solution as well as execution time. Initially for some small problem size, Simulated Annealing results are nearly same as NEH. However when the problem size increases, the performance of Simulated Annealing goes worse. For e.g. for VFR40_20_9, Simulated Annealing is off by 30% where NEH is off with only 10% which is much lesser than Simulated Annealing. Some of the NEH results are near to the Optimal Solution but Simulated Annealing results are farther from Optimal and NEH solutions. The comparison between the two can be visually represented in the figure 5.3.

Also, comparing NEH and Simulated Annealing based on the execution time, NEH outperforms Simulated Annealing. The difference is more than 100 times on average for VFR instances. This is visually represented in figure 5.4.

Result with Simulated Annealing and NEH combined

We tested the VFR instances using modification algorithm implemented in subsection 4.3.1 and 4.3.2 of Chapter 4. In this section we have compared modified algorithm with original algorithm and observed the improvement. Table 5.5 shows the result with respect to the absolute value and Figure 5.5 and figure 5.6 shows the improvement with respect to original algorithm.

Instance	SA	SA using NEH	NEH	NEH using SA
VFR10_10_1_Gap	1199	1199	1226	1239
VFR10_10_10_Gap	1164	1161	1194	1164
VFR10_15_1_Gap	1423	1425	1414	1414
VFR10_15_10_Gap	1567	1567	1570	1597
VFR20_10_10_Gap	1628	1635	1663	1654
VFR20_10_2_Gap	1777	1677	1645	1668
VFR20_15_1_Gap	2191	2189	2150	2095
VFR20_20_10_Gap	2543	2574	2480	2518
VFR30_5_1_Gap	2080	1900	1833	1909
VFR30_20_10_Gap	3375	3259	3106	3205
VFR40_20_9_Gap	4333	3698	3680	3749
VFR40_20_10_Gap	3897	3570	3483	3541
VFR50_10_10_Gap	3811	3321	3262	3298
VFR50_20_10_Gap	4865	4401	4204	4168
VFR60_5_1_Gap	3948	3431	3360	3374
VFR60_20_1_Gap	5236	4764	4677	4672
VFR100_20_10_Gap	7619	6794	6720	6698
VFR200_40_10_Gap	16706	16627	14718	14812
VFR300_20_10_Gap	20111	17391	17376	17474
VFR400_20_6_Gap	25062	22119	22025	22036

