A NEW APPROACH TO GENERATING NON-PERMUTATION SCHEDULES FOR FLOWSHOPS WITH MISSING OPERATIONS

METİN TABALU

DECEMBER 2006

A NEW APPROACH TO GENERATING NON-PERMUTATION SCHEDULES FOR FLOWSHOPS WITH MISSING OPERATIONS

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

BY

METİN TABALU

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

DECEMBER 2006

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Çağlar Güven
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Ömer Kırca
Supervisor

Examining Committee Members

Prof. Dr. Meral Azizoğlu (METU IE)

Prof. Dr. Ömer Kırca (METU IE)

Prof. Dr. İhsan Sabuncuoğlu (BILKENT IE)

Assist. Prof. Dr. Sedef Meral (METU IE)

Assist. Prof. Dr. Esra Karasakal (METU IE)

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name:

Signature:

ABSTRACT

A NEW APPROACH TO GENERATING NON-PERMUTATION SCHEDULES FOR FLOWSHOPS WITH MISSING OPERATIONS

TABALU, Metin

M. Sc. Thesis, Department of Industrial Engineering

Supervisor: Prof. Dr. Ömer KIRCA

December 2006, 85 pages

In this study, non-permutation flowshops with missing operations are considered. The primary performance criterion is the total cycle time (i.e. makespan) and secondary criterion is the total flowtime. In order to obtain the schedule with the minimum makespan and minimum total flowtime, non-permutation schedules are being generated instead of permutation ones by permitting multiple jobs bypassing stages where missing operations occur. A heuristic algorithm has been developed in order to generate non-permutation sequences through those stages. The heuristic algorithm has been compared with the existing heuristic methods in the literature, the ones generating permutation vs. the ones generating non-permutation schedules. Computational analysis is conducted to investigate the effects of certain parameter values such as the number of machines, the number of jobs and the percentage of missing operations. The results demonstrate slight improvement in the makespan as well as the significant improvement in total flowtime of schedules generated by the new heuristic procedure compared to leading non-permutation and permutation schedule generating heuristics, where the percentage of improvement gets higher with larger percentages of missing operations.

Keywords: Non-permutation schedules, flowshops with missing operations.

ÖZ

AKIŞ TİPİ ve EKSİK OPERASYONLAR İÇEREN ÇİZELGELEME PROBLEMLERİNDE PERMÜTASYON TİPİ OLMAYAN İŞ SIRALARININ OLUŞTURULMASINDA YENİ BİR YÖNTEM

TABALU, Metin
Yüksek Lisans Tezi, Endüstri Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Ömer KIRCA

Aralık 2006, 85 sayfa

Bu çalışmada eksik operasyonlar içeren akış tipi çizelgeleme problemleri incelenmiştir. Temel performans kriteri toplam çevrim zamanı olup, ikincil performans kriteri toplam akış zamanıdır. Permütasyon tipi iş sıraları yerine permütasyon tipi olmayan iş sıraları oluşturulmaya çalışılmış, ve bu esnada eksik operasyonlar içeren aşamalarda işlerin grup halinde öne geçişlerine izin verilmiştir. Aşamalar arası permütasyon tipi olmayan yeni iş sıraları oluşturan sezgisel bir de algoritma geliştirilmiştir. Yeni geliştirilen sezgisel algoritma literatürdeki permütasyon tipi ve permütasyon tipi olmayan iş sıraları oluşturan belli başlı sezgisel algoritmalar ile karşılaştırılmıştır. Toplam makine sayısı, toplam iş sayısı ve eksik operasyonların toplam operasyon sayısına oranı gibi parametrelere değişik değerler verilerek çeşitli sayısal değerlendirmeler yapılmıştır. Sonuçlar, yeni yöntemin oluşturulan permütasyon tipi olmayan çizelgelemeler ile toplam çevrim zamanını kısıtlı ölçüde, toplam akış zamanınını da önemli ölçüde azalttığını ve bu iyileştirmenin eksik operasyonların yüzdesine paralel olarak arttığını göstermektedir.

Anahtar kelimeler: Permütasyon tipi olmayan çizelgelemeler, eksik operasyonlu akış tipi çizelgeleme problemleri.

To my family & my love

ACKNOWLEDGEMENTS

I would like to express my gratitude to Prof. Ömer Kırca for his patient supervision and continual understanding through the course of this study. This study could not have been completed without his support.

I also would like to express my deepest thanks to my family for their endless patience and understanding. The completion of this task would not have been possible without their endless love and faith in me.

I want to express my thanks also to Dr. Sedef Meral for her support during the preparation phase of the thesis and moreover during various instances of my study in Middle East Technical University Industrial Engineering Master of Science program. Special thanks go to my friends Halecan Tuncay, Fatma Kılınç and Alper Karaduman for her reinforcement, sharing of her ideas with me while making literature research, and his technical help to complete the thesis work, respectively. I also acknowledge the continuous support of my other friends Aykut Mehmet Tutucu and Vehbi Özer for their encouragement during the establishment of various parts of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A Flowline-Based Manufacturing System (FBMS) is a manufacturing environment where machines are arranged in the order in which jobs continue their operations, and have a unidirectional flow pattern. In a real manufacturing environment we often have the FBMS, where 'cells' are formed to manufacture 'part families' such that each family is, as much as possible, manufactured within the same cell.

The urge for manufacturing part families in the same cell is mainly because of the need for minimizing material handling by minimizing inter-cell movements. The creation of manufacturing cells also enabled the manufacturers to form multi-functional teams within each of those cells, where employees/units form more than one set of tasks on various part families coming to the cell. By doing so, some leading manufacturers have managed to increase the utilization of each of their employees and improved the capacity associated with manufacturing cells in order to reduce costs. In a cellular manufacturing system, all parts in a part-family need not be processed on all machines in a cell, thus a part may have missing operations on some machines.

The configuration of a manufacturing cell may be either a flowline layout or a job-shop layout. However, the formation of a flowline layout has definite advantages over the job-shop layout in the sense that; material flow is simplified with avoidance of back-tracking, less material handling activities performed, and better control of production activities are enabled (Dumolien and Santen, 1983). Especially in the electronics, automotive, chemicals and pharmaceuticals industry, leading manufacturers have developed FBMS both in parts manufacturing and assembly operations. Within those systems, part families are being processed in various sizes

of batches, requiring operations at different manufacturing cells. The important aspect of those is that all of the jobs flowing through machines more or less follow the same unidirectional flow with some existing missing operations for each distinct job family. Having this fact on their minds, researches have tried to find ways of modeling flowshops, given the economic importance of FBMS.

The problem is then in order to utilize a FBMS in the most efficient way; one has to generate good solutions by modeling the system as a multi-product batch scheduling problem. The contents of this study includes a presentation of a study of the nature of a serial multi-product batch scheduling problem and tries to provide insights and justifications for considering flowshop scheduling problems (FSP) under a non-permutation schedule (NPS) instead of permutation schedules, where jobs flow through machine with the same sequence at each stage leading to forced idleness of machines having missing operations for some jobs within the sequence.

Throughout the following sections of this thesis, Chapter 2 first gives a brief explanation of flowshop by laying down the foundations of it together with a brief description of permutation schedules and their limits in terms of solutions to meet performance criterion. The mixed integer programming model used for permutation flowshops is also given in this chapter. Chapter 3 starts with a brief description of the Johnson's algorithm together with the 2-machine flowshop problem in order to initiate the discussion for the heuristics developed for flowshops. After that, the second part of Chapter 3 involves a brief survey of flowshop heuristics –mostly generating permutation schedules– by paying special attention to improvement heuristics, as their notion will be employed at later stages in order to develop non-permutation schedules for flowshop. The second part of Chapter 3 reveals the need for obtaining non-permutation schedules for the flowshop problem. Thinking in terms of the makespan as the primary performance criterion, a simple example shows the possibility of drastically decreasing the makespan of a schedule by employing non-permutation schedules.

Chapter 4, demonstrating the new heuristic approach developed using an existing heuristic, first draws out the benefits of enabling job-passing through stages in a permutation flowshop. Starting from an initial permutation flowshop, the use of job-passing enables the problem-solver to decrease both the makespan and the total flowtime by making small interchanges in the permutation schedule and forming partially permutation sequences through following stages of a flowshop. In terms of the new approach, while obtaining partially permutation sequences at each stage, jobs with missing operations are passed ahead with small distortions of the permutation sequence for the current stage. In that sense job-passing as a tool for NPS generation is emphasized. Each job-passing is made by simultaneously considering all jobs as candidates to move ahead, and re-sorting the group of candidates within themselves in order to obtain 'robust' schedules for later stages. Simple dispatching rules have been brought into the discussion and further have been employed in order to make a reliable comparison of the new method by the other permutations.

Chapter 5 summarizes the outcomes of the computational experimentations made in order to highlight the performance of the new heuristic compared to the existing methods for deriving permutation schedules and the NPS. Results of extensive computational experimentation, with makespan as the primary criterion and total flowtime as the secondary criterion are presented. Results are tabularized in order to see the effectiveness of the new approach in terms of producing NPS providing much less total flowtime with decreased makespan value as well.