Table 5.5: Simulated Annealing and NEH solution along with combined results

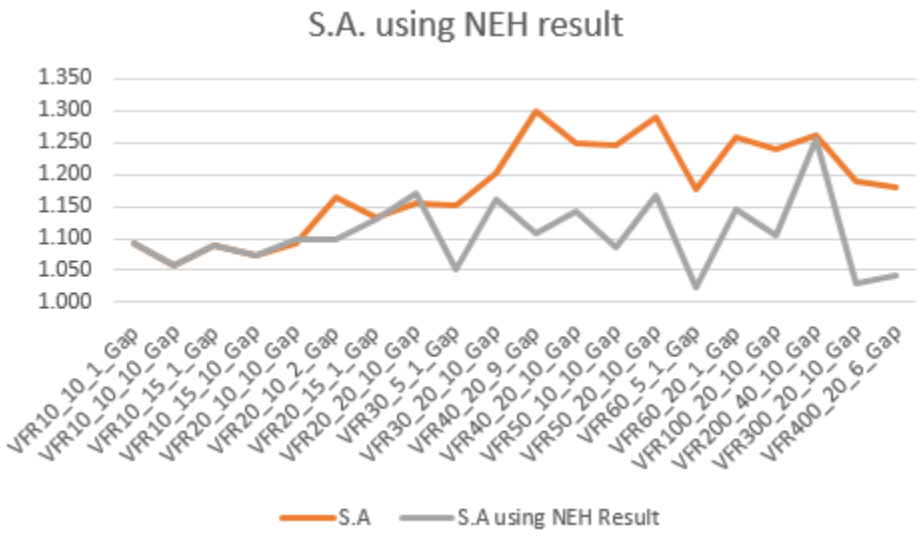

Figure 5.5: Graph with Approximate ratio of SA and modified SA

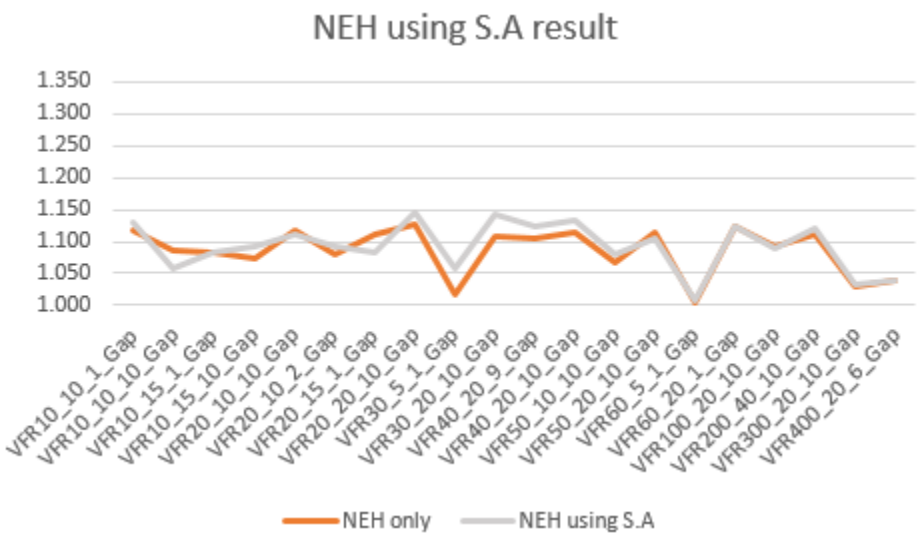

Figure 5.6: Graph with Approximate ratio of NEH and modified NEH

In table 5.5 the first column is the instance name, second is the result using Simulated Annealing algorithm with 500 iterations, third column is the result with modification of Simulated Annealing algorithm with initial sequence from NEH final sequence, fourth column is result from NEH algorithm and fifth column is the result from NEH modification i.e. NEH using bestSequence of Simulated Annealing algorithm. While comparing the results of Simulated Annealing with and without modification, Simulated Annealing result improves a lot with the modification and is near to NEH result.

However same thing cannot be said for NEH. The result for NEH with modification is worse compared to NEH in most of the cases. The comparison for Simulated Annealing and NEH can be visually seen from figure 5.5 and 5.6 respectively.

5.3 Taillard's Instances

Taillard is well known researcher in the field of Shop Problems. He have performed several experiments on shop problems. He has created benchmark for many hard instances Shop problems. In this thesis, we have selected 20 instances of hard problems in category of Flow Shop problems. He has provided upper bound and lower bound for the problems. We have used modified Simulated Annealing algorithm for the instances and discarded the modified NEH algorithm as no improvement was seen with modified NEH algorithm. We have used our modified algorithm and with Taillard's standard result along with NEH and Simulated Annealing results.

5.3.1 Observations

First column in Table 5.6 is the Taillard instance number with job size and machine size. We have taken 20 jobs problem with 5 and 10 machines. We have 10 instances with 20 jobs 5 machines and 10 instances from 20 jobs 10 machines. Second and Third column is the Upper and Lower bound of Taillard's experiment. Column 4 and column 5 are the solution obtained by using NEH algorithm and Simulated Annealing with 500 iterations. The last column is the result obtained using modified Simulated Annealing algorithm.

Instances	**Upper Bound**	**Lower Bound**	**NEH**	**SA**	**Modified SA**
20_5_1	1278	1232	1286	1344	1286
20_5_2	1359	1290	1365	1406	1365
20_5_3	1081	1073	1159	1459	1159
20_5_4	1293	1268	1325	1534	1325
20_5_5	1236	1198	1305	1431	1305
20_5_6	1195	1180	1228	1498	1224
20_5_7	1239	1226	1278	1267	1251
20_5_8	1206	1170	1223	1374	1223
20_5_9	1230	1206	1291	1334	1277
20_5_10	1108	1082	1151	1297	1151
20_10_1	1582	1448	1680	1775	1651
20_10_2	1659	1479	1729	1852	1729
20_10_3	1496	1407	1557	1660	1531
20_10_4	1378	1308	1439	1565	1406
20_10_5	1419	1325	1502	1578	1493
20_10_6	1397	1290	1453	1877	1433
20_10_7	1484	1388	1562	1726	1526
20_10_8	1538	1363	1609	1743	1609
20_10_9	1593	1472	1647	1735	1631
20_10_10	1591	1356	1653	1802	1653