Chapter 6 sums up the purpose and contents of this study about developing a new heuristic approach for generating non-permutation schedules for flowline-based manufacturing systems with missing operations, which substantially decreases the total flowtime, as well as significant improvement in makespan. The computational results included in the previous chapter are analyzed and conclusions have been made based on the analysis of those results. Some key remarks for future research also have been proposed.

CHAPTER 2

PROBLEM DEFINITION

2.1. FLOWSHOP SCHEDULING

A flowshop is a basic FBMS in which the machines (i.e. stages) are arranged in a series order. In such a shop, starting from an initial machine, jobs flow through several intermediary machines and ultimately get their operations done at a final completion machine. Traditionally, such designs are referred to as '*flowshop*', even though an actual shop may comprise much more than a single configuration. Through a flowshop, the work in each job is broken down into separate tasks called operations and each operation is performed at a different machine. In particular, each operation after the first has exactly one direct predecessor and each operation before the last has exactly one direct successor, as shown below in Figure 2.1. Therefore, each job requires a specific sequence of operations to be carried out for the job to be complete. This type of structure is sometimes called a linear precedence structure (Baker 1995).

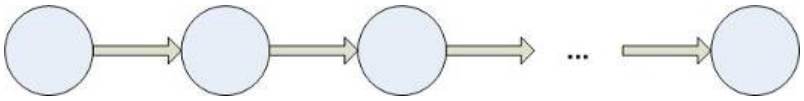

Figure 2.1: The precedence structure of a job in a flowshop

The shop contains m different machines, and in the "pure" flowshop model each job consists of m operations, each of which requiring a different machine. The machines in a flowshop can thus be numbered $1, 2, ..., m$; and the operations of job j numbered $(1, j), (2, j),$ ... $, (m, j)$, so that they correspond to the machine required. For example, p_{53} denotes the operation time on machine 5 for job 3. Figure 2.2 represents the flow of work in "pure" flowshop, in which all operations require one

operation on each machine.

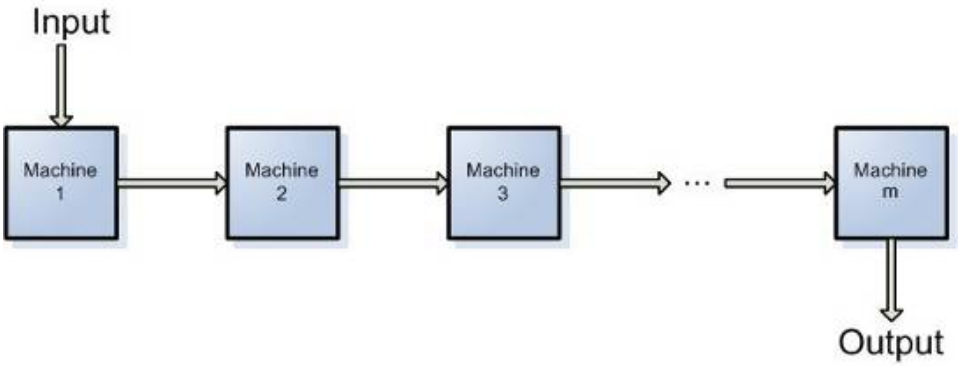

Figure 2.2: Workflow in a pure flowshop

Figure 2.3 represents the flow of work in a more general flowshop, which will be the subject matter of this thesis study. In the general case, jobs may require fewer than m operations, their operations may not always require adjacent machines in the numbered order i.e. there might be some missing operations, and the initial and final operations may not always occur at machines 1 and m. Nevertheless, the flow of work is still unidirectional, and the general case can be represented as a pure flowshop in which some of the operations times are zero.

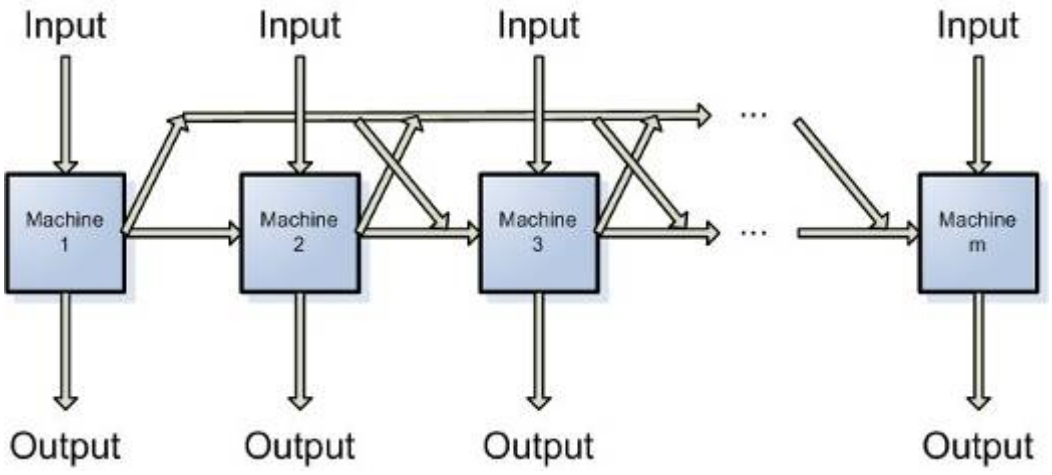

Figure 2.3: Workflow in a general flowshop

With machines in series, the five main properties of the flowshop model are similar to those of the basic single-machine model.

Property 1: A set of n independent, multiple-operation jobs is available for

processing at time zero. (Each job requires m operations, and each operation requires a different machine.)

Property 2: Setup times for the operations are sequence-independent and included in the processing times.

Property 3: Job descriptors are known in advance.

Property 4: All machines are continuously available.

Property 5: Once an operation begins, it proceeds without interruption.

One difference from the basic single-machine case is that inserted idle time may be advantageous. In particular, also for the case of missing operations the unidirectional sequences with partial changes through different stages provides various modeling advantages for flowshops. In the single-machine model with simultaneous arrivals the assumption that the "*machine need never be kept idle when work is waiting*" can be made. In the flowshop case, however, inserted idle time may be needed to achieve theoretical optimality. For example, consider the following two-job four-machine problem.

Table 2.1: Processing times for two-job four-machine problem

Job j	**1**	**2**
p_{1j}	1	4
p_{2j}	4	1
p_{3j}	4	1
p_{4j}	1	4

Suppose that F (total flow time: $F = \sum_{j=1}^{n} F_j$) is the measure of performance. The two schedules shown in Figures 2.4a and 2.4b are the only schedules with no inserted idle time, and in either schedule F = 24. The schedule in Figure 2.4c instead is the optimal schedule, with F = 23. Note that in this third schedule, machine 3 is kept idle at time t = 5, when operations (3, 1) could be started, in order to await the completion of operation (2, 2).

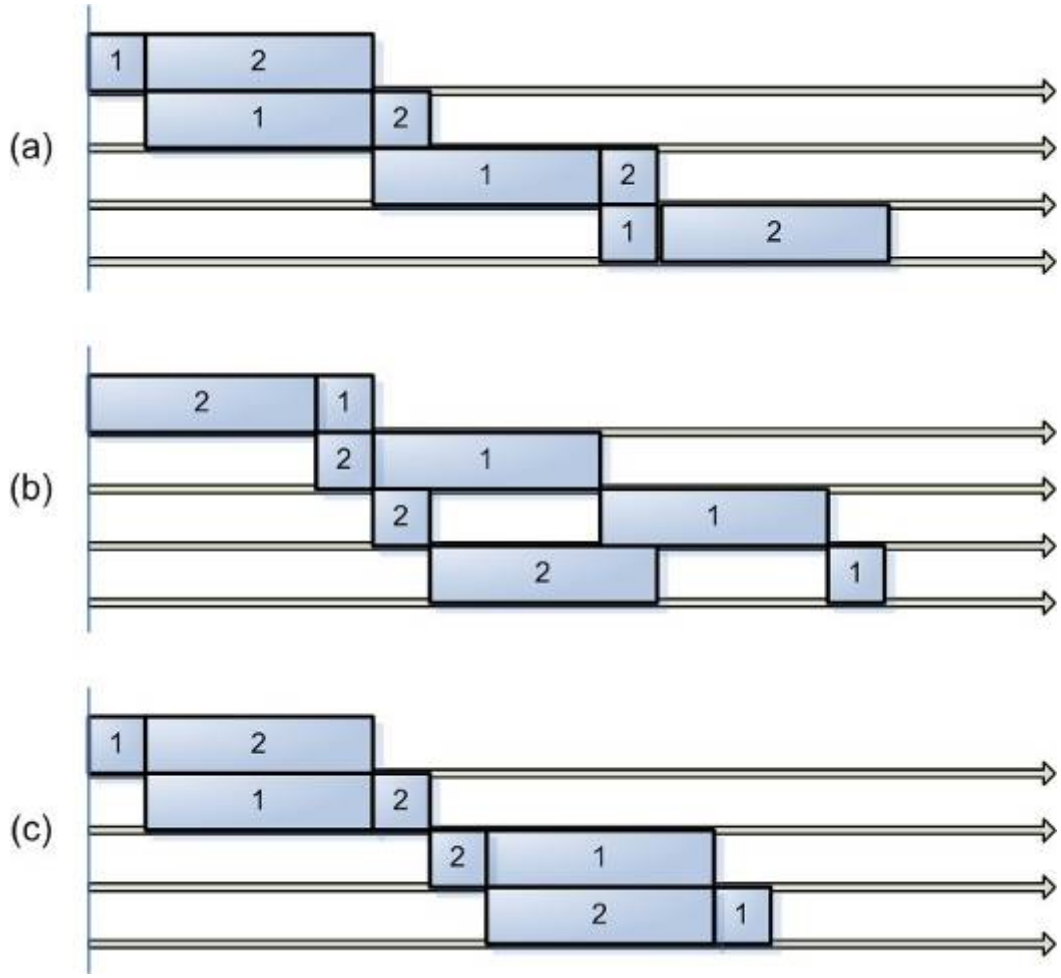

Figure 2.4: Three schedules for the example problem

For the single-machine model, it is trivial that there is a one-to-one relation between a job sequence and a permutation of the numbers $1, 2, \ldots, n$. To find an optimum sequence, it was necessary to examine (at least implicitly) each of the sequences to the $n!$ different permutations. In the flowshop problem, there are $n!$ different job sequences possible for each machine, and as many as $(n!)^m$ different schedules can be generated for the whole flowshop. While searching for the optimum, it would obviously be helpful if many of these possibilities could be ignored. Through the next section, the extent to which the search for an optimum can be reduced will be discussed under the name of permutation flowshops. Then, the case $m = 2$ will be discussed in order to show the interesting points in its own right and to obtain a building block for solving larger problems. The problem is generally formulated as

integer or mixed integer programming problems with makespan as the objective (Baker, 1995). However, throughout this thesis study, the minimization of total (weighted) flowtime is also considered as a secondary objective of scheduling, as this objective is more important and relevant than the objective of minimizing the makespan in real life situations (Pinedo, 2002). Models with due-date-related objectives are few and are out of the scope of this study.

2.2. PERMUTATION SCHEDULES

The example given in the previous section illustrates that it may not be sufficient to consider only schedules in which the same job sequence occurs on each machine. On the other hand, it is not always necessary to consider $(n!)^m$ schedules in determining an optimum. The two dominance properties given below indicate how much of a reduction is possible in flowshop problems.

Theorem 1: With respect to any regular measure of performance in the flowshop model, it is sufficient to consider only schedules in which the same job sequence occurs on the first two machines (Baker, 1995).

Consider a schedule in which the sequences on machines 1 and 2 are different. Somewhere in such a schedule a pair of jobs, i and j can be found such that operation $(1, i)$ preceding an adjacent operation $(1, j)$ but operation $(2, j)$ preceding $(2, i)$, as in Figure 2.5(a). For this pair, the order of the jobs on machine 2 can be imposed to machine 1 (j before i), without adversely affecting the performance measure (Baker 1995). If we interchange operations $(1, i)$ and $(1, j)$, resulting in the schedule shown in Figure 2.5(b), then

- with the exception of $(1, i)$, no operation is delayed,
- operation $(2, i)$ is not delayed, and
- earlier processing of $(2, j)$ and other operations as well, may result.

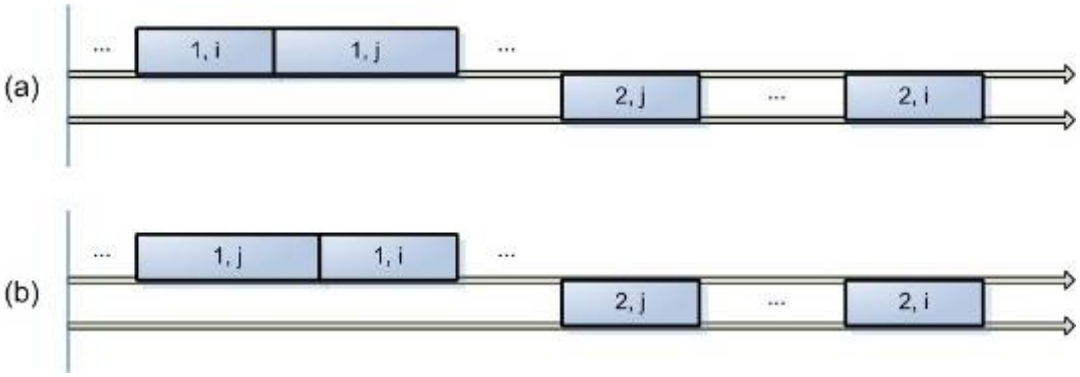

Figure 2.5: A pair-wise interchange of two operations on machine 1

Therefore, the interchange would not increase the completion time of any operation on machine 2 or on any subsequent machine. This means that no increase in any job completion time could result from the interchange, and hence no increase will occur in any regular measure of performance. Since the same argument applies to any schedule in which job sequences differ on machines 1 and 2, the property must hold.

Theorem 2: With respect to the makespan of the flowshop model, it is sufficient to consider only schedules in which the same job sequence occurs on the last two machines (Baker, 1995).

Consider a schedule in which the sequences on machines $(m-1)$ and m are different. Somewhere in such a schedule a pair of jobs, i and j can be found such that, operation (m, j) preceding an adjacent operation (m,i), but operation $(m-1,i)$ preceding $(m-1, j)$. As a result of interchanging operations (m,i) and (m, j),

- with the exception of (m, j), no operation is delayed,
- operation (m, j) completes no later than (m,i) in the original schedule,
- earlier processing of operations (m,i) and (m, j) may result.

Therefore, the interchange would not lead to an increase in the makespan of the schedule. Again, this type of argument applies to any schedule in which job sequences differ on machines $(m-1)$ and m. The implication of these two theorems is that in searching for an optimal schedule, it is necessary to consider different job sequences on different machines with these two general exceptions.

1. For the case of obtaining any regular performance measure, it is sufficient to keep the same job order to occur on the first two machines, so that $(n!)^{m-1}$ schedules constitute a *dominant set*.
2. For the problems with the makespan criterion, it is also sufficient for the same job order to occur on the last two machines, so that $(n!)^{m-2}$ schedules constitute a dominant set for $m > 2$.

The outcomes of these two theorems will lead to the basic problem statement of this thesis. Based on any regular performance measure, making partial pair wise interchanges in the job sequence of a permutation schedule (i.e. making it a non-permutation schedule) schedule may lead to significant improvements in terms of that specific performance measure. For larger problems, a new heuristic approach has to be developed in order to improve the "makespan" together with the "total flow time" of the newly formed NPS (i.e. non-permutation schedule) based on an initial PS on hand.

2.3. MATHEMATICAL FORMULATION

The mathematical formulation developed for the permutation flowshop scheduling problem by Gupta and Tseng (2004) is as follows. Before the listed equations of the mathematical model, the notations used for the permutation flowshop MIP (mixed-integer programming) model are as follows. The subscript symbols are for r for machines, for $(1 \le r \le M)$; i and k for jobs $(1 \le i, k \le N)$ where the parameters M and N represent the number of machines and jobs, respectively. $T = \{T_{rj}\}$ is the $M \times N$ matrix of job processing times, with $T_{ri} =$ processing time of job i on machine r. The variables of the MIP model are then defined as follows:

B_{rj} start (begin) time of job in sequence position j on machine r

C_{ri} completion time of job i on machine r

D_{ik} 1, if job i is scheduled any time before job k; 0 otherwise; $i < k$

E_{rj} completion (end) time of the job in position j on machine r

S_{ri} start time of job i on machine r

X_{rj} idle time on machine r before the start of job in sequence position j

Y_{rj} idle time of job in sequence position j after it finishes processing on machine r

Z_{ij} 1, if job i is assigned to sequence position j, 0 otherwise.

$C_{\max}$ maximum flowtime (makespan) of the schedule determined by the completion time of the job in the last sequence position on the last machine.

D_{ik} and Z_{ij} are binary integer variables. The others are real variables that take integer values when processing times are also given as integer values.