Table 5.6: Results with Taillard Instances

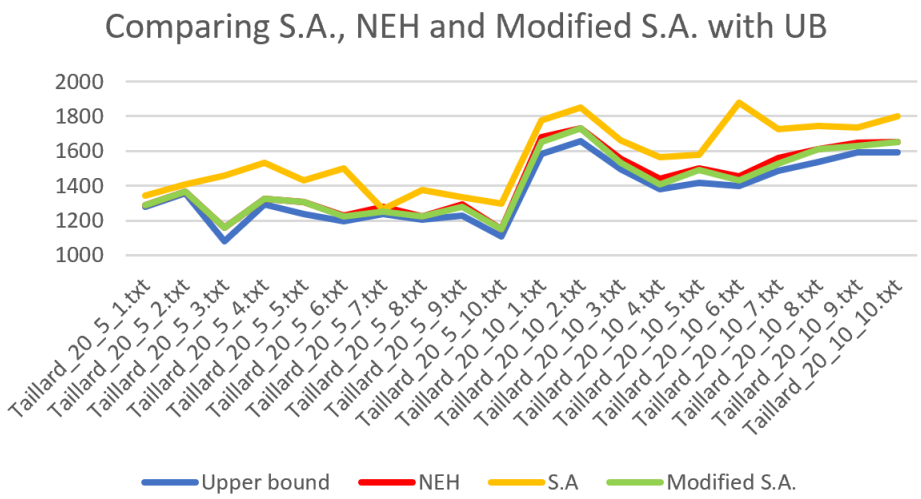

Figure 5.7: Graph with results for Taillard instances

The results shows that the modified algorithm for NEH have worked really well for Taillard's instances as well. Figure 5.7 shows the comparison of original Simulated Annealing, NEH and modified Simulated Annealing with Taillard result. Simulated Annealing still works poorly even for Taillard instances. NEH and modified Simulated Annealing worked really well with some instances overlapping with Upper bound. However, both NEH and modified Annealing are close to Upper bound but still away from the lower bound.The improvement percentage of modified Simulated Annealing over Simulated Annealing is shown in the figure 5.8.

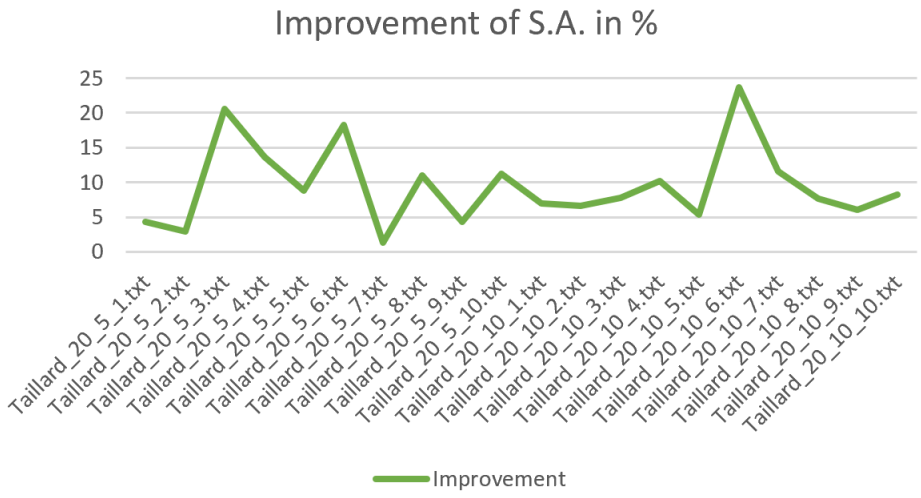

Figure 5.8: Percentage improvement of Simulated Annealing after modifying initial sequence

Chapter 6

Conclusion and Future Work

In this thesis, we have implemented Simulated Annealing algorithm in Chapter 3, NEH algorithm and its variation in Chapter 4 and combined Simulated Annealing and NEH in Chapter 4. Chapter 5 includes the findings with the algorithms implemented. We have tested the efficiency of algorithm implemented with the standard results. We compared existing algorithms Simulated Annealing and NEH based on final result and execution time and also compared combined algorithm with the existing.