Minimize (Makespan) $= C_{\max} = C_{MN}$

subject to

$$\sum_{j=1}^{N} Z_{ij} = 1; \qquad 1 \le i \le N, \tag{2.1}$$

$$\sum_{i=1}^{N} Z_{ij} = 1; \qquad 1 \le j \le N \tag{2.2}$$

$$\sum_{i=1}^{N} T_{ri} Z_{i,j+1} - \sum_{i=1}^{N} T_{r+1,i} Z_{ij} + X_{r,j+1} - X_{r+1,j+1} + Y_{r,j+1} - Y_{rj} = 0;$$

$$(1 \le r \le M-1; 1 \le j \le N-1), \tag{2.3}$$

$$\sum_{i=1}^{N} T_{ri} Z_{i1} + X_{r1} - X_{r+1,1} + Y_{r1} = 0;$$

$$(1 \le r \le M-1), \tag{2.4}$$

$$C_{MN} = \sum_{i=1}^{N} T_{Mi} + \sum_{p=1}^{N} X_{Mp} \tag{2.5}$$

For this MIP model, the first two constraints (i.e. the equation groups (2.1) and (2.2)) ensure that each job is assigned to just one sequence position and each sequence

position is filled with one and only one job, respectively. Constraints (2.3) and (2.4) ensure that;

a) the job in sequence position j cannot begin processing on machine $r+1$, until it has completed its processing on machine r;
b) the job in sequence position $j+1$ cannot begin its processing on machine r until the job in sequence position j has completed its processing on that machine.

And the final constraint, namely equation (2.5) measures makespan of the set of jobs. There are various derivations and improvements on this basic MIP model developed by Wagner (1959). This basic notation can be extended in order to make it applicable also for the cases where non-permutation schedules are needed. This simple shift can be made by adding another index s where $(1 \le s \le M)$ indicating the sequence of jobs (which is differing in this case) through each machine. Therefore, with the addition of the new index, the new variables of the model in an updated form are as follows:

B_{rsj} start (begin) time of job in sequence position j in sequence s on machine r

C_{ri} completion time of job i on machine r

D_{iks} 1, if job i is scheduled before job k in sequence s; 0 otherwise; $i < k$

E_{rjs} completion (end) time of the job in position j in sequence s on machine r

S_{ri} start time of job i on machine r

X_{rjs} idle time on machine r before the start of job in sequence position j, in sequence s

Y_{rjs} idle time of job in sequence position j, in sequence s after it finishes processing on machine r

Z_{ijs} 1, if job i is assigned to sequence position j, in sequence s; 0 otherwise.

$C_{\max}$ maximum flowtime (makespan) of the schedule determined by the completion time of the job in the last sequence position on the last machine.

Based on the new variables, the MIP model formulation (with the same objective function minimizing $= C_{\max} = C_{MN}$) giving non-permutation schedules is as follows:

subject to

$$\sum_{j=1}^{N} Z_{isj} = 1; \qquad 1 \le i \le N, \ (1 \le s \le M) \tag{2.6}$$

$$\sum_{i=1}^{N} Z_{ijs} = 1; \qquad 1 \le j \le N, \ (1 \le s \le M) \tag{2.7}$$

$$\sum_{i=1}^{N} T_{ri} Z_{i,j+1,s} - \sum_{i=1}^{N} T_{r+1,i} Z_{ijs} + X_{r,j+1,s} - X_{r+1,j+1,s} + Y_{r,j+1,s} - Y_{rjs} = 0;$$

$$(1 \le r \le M-1; 1 \le j \le N-1), \ (1 \le s \le M) \tag{2.8}$$

$$\sum_{i=1}^{N} T_{ri} Z_{i1s} + X_{r1s} - X_{r+1,1,s} + Y_{r1s} = 0;$$

$$(1 \le r \le M-1), \ (1 \le s \le M) \tag{2.9}$$

$$C_{MN} = \sum_{i=1}^{N} T_{Mi} + \sum_{p=1}^{N} X_{Mp} \tag{2.10}$$

This second model will ensure that Z_{ij} values are not necessarily the same for each distinct value of s implying the generation of non-permutation sequences. However, two more constraints (i.e. 2.11 & 2.12) have to be added to ensure that job i either precedes job k or follows job k in the sequence s but not both. By taking P as a very large number:

$$C_{ri} - C_{ik} - P \times D_{ik} \ge T_{ri};$$

$$(1 \le r \le M; 1 \le i <\le N) \tag{2.11}$$

$$C_{ri} - C_{ik} - P \times D_{ik} \ge P - T_{rk};$$

$$(1 \le r \le M; 1 \le i <\le N) \tag{2.12}$$

CHAPTER 3

LITERATURE REVIEW & MOTIVATION FOR THIS STUDY

3.1. HEURISTICS FOR THE FLOWSHOP SCHEDULING PROBLEM

This section focuses on the different types of heuristics existing in the literature in order to create schedules with better performance in terms of makespan as the essential performance criterion. The complexity of the flowshop scheduling problem renders exact solution methods impractical for instances of more than a reasonable jobs and/or machines. Some of these heuristics are going to be employed in the core discussions of this study. The heuristics can be separated as either constructive heuristics or improvement heuristics, the former are heuristics that build a feasible schedule from scratch and the latter are heuristics that try to improve a previously generated schedule by normally applying some form of specific problem knowledge.

3.1.1. CONSTRUCTIVE HEURISTICS

Johnson's algorithm (1954) is the earliest known heuristic for the PFSP, which provides an optimal solution for two machines. Moreover, it can be used as a heuristic for the m machine case by clustering the m machines into two "virtual" machines. The computational complexity of this heuristic is $O(n \log n)$. Other authors have used the general ideas of Johnson's rule in their algorithms, for example, Dudek and Teuton (1964) developed an $m - stage$ rule for the permutation flowshop scheduling problem (PFSP) that minimizes the idle time accumulated on the last machine while processing each job by using Johnson's approach.

Campbell et al. (1970) developed a heuristic algorithm which is basically an extension of Johnson's algorithm. In this case, several schedules are constructed and the best one is given as result. The heuristic is known as CDS and builds $m-1$ schedules by clustering the m original machines into two virtual machines and

solving the generated two machine problem by repeatedly applying the Johnson's rule. The CDS heuristic has a computational complexity of $O(m^2 n + mn \log n)$. In a more recent work, Koulamas (1998) reported a new two phase heuristic, called HFC. In the first phase, the HFC heuristic makes extensive use of Johnson's algorithm. The second phase improves the resulting schedule from the first phase by allowing job passing between machines, i.e. by allowing non-permutation schedules. This is a very interesting idea, since it is known that permutation schedules are only dominant for the three-machine case. In the general m machine case, a permutation schedule is not necessarily optimal anymore (Potts et al., 1991). The significance of this heuristic relies on the fact that it departs from the PFSP problem by allowing job passing. Therefore, it has been included in the discussion for comparison reasons since as for some instances job passing will be quite beneficial. The benefits of permitting job passing are furthermore extensively employed throughout the later stages of this thesis study. Taking into account both phases, the general computational complexity of this heuristic is roughly $O(m^2 n^2)$.

Another approach is to assign a weight or "index" to every job and then arrange the sequence by sorting the jobs according to the assigned index. This idea was first exploited by Palmer (1965) when he developed a very simple heuristic in which for every job a "slope index" is calculated and then the jobs are scheduled by non-increasing order of this index, which leads to a computational complexity of $O(nm + n \log n)$.

3.1.1.1. NAWAZ ET AL.'S (NEH) HEURISTIC

Nawaz et al.'s (1983) NEH heuristic is regarded as the best heuristic for the PFSP (Taillard, 1990). It is based on the idea that jobs with high processing times on all the machines should be scheduled as early in the sequence as possible. The procedure is straightforward:

i. The total processing times for the jobs are calculated using the formulae:

$\forall$ job i, $i = 1,.....,n, P_i = \sum_{j=1}^{m} p_{ij}$.

ii. The jobs are sorted in non-increasing order of P_i. Then the first two jobs (those two with higher P_i) are taken and the two possible schedules containing them are evaluated.

iii. Take job i, $i = 3,.....,n$ and find the best schedule by placing it in all the possible i positions in the sequence of jobs that are already scheduled. For example, if $i = 4$ the already constructed sequence would contain the first three jobs of the sorted list calculated in step 2, then the fourth job could be placed either in the first, in the second, in the third or in the last position of the sequence. The best sequence of the four would be selected for the next iteration.

Recalling the previous paragraphs, it is obvious that the NEH heuristic is based neither on Johnson's algorithm nor on slope indexes. The only drawback is that a total of $[n(n+1)/2]-1$ schedules have to be evaluated, being n of those schedules complete sequences. This makes the complexity of NEH rise to $O(n^3m)$ which can be lengthy for big problem instances.

However, Taillard (1990) reduced NEH's complexity to $O(n^2m)$ by calculating all the partial schedules in a given iteration in a single step. Sarin and Lefoka (1993) exploited the idea of minimizing idle time on the last machine since any increase in the idle time on the last machine will translate into an increase in the total completion time or makespan. In this way, the sequence is completed by inserting one job at a time and priority is given to the job that, once added to the sequence, would result in minimal added idle time on machine m. The method proposed compares well with the NEH heuristic but only when the number of machines in a problem exceeds the number of jobs.

Pour (2001) proposed another insertion method. This new heuristic is based on the idea of job exchanging and is similar to the NEH method. The performance of this method is evaluated against the NEH, CDS and Palmer's heuristics showing better

effectiveness only when a big number of machines is considered, and being the computational complexity $O(n^3m)$. More recently, Framinan et al. (2003) have published a study about the NEH heuristic where different initializations and orderings are considered. The study also includes different objective functions including makespan, idle-time and flowtime. Framinan et al. (2003) have proven that NEH heuristic as an insertion method outperforms most of the other heuristics based on all of the performance criteria mentioned.

Other authors have proposed heuristics that use one or more of the previous ideas, for example, Gupta (1972) proposed three heuristic methods, named minimum idle time (MINIT), minimum completion time (MICOT) and MINIMAX algorithms, the first two are based on job pair exchanges and the MINIMAX is based on Johnson's rule. These three algorithms were tested with the objectives of $C_{\max}$ and mean flowtime $(\overline{F})$ and compared with the CDS algorithm, proving to be superior only when considering the F objective. Additionally, there are many other methods developed, which are neither based on Johnson's nor Palmer's ideas and not constructing sequences by job exchanges and/or insertions only. For example, King and Spachis (1980) evaluated various heuristics for the PFSP and for the flowshop with no job waiting (no-wait flowshop). For the PFSP, a total of five heuristics based on dispatching rules were developed. A different approach is shown in Stinson and Smith (1982) where the authors solve the permutation flowshop problem by using a well known heuristic for the Traveling Salesman Problem (TSP) as indicated by Ruiz and Maroto (2005).

3.1.2. IMPROVEMENT HEURISTICS

Contrary to constructive heuristics, improvement heuristics start from an already built schedule and try to improve it by some given procedure. Dannenbring (1977) proposed two simple improvement heuristics; these are Rapid Access with Close Order Search (RACS) and Rapid Access with Extensive Search (RAES). The reason behind these two heuristics is that Dannenbring found that simply swapping two adjacent jobs in a sequence obtained by the RA heuristic resulted in an optimal

schedule. RACS works by swapping every adjacent pair of jobs in a sequence (this is $n-1$ steps). The best schedule among the $n-1$ generated is then given as a result. In RAES heuristic, RACS is repeatedly applied while improvements are found. Both RACS and RAES heuristics start from a schedule generated with the RA constructive heuristic.

Ho and Chang (1991) developed a method that works with the idea of minimizing the elapsed times between the end of the processing of a job in a machine and the beginning of the processing of the same job in the following machine in the sequence. The authors refer to this time as ‘‘gap’’. The algorithm calculates the gaps for every possible pair of jobs and machines and then by a series of calculations, the heuristic swaps jobs depending on the value of the gaps associated with them. The heuristic starts from the CDS heuristic by Campbell et al (1970).

Ho (1995) developed a heuristic composed of several iterations of an improvement scheme based on finding a local optimum by adjacent pairwise interchange of jobs, and improving the solution by insertion (or shift) movements. This heuristic performs significantly better than the others, although its main disadvantage is that it employs much higher CPU time. In fact, this heuristic seems closer to local search techniques like simulated annealing or taboo search and it probably has to be discarded for large problem sizes and/or in those environments where sequencing decisions are required in very short time intervals.

Suliman (2000) developed an improvement heuristic, which in the first phase, generates a schedule with the CDS heuristic method. In the second phase, the schedule generated is improved with a job pair exchange mechanism. In order to reduce the computational burden of an exhaustive pair exchange mechanism, a directionality constraint is imposed to reduce the search space. For example, if by moving a job forward, a better schedule is obtained, it is assumed that better schedules can be achieved by maintaining the forward movement and not allowing a backward movement.

Table 3.1: Constructive and improvement heuristics for the PFSP

Year	Author/s	Acronym	Type[a]	Comments[b]
1954	Johnson	Johns	C	Exact for two machine case
1961	Page	Page	C	Based on sorting
1964	Dudek and Teuton		C	Based on Johnson's rule
1965	Palmer	Palme	C	Based on slope indexes
1970	Campbell et al.	CDS	C	Based on Johnson's rule
1971	Gupta	Gupta	C	Based on slope indexes
1972	Gupta		C	Three heuristics considered
1976	Bonney and Gundry		C	Based on slope matching
1977	Dannenbring	RA, RACS, RAES	C/I	Three heuristics considered: RA, RACS, and RAES
1980	King and Spachis		C	5 Dispatching rule based heuristics
1982	Stinson and Smith		C	6 Heuristics, based on TSP
1983	Nawaz et al.	NEH	C	Job priority/insertion
1988	Hundal and Rajgopal	HunRa	C	Palmer's based heuristic
1991	Ho and Chang	HoCha	I	Gap minimization in between jobs
1993	Sarin and Lefoka		C	Last machine idle time minimization
1998	Koulamas	Koula	C/I	Two phases, 1st Johnson-based, 2nd phase improvement by job passing
2000	Suliman	Sulim	I	Job pair exchange
2001	Davoud Pour	Pour	C	Job exchanging
2003	Framinan et al.		C	Study on the NEH heuristic

[a] C: Constructive, I: Improvement.

[b] Makespan is the primary objective

3.1.3. METAHEURISTICS FOR THE FSP

Metaheuristics are general heuristic procedures that can be applied to many problems, and, in our case, to the PFSP. These methods normally start from a sequence constructed by heuristics and iterate until a stopping criterion is met. There

is plenty of research work done for the PFSP and metaheuristics. Table 3.5 includes a summary of the some of the noteworthy papers mainly dealing with simulated annealing (SA), Tabu search (TS) and genetic algorithms (GA) and other metaheuristics, as well as hybrid methods. Makespan is the primary criterion of performance also for these heuristics for the PFSP.

Table 3.2: Metaheuristics for the permutation flow-shop problem

Year	Author/s	Acronym	Type	Comments
1989	Osman and Potts	SAOP	SA	
	Widmer and Hertz	Spirit	TS	Initial solution based on the OTSP
1990	Taillard		TS	
	Ogbu and Smith		SA	
1993	Werner		Other	Path algorithms
	Reeves		TS	
1995	Chen et al.	GAChen	GA	PMX crossover
	Reeves	GAReev	GA	Adaptive mutation rate
	Ishibuchi et al.		SA	Two SA considered
	Zegordi et al.		SA	Combines sequence knowledge
	Moccellin		TS	Based on SPIRIT
1996	Murata et al.	GAMIT	Hybrid	GA + Local Search/SA
	Nowicki and Smutnicki		TS	Neighbourhood by blocks of jobs
1998	Stützle	ILS	Other	Iterated Local Search
	Ben-Daya and Al-Fawzan		TS	Intensification + diversification
	Reeves and Yamada		GA	GA operators with problem knowledge
2000	Moccellin and dos Santos		Hybrid	TS + SA
2001	Ponnambalam et al.	GAPAC	GA	GPX crossover
	Wodecki and Bozejko		SA	Parallel simulated annealing
2003	Wang and Zheng		Hybrid	GA + SA, multicrossover operators

The methods listed in tables 3.4 and 3.5 are the most known heuristics and metaheuristics and also some original methods that are either recent and have not been evaluated before or some that incorporate new ideas not previously used by other algorithms.

3.2. COMPUTATIONAL EVALUATIONS AND INSIGHTS

The discussion for the evaluation of various heuristics generating permutation schedules has given a lot of insight to the main direction followed throughout this thesis study. At this point, the outcome of the research done by Dannenbring (1977) provides valuable insights for the way of generating non-permutation schedules. Dannenbring evaluates the following 11 heuristics;

- rapid access with closed order search (RACS),
- rapid access with extensive search (RAES),
- individual exchange heuristic (IE),
- and the group exchange heuristic (GE),

as the improvement procedures. Adding those four improvement procedures, he also considers the following five heuristics;

- rapid access (RA),
- slope order index (SO),
- merging (M),
- pairing (P),
- linear branch and bound (LBB)

as the solution-generating procedures which give a single solution. Additionally the two heuristics, namely CDS and random search (R) generate multiple solutions from which the best one is chosen.

While making extensive testing over a total of 1580 problems with the derivation of solutions using each heuristic, Dannenbring divides the discussion of the results into two parts on the basis of the problem size. The problems having number of jobs in between 3 and 6 are treated as small, and the ones having number of jobs in between 7 and 50 are referred to as large problems. It can be seen from Table 3.3 that RAES heuristic outperforms all other heuristics based on all criteria for the small problems. When the performance of RA with two of its derivations, namely RACS and RAES is compared, it is clearly evident that using a solution improvement routine is clearly advantageous.