The first comparison is done with Simulated Annealing by changing the number of iterations. The comparison in subsection 5.2.1 shows that as we increase the number of the iterations, better is the result. However execution time also increases as number of iterations increases. Even the solution improved by increasing the iterations but still with highest iteration, the result is far from optimal solution. Second comparison is with the NEH and its variation .NEH variation where the initial sequence is sorted in ascending order perform worse than the variation where initial sequence are sorted in descending order in subsection 5.5. Among all the NEH variation, NEH where initial sequence is obtained by finding sum of Standard Deviation and Average performed best.

Third comparison is between Simulated Annealing and NEH based on the final result as well as execution time. Comparison of NEH and Simulated Annealing with VFR problems in 5.2.1 as well as Taillard's instances in 5.3.1 shows that NEH outperforms Simulated Annealing in both C_{max} value as well as execution time. However, we are able to improve the solution of Annealing algorithm using NEH sequence. Improved Simulated Annealing works really well when tested with VRF instances 5.2.1 as well as Taillard instances 5.3.1. The result obtained with modified Simulated Annealing and NEH are almost similar and sometimes better than NEH. We tried to improve

NEH with Simulated Annealing result. But, the result 5.2.1 is worse(5.2.1) when compared to original NEH.

From all the comparisons between Annealing and NEH, NEH beats Annealing. Simulated Annealing result got improved after using NEH sequence and results were similar with NEH but execution time was same. Hence, NEH worked best with the problems included in this thesis.

Further improvement can be done with both Simulated Annealing and NEH. For Simulated Annealing, further improvement can be done in terms of execution time. Simulated Annealing is taking long time to complete the execution. As the number of iterations increases, the execution time also increases at the same level. The temperature and cooling rate plays a vital role in determining the execution time. For this thesis, the temperature and cooling rate are constant for all the problems. So, we can change temperature and cooling rate to reduce execution time without impacting the result.

We modified NEH by combining Annealing result but it couldn't improve the solution for NEH. For future, we can combine NEH with some other algorithms like Genetic Algorithm, Tabu Search, Branch and Bound, etc. for improving the result.