Table 3.3: Evaluation results of heuristics on small problems (Dannenbring 1977)

Solution Method	Relative Error (%)	Consistency	Error Potential Ratio (%)	Proportion Optimal* (%)	Improvement Potential (%)	Sampling Quality (%)
RAES	0.64	3.18	2.32	75.86	1.33	8.40
RACS	1.30	7.87	4.66	63.13	3.02	10.28
CDS	1.73	11.66	5.68	55.47	3.09	10.50
M	1.74	11.31	5.97	56.88	3.92	11.29
R	2,03	14.52	7.12	45.00	4.12	11.68
GE	2.02	25.88	8.82	50.32	5.19	13.15
RA	3.65	33.34	13.57	34.77	11.21	18.36
LBB	3.82	42.84	13.25	42.89	10.12	17.44
SO	3.98	37.24	14.26	29.78	10.46	19.57
P	4.38	42.51	13.77	29.22	9.04	17.13
IE	4.99	69.70	15.71	29.22	11.92	20.17
Average	2.82	27.28	9.56	47.65	6.67	14.36

* Percentage of the solutions equaling the optimum or estimate of the optimum makespan.

The RA heuristic, which examines a single solution, is below average in performance. The RACS and RAES procedures, which improved the output solution of RA, did considerably better. It has also to be noted that the four of the five worst-performing algorithms are single-shot solution generating heuristics. This substantiates the intuitive notion that solution improvement heuristics are preferable to those that consider only one situation. The later stages will provide strong arguments for the benefits of the use of an initial schedule at hand prior to developing NPS, and employing simple sequencing rules while making partial interchanges (i.e. job-passing) in order to get the improved NPS for the flowshop. The jump made by R from being fifth to best indicates a decline in the performance of all heuristic methods, as the problem size is enlarged. RACS and RA together declined from second to fourth and seventh to eight respectively, as RACS uses RA's output as the initial solution.

Table 3.4: Evaluation results of heuristics on large problems (Dannenbring 1977)

Solution Method	Maximum Relative Error (%)	Estimated Relative Error (%)	Minimum Relative Error (%)	Consistency	Prop. Opt. or Est'd. (%)	Prop. Best Heuristic* (%)
RAES	4.96	1.58	1.52	7.67	19.38	71.25
R	6.70	3.17	3.11	16.56	6.88	18.13
CDS	7.62	4.11	4.06	29.71	12.50	15.00
RACS	7.80	4.26	4.20	31.53	13.13	14.38
M	8.25	4.68	4.62	37.14	7.50	11.25
GE	9.27	5.68	5.63	48.83	0.63	5.00
SO	9.76	6.18	6.12	54.96	2.50	3.75
RA	10.21	6.61	6.55	65.55	5.00	5.00
LBB	11.97	8.19	8.13	101.88	0.63	0.63
P	12.09	8.40	8.34	93.90	9.38	10.00
IE	12.83	9.19	9.13	119.72	0.63	1.25
Average	9.22	5.64	5.58	55.22	7.11	14.15

* Percentage of the solutions equaling the best heuristic solution (minimizing makespan).

The further decline in RACS performance indicates that the enlarging solution space requires more number of interchanges rather than a single interchange, as RACS with a single interchange cannot get close to the optimum on larger problems. The RAES algorithm remains best (with an increasing gap over performance), which gives an insight that RAES is a good candidate for use in future comparisons with non-permutation schedule generators executing in an 'improvement' manner. Dannenbring also declared that the hardest problems for heuristics to solve (i.e., more subject to error) are not necessarily the largest problems, but are in fact the intermediate-sized problems where the number of jobs is approximately equal to the number of machines.

3.3. BASIC MOTIVATION FOR DEVELOPING NPS

For the m-machine flow shop scheduling problem, most studies are performed in order to develop permutation schedules. Parallel to the discussion and research made

for the development of new heuristic methods, which generate permutation schedules, some researchers have emphasized the need for obtaining non-permutation sequences through steps as they provide better solutions with performance indicators closer to optimal scenario. In their relevant paper, Potts et al. (1991) have shown that for the problem of minimizing maximum completion time, developing permutation schedules becomes very costly. Potts et al. proven this result by exhibiting a family of instances for which the value of best permutation schedule is worse than that of the true optimal schedule by a factor more than $\sqrt{m}/2$.

The objective for the permutation flow shop problem is taken as to find a schedule, which minimizes the maximum completion time of any job, i.e. $C_{\max} = \max_j (C_j)$. All schedules are;

- non-preemptive schedules,
- minimizing $C_{\max} = \max_j (C_j)$.

Most research has focused on permutation schedules, because of the relative combinatorial simplicity. Unfortunately, this simplicity is bought at the price of drastically inferior schedules in terms of minimizing maximum completion time (see Table 3.7). The purpose of the following derivations is to study the worst-case behavior of the ratio of the maximum completion time of an optimum permutation schedule, denoted by $C^*_{\max}(\pi)$, to the optimum value $C^*_{\max}$. Potts et al. focused on the ratio of the outcome of permutation schedule to that of the optimal one, namely: $C^*_{\max}(\pi)/C^*_{\max}$, and they have indicated:

- $p(I) =$ ratio for instance I (proven that not bounded by any constant)
- $p(I_m) \geq \lfloor \sqrt{m} + 1/2 \rfloor / 2$ (where the instance I_m involves m machines).

The example with case I_{2n} is given in order the make things easy to grasp:

- there are n jobs to be processed in $2n$ machines

$$p_{ij} = \begin{cases} 1 & \text{if i = j + n or i = n + 1 - j} \\ 0 & \text{otherwise} \end{cases}$$

This guarantees that the processing time is 1 just for two operations per each job, and those two operations are distinct for each job to be processed. Also, when $p_{ij} = 0$, this value should rather be interpreted as an arbitrarily small positive constant. It is easy to see that $C^*_{\max} = 2$, as all the jobs are completed at the end of the second time unit. The fundamental insight into analyzing the length of the permutation schedules for these instances is the following easy fact:

- For the instance I_{2n}, $C_{\max} = n+1$ for the schedules given by either of the formulations $1,2,.....n$ or $n, n-1,.....1$. Figure 3.4 involves a simple illustration of this fact on a Gantt chart.

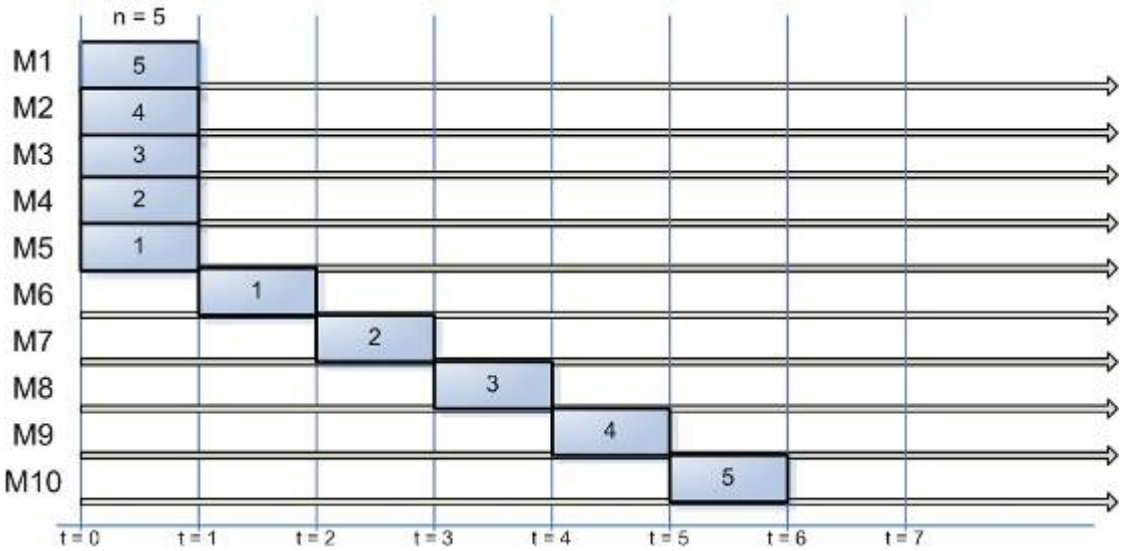

Figure 3.4: Gantt chart of the permutation schedule for the instance I_{2n}

This simple fact has important consequences; Potts et al. (1991) prove that if there is an increasing or decreasing subsequence of length s in the permutation, then for this permutation schedule $C_{\max} \geq s+1$.

3.4. A RECENT PAPER

Just before the completion of this thesis study, a very recent study by Liao et al. (2006) has been effective on the directions of the computational evaluations. Liao et al. present an extensive computational investigation concerning the performance comparison between permutation and non-permutation schedules. The computational results indicate that in general, there is little improvement made by non-permutation schedules over permutation schedules with respect to completion-time based criteria, for the problem instances where percentage of missing operations are low.

By making a comparison of the results of NPS-generating heuristics, namely Tandon's Simulated Annealing (SA) based heuristic (Tandon et al. 1991) and Pughazhendhi et al.'s (2002) heuristic with the resultant values of permutation schedules and optimal values, Liao et al. point out that the percentage improvements of NPS generated by most of the heuristics is not that much (i.e. that significant) based on the makespan as the primary performance criterion. By using Taillard's benchmark problems (Taillard 1993) and NEH heuristic as the PS generator for problem instances, Liao et al. have demonstrated the following results on the basis of computational experiments:

a) The non-permutation schedule requires much more computation time than the permutation schedule. The CPU time required for obtaining the NPS for the same problem instance compared to the PS is 5 to 10 times more.

b) The number of improved problems (i.e. non-permutation schedules better than permutation schedules) is small with respect to the $C_{\max}$ and $T_{\max}$, but it is quite large with respect to the $\sum C_j$, $\sum w_j C_j$, $\sum T_j$ and $\sum w_j T_j$ (weighted tardiness) criteria.

c) For the makespan as the primary performance criterion, the percentage improvements are rather small and limited. The percentage of problems that a

permutation schedule can be improved by a non-permutation schedule is high, except for the $C_{\max}$ criterion.

These results actually have driven the contents of this thesis to also involve the flow-time as the secondary performance criterion, in order to carefully assess the improvement made by the proposed heuristic approach generating NPS.

CHAPTER 4

THE PROPOSED APPROACH

4.1. JOB-PASSING: THE TOOL FOR NPS GENERATION

As discussed earlier in detail; flowshop scheduling problems are usually handled as permutation flowshops, because of the computational complexity associated with the non-permutation schedules. However, the technical framework in real-life cases often allows jobs to pass each other during the execution of various operations through machines within a flowshop. Having a missing operation of a job on a machine, this 'job passing' becomes a natural way to process jobs unless the underlying system is not an inflexible flow-line. For the case of the generation of permutations schedules, missing operations are usually handled as zero processing time operations, in order not to allow job passing through machines. A study by Leisten and Kolbe (1998) has shown that even permitting job passing for missing operations, while keeping the permutation constraint improves total completion time only under rather specific circumstances. Leisten and Kolbe (1998) also propose 'partial permutation flowshop sequencing' to overcome these specific formal requirements with respect to the real-world problem setting.

The basic motivation for generation of non-permutation schedules based on the fact that for most of the flow-shops some jobs were not to be processed on every machine. The literature on flowshop sequencing (e.g. Graves 1981, Domschke et al. 1993) has usually been handling the missing operations as 0-processing time operations. Nevertheless, a job with missing operations visits every machine, although Sridhar and Rajendran (1993) pointed out that there may be differences in completion times between zero-processing times and missing operations. In the a real-world flow-line based manufacturing setting, having a missing operation on a machine allows the job to pass by this machine, independent of whether the machine may be busy with another job or there may even be a queue of jobs in front of this

machine. This way of scheduling jobs with missing operations is closer to the real-world problem, thus the strict permutation constraint has to be someway violated. The analysis of the situation of missing operations and the derivation of results for schedules of a (permutation) flowshop with missing operations follows.

For a given job sequence, let $S(i, j)$ and $T(i, j)$ be the starting and finishing time, and p_{ij} the processing time of job i in the job sequence $i \in [1,....,I]$ on machine j in the machine sequence $i \in [1,....,I]$, i.e.

$$T(i, j) = S(i, j) + p_{ij}.$$

For a permutation flowshop, the conventional recursive formulation of operation finishing times (T^C) is:

$$T^C(i, j) = \max\left[T^C(i-1, j); T^C(i, j-1)\right] + p_{ij}.$$

Instead of using this conventional recursive formula, Sridhar and Rajendran (1993) formulates the model where a job finished on one machine having a missing operation on the next machine is allowed to pass this next machine and go straight to the machine where it has its next non-missing operation, using the following formula:

Compute $T^{SR}(i, j)$ job by job and within the jobs, machine by machine according to

$$T^{SR}(i, j) = \begin{cases} \max\left[T^{SR}(i-1, j); V\right] + p_{ij} & \text{if } p_{ij} \neq \text{mo} \\ \quad \text{With } V = \text{finishing time of the last} \\ \quad \text{non - zero processing time operation} \\ \quad \text{of job } i \text{ on machines } 1,..., j-1 \\ T^{SR}(i-1, j) & \text{if } p_{ij} = \text{mo} \end{cases}$$

$p_{ij} = \text{mo}$ means that the operation (i, j) is a missing operation. V determines job i's earliest availability for processing on machine j.

The relative job-loading sequence on every machine remains constant as in permutation schedules. Sridhar and Rajendran (1993) show positive effects to the objective function's makespan and total flowtime. However, if $p_{ij} = \text{mo}$, then

$p_{i-1,j+1} \neq \text{mo}$ results in $S(i, j+1) \geq T(i-1, j+1)$ to keep the permutation on machine $j+1$. Therefore, it is obvious that a necessary condition for an improvement (reduction of the makespan, or as also stated as 'acceleration of the schedule') by using the approach of Sridhar and Rajendran is $p_{ij} = p_{i-1,j+1} = \text{mo}$. The following iterations give the renewed recursive formula of $T^{SR}(i, j)$ for $p_{ij} = p_{i-1,j+1} = \text{mo}$.

Evidently, if $p_{ij} = p_{i-1,j+1} = \text{mo}$,

$$\begin{aligned} T^C(i, j+1) &= \max\left[T^C(i, j); T^C(i-1, j+1)\right] + p_{i,j+1} \\ &= \max\left[\max\left[T^C(i-1, j); T^C(i, j-1)\right]\right] \\ &= \max\left[T^C(i-2, j+1); T^C(i-1, j)\right] + p_{i,j+1} \\ &= \max\left[T^C(i-2, j+1); T^C(i-1, j); T^C(i, j-1)\right] + p_{i,j+1} \end{aligned}$$

and

$$T^{SR}(i, j+1) = \max\left[T^{SR}(i-2, j+1); T^{SR}(i, j-1)\right] + p_{ij}.$$

Hence,

$$T(i-1, j) > \max\left[T(i-2, j+1); T(i, j-1)\right] \Rightarrow T^{SR}(i, j+1) < T^C(i, j+1).$$

To be more specific, for an acceleration of the schedule we need the following conditions:

$$p_{i,j} = p_{i-1,j+1} = \text{mo}$$

and

$T(i-1, j) > \max\left[T(i, j-1); T(i-2, j+1)\right]$. Figure 4.1 gives an illustration of the situation mentioned above. These specific requirements guarantee the permutation to be kept in the overall schedule while 'improving' the completion time (at least of job i on machine j). This is a good explanation for scheduling while keeping the relative job sequence (Leisten & Kolbe, 1998).

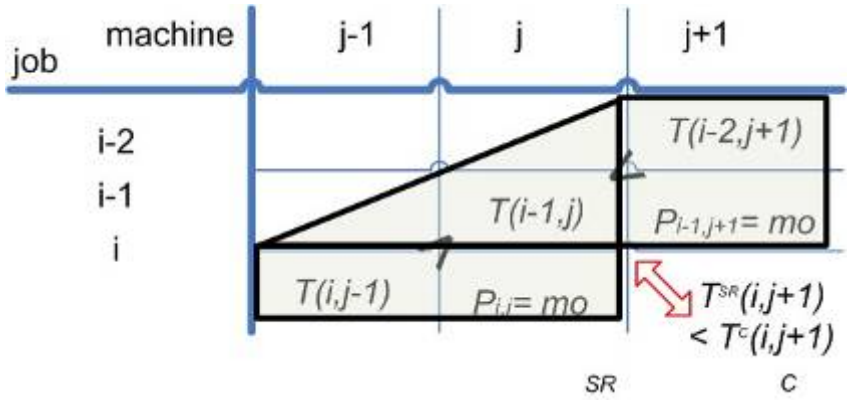

Figure 4.1: Necessary and sufficient conditions for $T^{SR}(i, j+1) < T^{C}(i, j+1)$

The conditions are 'necessary and sufficient' for an additional acceleration overlaying possible effects caused by an identical situation 'north-west' of operation (i, j) in Figure 4.1.

Allowing job-passing seems to be more adequate with respect to the real-world problem unless it is not an inflexible flowshop rather than keeping the strict permutation schedule through each machine.

A job i with missing operation on machine j can directly be appended to the queue in front of machine $j'+1$ for $j' \geq j$, where it has its next non-missing operation when it is finished on machine $j-1$. Since from the real-world view it is obvious that this violation of permutation constraint might be allowed, we propose to keep the new sequence of jobs throughout the machines down the flowshop.

To guarantee an at least myopic acceleration of the schedule, passing could be restricted to situations where job i can start on machine $j'+1$ before job $i-1$ can start on this machine, i.e.

$T(i-1, j') > \max\left[T(i, j-1); T(i-2, j'+1)\right]$ (see Figure 4.1). (Equation 4.1)

This relaxation of permutation flowshop is called 'partial permutation flowshop' by Leisten & Kolbe (1998), since job passing is allowed only in the appearance of

missing operations, whereas the permutation assumption is kept at every non-missing operation.