Appendix A

Selected Code

```
\section{NEH}
This is used to find the partial makespan of a schedule. This takes partial matrix and column s
int find_makespan(vector<vector<int>> matrix, int val) {
int i, j;
int column = val + 1; //
int** makespan_matrix = new int*[rows];
for (int i = 0; i < rows; ++i)
makespan_matrix[i] = new int[column];
makespan_matrix[0][0] = machinejobs[0][permutation[0] - 1];
//Calculates the completion time of jobs in first machine;
for (j = 1; j < cols; j++)
makespan_matrix[0][j] = m[0][j - 1] + machinejobs[0][permutation[j] - 1];
//calculates the completion time of first job in all machine
for (j = 1; j < rows; j++) {
makespan_matrix[j][0] = makespan_matrix[j - 1][0] + machinejobs[j][permutation[
}
44
//calculates the completion time for all the machines
for (i = 1; i < rows; i++) {
makespan_matrix[i][1] = makespan_matrix[i - 1][1] + machinejobs[i][permutation[
for (j = 1; j < cols; j++) {
if (makespan_matrix[i - 1][j] >= makespan_matrix[i][j - 1]) {
makespan_matrix[i][j] = makespan_matrix[i - 1][j] + machinejobs[i][permutation[j]
```

```
}
else {
makespan_matrix[i][j] = makespan_matrix[i][j - 1] + machinejobs[i][permutation[j]
}
}
}
return makespan_matrix[rows - 1][cols - 1];
}
// First step of NEH
// Finding sum of the jobs
for (int i = 0; i < cols; i++) {
int sum = 0;
for (int j = 0; j<rows; j++)
sum += machinejobs[j][i];
job_sum.push_back(sum);
}
//Find the initial sequence by sorting in descending order of the sequence
std::multimap<float, int, std::greater <float>> mm;
for (std::size_t i = 0; i != job_sum.size(); ++i)
mm.insert(make_pair(job_sum[i], i));
std::vector<std::size_t> permutation;
for (const auto & kv : mm) {
45
permutation.push_back(kv.second);
}
for (int i = 0; i < rows; i++) {
for (int j = 0; j <cols; j++) {
initial_matrix[i][j] = machinejobs[i][permutation[j]];
}
}
for (int i = 0; i<cols; i++) {
find_result(i, permutation[i], initial_matrix);
}
//This finds the partial sequence
```

```
//Final_matrix, sequence and C_max are global variable. This function takes the length of part
void find_result(int val, int perm, int**m) {
std::vector<int>temp_sequence;
vector<vector<int>> t(cols, vector<int>(val + 1));
int bestsolution = MAX;
int position = 0;
int currentsolution;
if (val != 0) {
for (int iter = 0; iter <= val; iter++) {
partial_matrix = final_matrix;
for (int i = 0; i < rows; i++)
partial_matrix[i].insert(partial_matrix[i].begin() + iter, m[i][val]);
currentsolution = calculate_val(t, val);
if (bestsolution > currentsolution) {
bestsolution = currentsolution;
position = iter;
}
}

}
for (int i = 0; i < rows; i++) {
final_matrix[i].insert(final_matrix[i].begin() + position, m[i][val]);
}
sequence.insert(sequence.begin() + position, perm);
Cmax = bestsolution;
}
// Initial Sequence for NEH variation 1 i.e sorting the jobs in ascending order
// Finding sum of the jobs
for (int i = 0; i < cols; i++) {
int sum = 0;
for (int j = 0; j<rows; j++)
sum += machinejobs[j][i];
job_sum.push_back(sum);
}
```

```
//Find the initial sequence by sorting in ascending order of the sequence
std::multimap<float, int, std::less <float>> mm;
for (std::size_t i = 0; i != job_sum.size(); ++i)
mm.insert(make_pair(job_sum[i], i));
std::vector<std::size_t> permutation;
for (const auto & kv : mm) {
permutation.push_back(kv.second);
}
//Remaining part i.e finding partial sequence is same for this variation
//Final initial sequence as Variation 2 i.e sorting the initial sequence in descending order o
for (int i = 0; i < cols; i++) {
for (int j = 0; j < rows; j++)
sum += machinejobs[j][i];
avg = (float)sum / rows;
for (int j = 0; j < rows; j++)
variance += pow(machinejobs[j][i] - avg, 2);
variance = variance / rows;
stdDev = sqrt(variance);
job_sum.push_back(stdDev);
}
//Initial permutation or sequence by Sorting in descending order
std::multimap<float, int, std::greater <float>> mm;
for (std::size_t i = 0; i != job_sum.size(); ++i)
mm.insert(make_pair(job_sum[i], i));
std::vector<std::size_t> permutation;
for (const auto & kv : mm) {
permutation.push_back(kv.second);
}
//The variation3 is obtained by sorting jobs on ascending order of standard deviation.
std::multimap<float, int, std::less <float>> mm;
for (std::size_t i = 0; i != job_sum.size(); ++i)
mm.insert(make_pair(job_sum[i], i));
std::vector<std::size_t> permutation;
for (const auto & kv : mm) {
```

```
permutation.push_back(kv.second);
}
//The variation 4 is obtained by finding sum of sum and standard deviation and then sorting in
for (int i = 0; i < cols; i++) {
for (int j = 0; j < rows; j++)
sum += machinejobs[j][i];
avg = (float)sum / rows;
for (int j = 0; j < rows; j++)
variance += pow(machinejobs[j][i] - avg, 2);
variance = variance / rows;
stdDev = sqrt(variance);

add=stdDev+sum
job_sum.push_back(stdDev+sum);
}
//Initial permutation or sequence by Sorting in descending order
std::multimap<float, int, std::greater <float>> mm;
for (std::size_t i = 0; i != job_sum.size(); ++i)
mm.insert(make_pair(job_sum[i], i));
std::vector<std::size_t> permutation;
for (const auto & kv : mm) {
permutation.push_back(kv.second);
}
```

Bibliography

[AA88] T. Aldowaisan and A. Allahverdi. Total flowtime in no-wait flowshops with separated set-up times. *Comput. Oper. Res.*, 25:757765, 1988.

[AA03] T. Aldowaisan and A. Allahverdi. New heuristics for no-wait flowshops to minimize makespan. *Comput. Oper. Res.*, 30:12191231, 2003.

[A.B16] A.Baskar. Revisiting the neh algorithm- the power of job insertion technique for optimizing the makespan in permutation flow shop scheduling. *International Journal of Industrial Engineering Computations*, 7:353–366, 2016.

[ANH96] J. Haddock A. Nagar and S. S. Heragu. A combined branch-and-bound and genetic algorithm based approach for a flowshop-scheduling problem. *Eur. J. Oper. Res.*, 63:397–414, 1996.

[AS96] Ziyong Cai Avraham Shtub, Larry J. LeBlanc. Scheduling programs with repetitive projects: A comparison of a simulated annealing, a genetic and a pair-wise swap algorithm. *European journal of Operational Research*, 88:124–138, 1996.

[Bru04] Dr. Peter Brucker. *Scheduling Algorithm*. Springer-Verlag Berlin Heidelberg, 4 edition, 2004.

[Chi14] Arunasri Chitti. A characterization of open shop scheduling problems using the hall theorem and network flow. 2014.

[CLL93] T.C.E. Cheng C.Y. Lee and B.M.T Lin. Minimizing the makespan in the 3-machine assembly-type flowshop-scheduling problem. *Manag. Sci.*, 39:616625, 1993.

[CPZ95] V.A Strusevich C.N. Potts, S.V. Sevastjanov and L.N.Wassenhove C.M. Zwaneveld. The two-stage assembly scheduling problem: complexity and approximation. *Oper. Res.*, 43:346–355, 1995.

[CSC02] Omer Kircab Chia-Shin Chung, James Flynn. A branch and bound algorithm to minimize the total flow time for m-machine permutation flowshop problems. *International Journal of Production Economics*, 79:185–196, 2002.

[D.A72] D.A.Wismer. Solution of the flowshop sequencing problem with no intermediate queues. *Oper. Res.*, 20:689697, 1972.

[Dan77] DG Dannenbnng. An evaluation of flowshop sequencing heuristic. *Management Science*, 23:1174–1182, 1977.

[E.T93] E.Taillard. Benchmarks for basic scheduling problems. *European journal of Operational Research*, 64:275–285, 1993.

[EV15] Jose M.Framinan Eva Vallada, Ruben Ruiz. New hard benchmark for flowshop scheduling problems minimising makespan. *Computer and Operations Research*, 240:666–677, 2015.

[GR93] R. Gangadharan and C. Rajendran. Heuristic algorithms for scheduling in the no-wait flowshop. *Int. J. Prod. Econ.*, 32:285–290, 1993.

[GR94] R. Gangadharan and C. Rajendran. A simulated annealing heuristic for scheduling in a flowshop with bicriteria. *Comput. Ind. Eng.*, 27:473476, 1994.

[Gui00] A. Guinet. Efficiency of reductions of job-shop to flow-shop problems. *Eur. J. Oper. Res.*, 125:287–326, 2000.

[HCS70] RA Dudek HG Campbell and ML Smith. A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, 16:630–637, 1970.

[HS96] N.G. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Oper. Res*, 44:510–525, 1996.

[HS05] S.Reza Hejazi and S.Saghafian. Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research*, 43:2895–2929, 2005.

[H.S11] M.J.Moayed H.S.Mirsanei, M.Zandieh. A simulated annealing algorithm approach to hybrid flow shop scheduling with sequence-dependent setup times. *Journal of Intelligent Manufacturing*, 22:965–978, 2011.

[IAS96] N.G. Hall I.N.K. Abadi and C. Sriskandarajah. Minimizing cycle time in a blocking flowshop. *Oper. Res*, 48:177–180, 1996.

[IOR89] D.S. Johnson IH OSMAN and R.Sethi. Simulated annealing for permutation flow-shop scheduling. *Omega*, 17:551–557, 1989.

[JF03] C.Rajendran J.M. Framinan, R.Leisten. Different initial sequences for the heuristic of nawaz, enscore and ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *The International Journal of Production Research*, 41:121–48, 2003.

[JMF04] R Leisten J M Framinan, J N D Gupta. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55:1243–1255, 2004.

[Joh54] S.M. Johnson. Opimal two-and three-stage production schedules with setup times included. *Naval research Logistics Quarterly*, 1:61–68, 1954.

[JXG14] T.C.E. Cheng Chin-Chia Wu Jianyou Xu, Yunqiang Yin and Shusheng Gu. An improved memetic algorithm based on a dynamic neighbourhood for the permutation flowshop scheduling problem. *International Journal of Production Research*, 52:1188–1199, 2014.

[KK04] C. Koulamas and G.J Kyparisis. Concurrent flowshop-scheduling to minimize makespan. *Eur. J. Oper. Res.*, 156:524529, 2004.

[Kod12] Swapna Kodimala. A branch and bound method for sum of completion permutation flow shop. 2012.

[LSW11] Gengcheng Liu, Shiji Song, and Cheng Wu. Two techniques to improve the neh algorithm for flow-shop scheduling problems. *International Journal of Industrial Engineering Computations*, 6839:41–48, 2011.

[MGR76] D.S. Johnson M.R. Garey and R.Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operational Research*, 1:117–129, 1976.

[MN83] Inyong Ham Muhammad Nawaz, E Emory Enscore Jr. A heuristic algorithm for the m machine n-job flow-shop sequencing problem. *Omega, The International Journal of Management Science*, 11:91–95, 1983.

[NM53] Marshall N. Rosenbluth Augusta H. Teller Nicholas Metropolis, Arianna W. Rosenbluth. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21:1087–1091, 1953.

[Pal65] DS Palmer. Sequencing job through a multistage process in the minimum total time-a quick ,ethod of obtaining near optimum. *Opt Res Q*, 16:101–107, 1965.

[PCF91] G. Anandalingam Peter C. Fetterolf. Optimizing interconnection of local area networks: A simulated annealing approach. *ORSA Journal on Computing*, 3:275–287, 1991.

[PK08] J. Kamburowski P.J. Kalczynski. An improved neh heuristic to minimize makespan in permutation flow shops. *Computers and Operational Research*, 35:3001–3008, 2008.

[RK79] J.K.Lenstra R.L.Graham, E.L.Lawler and A.H.G.Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[Roc84] Hans Rock. The three-machine no-wait flowshop is np-complete. *J. Assoc. Comput. Mach.*, 31:336345, 1984.

[RR05] C. Maroto R. Ruiz. A comprehensive review and evaluation of permutation flow shop heuristics. *European Journal of Operational research*, 165:479–494, 2005.

[RWCM67] William L. Maxwell Richard W. Conway and Louis W. Miller. *Theory of scheduling.* Addison-Wesley, 1967.

[SK83] M. Vecchi S. Kirkpatrick, C. Gelatt. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[ST90] E.F Stafford and F.T. Tseng. On the srikarghosh milp model for the n x m sdst flowshop problem. *Int. J. Prod. Res.*, 28:1817–1830, 1990.

[TAA99] J.N.D. Gupta T. Aldowaisan and A. Allahverdi. A review of scheduling research involving set-up considerations. *Omega*, 27:219–239, 1999.

[VNG96] C.L. Chen V.R. Neppalli and J.N.D. Gupta. Genetic algorithms for the two-stage bicriteria flowshop problem. *Eur. J. Oper. Res.*, 95:356373, 1996.

[XD08] Ping Chen Xingye Dong, Houkuan Huang. An improved neh-based heuristic for the permutation flowshop problem. *Computer and Operations Research*, 35:3962–3968, 2008.

[Yel11] Sadhana Yellanki. Simulated annealing approach to flow shop scheduling. 2011.

Curriculum Vitae

Graduate College
University of Nevada, Las Vegas

Pooja Bhatt
bhatt.pooza@gmail.com

Degrees:

Bachelor of Science in Computer Science and Information Technology
Tribhuwan University, Nepal

Thesis Title: Permutation Flow Shop via Simulated Annealing and NEH

Thesis Examination Committee:

Chairperson, Wolfgang Bein, Ph.D.
Committee Member, Ajoy K. Datta, Ph.D.
Committee Member, Laxmi Gewali, Ph.D.
Graduate Faculty Representative, Henry Selvaraj, Ph.D.