If missing operations exist and the conditions stated above are satisfied, the advantages of job passing might occur when considering the objective functions total flowtime or makespan. However, the important thing is that; the effect of job passing usually cannot be evaluated without considering the whole schedule. Therefore, at every point of applicability, branching into job passing and non-job passing seems to be reasonable. If many missing operations occur, Leisten & Kolbe (1998) recommends the application of branch-and-bound procedures. The partial permutation therefore, is realizable with the following example algorithm as an example algorithm, which employs job-passing:

Step 1: Generate a good initial permutation [e.g. by one of the methods mentioned in King & Spachis 1980, Park et al. 1984, Lahiri et al. 1993, using, e.g. the classical approach (Step 2)].

Step 2: Search machine by machine and within the machines job by job for the next missing operation (i, j). If this is the first iteration, start with job 1 on machine 1.

Step 3: When the inequality $T(i-1, j') > \max[T(i, j-1); T(i-2, j'+1)]$ does not apply continue with Step 4. Else, using Step 3, calculate the finishing times up to the relevant finishing times $T(i, j-1), T(i-1, j'), T(i-2, j'+1)$ with j' being the machine where job i has last consecutive missing operation behind machine $j-1$. Analyze whether necessary conditions for partial permutation are fulfilled. If not, continue with Step 2.

Step 4: Branch:

- keep the permutation unchanged up to the next node (next missing operation),
- put job i with missing operation on machine j in front of its next machine $j'+1$ with non-missing operation (using the FCFS-rule).

Step 5: For each branch, continue with Step 2.

The algorithm given above in a simplistic manner will terminate when no more missing operations exist and every branch has been evaluated for all jobs. (The integration of a bounding procedure – if necessary – is obvious.) Here, it has to be noted that while calculating finishing times (via step 3), a further missing operation (i^*, j^*) with $i^* < j$ and $j < j^* < j'$ might appear 'north-east' of operation (i, j). Since without a decision how to treat this missing operation finishing times cannot be calculated, Steps (3) and (4) of the algorithm may be executed at this additional node (i^*, j^*) as an inner loop of the algorithm (Leisten & Kolbe, 1998). Alternatively, to reduce complexity it should be considered

(a) to ignore the inequality $T(i-1, j') > \max\left[T(i, j-1); T(i-2, j'+1)\right]$ at node (i, j), or

(b) to analyze whether at least myopic acceleration can be guaranteed by taking job i only to machine j^*.

Then, "Equation 4.1" has to be modified as

$$T(i-1, j') > \max\left[T(i, j-1); T(i-2, j^* + 1)\right]$$

in order to use this information at Step (4b).

Tables 4.1, 4.2, and 4.3 include an illustration of the use of job-passing through flowshops. Table 4.1 includes the operation durations for a 4-job, 3-machine problem with missing operations of jobs. The missing operations for jobs are indicated by the zero processing times for the job on the corresponding machine. Tables 4.2 and 4.3 include a permutation and non-permutation schedule of the jobs given at table 4.1 respectively. A close look into the table 4.3 reveals that the permutation sequence could be changed so as to result in a feasible non-permutation schedule (i.e., a partial permutation schedule as discussed within the beginning of this section). This finding in fact forms the basics of the heuristic approach that has been developed and presented throughout this thesis study.

Table 4.1: Processing times for a 4-job, 3-machine problem

Job	**Machine**		
	1	**2**	**3**
1	20	0 *	30
2	0 *	10	10
3	10	10	0 *
4	20	70	80

Looking at the Table 4.1 (and recalling the previous discussions) it is obvious that solutions obtained by employing permutation schedules (Table 4.2) need not be optimal for a problem with more than three machines with makespan objective. This finding, along with some job-passing through the schedule leads us to the timetable (indicating the start times and finishing times for each job) illustrated in Table 4.3.

Table 4.2: Starting and finishing times of jobs in permutation sequence [4123]

Job	**Job** *i*	**Machine**		
		1	**2**	**3**
1	**4**	0/20	20/90	90/170
2	**1**	20/40	-	170/200
3	**2**	-	90/100	200/210
4	**3**	40/50	100/110	-
Job sequence		[413]	[423]	[412]

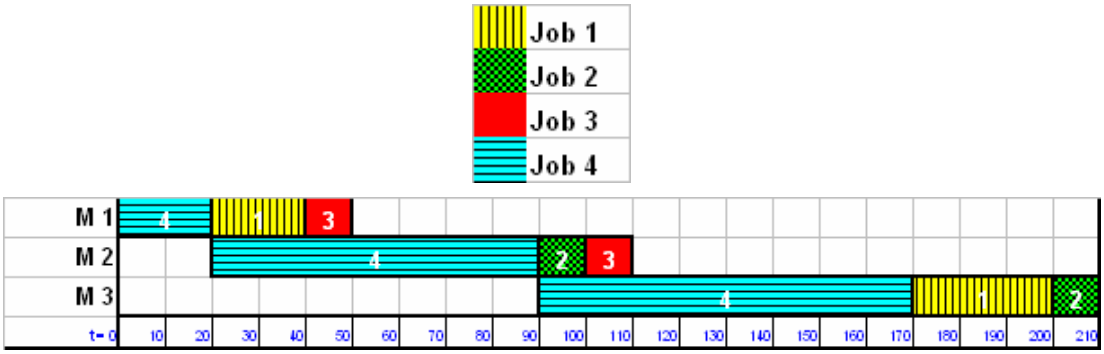

Figure 4.2: Gantt chart representing Table 4.2

Table 4.2 gives the representation of the schedule with the permutation sequence [4132] through each of the three machines with a makespan value of 210 and the total flowtime value of 690. The missing operations for jobs are not indicated within the job sequences for a given machine. In between machine 1 and machine allowing job 2 to pass before jobs 1 and 3, and in between machine 2 and machine 3, allowing job 1 to pass before job 4 gives the schedule in Table 4.3.

Table 4.3: Starting and finishing times of jobs in non-permutation sequence

Job	**Job** *i*	**Machine**		
		1	**2**	**3**
1	**4**	0/20	20/90	90/170
2	**1**	20/40	-	40/70
3	**2**	-	0/10	10/20
4	**3**	40/50	90/100	-
Job sequence		[413]	[243]	[214]

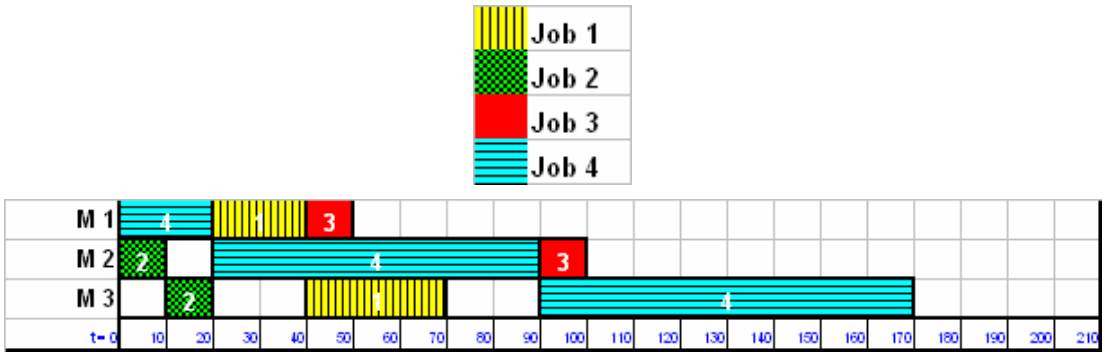

Figure 4.3: Gantt chart representing Table 4.3

The non-permutation schedule generated by a single move at each stage (machine) gives a total makespan value of 170, with a total flowtime value of 360. There has been a significant improvement due to both performance criteria even for the case of this simple illustrative example.

4.2. DERIVING NPS FROM A GIVEN PS

The newly developed heuristic has a similar logic to the heuristic proposed by Pugazhendhi et al. (2002). The logic of the proposed heuristic (parallel to the discussions at section 4.1) is depicted at Figure 4.4. The heuristic first obtains a seed permutation sequence (which is obtained either by RAES of NEH heuristic) and tries to execute multiple job-passing tasks at various stages without violating the following feasibility restriction, namely each machine processes one and only one job at any point in time.

In order to setup the heuristic, the following notation and terminology will be employed through later stages:

n / m	Number of jobs/machines available at time zero
$t(i, j)$	Processing time of job i on machine j
π	Set of jobs already scheduled, out of n jobs, at a given time instant τ
i	The job at hand, which is going to be scheduled
$ST(i, j)$	Starting time for job i on machine j
$FT(i, j)$	Finishing time for job i on machine j
$\{Seq^j\}$	Job sequence on machine j (the sequence in which jobs are processed on machine j).
n_j	Number of jobs processed on machine j
s_k^j	Job processed in the k-th position of $\{Seq^j\}$ on machine j
$RST(k, j)$	On machine j, start time of job i found in the k-th position of $\{Seq^j\}$
$RFT(k, j)$	On machine j, finishing time of job i found in the k-th position of $\{Seq^j\}$

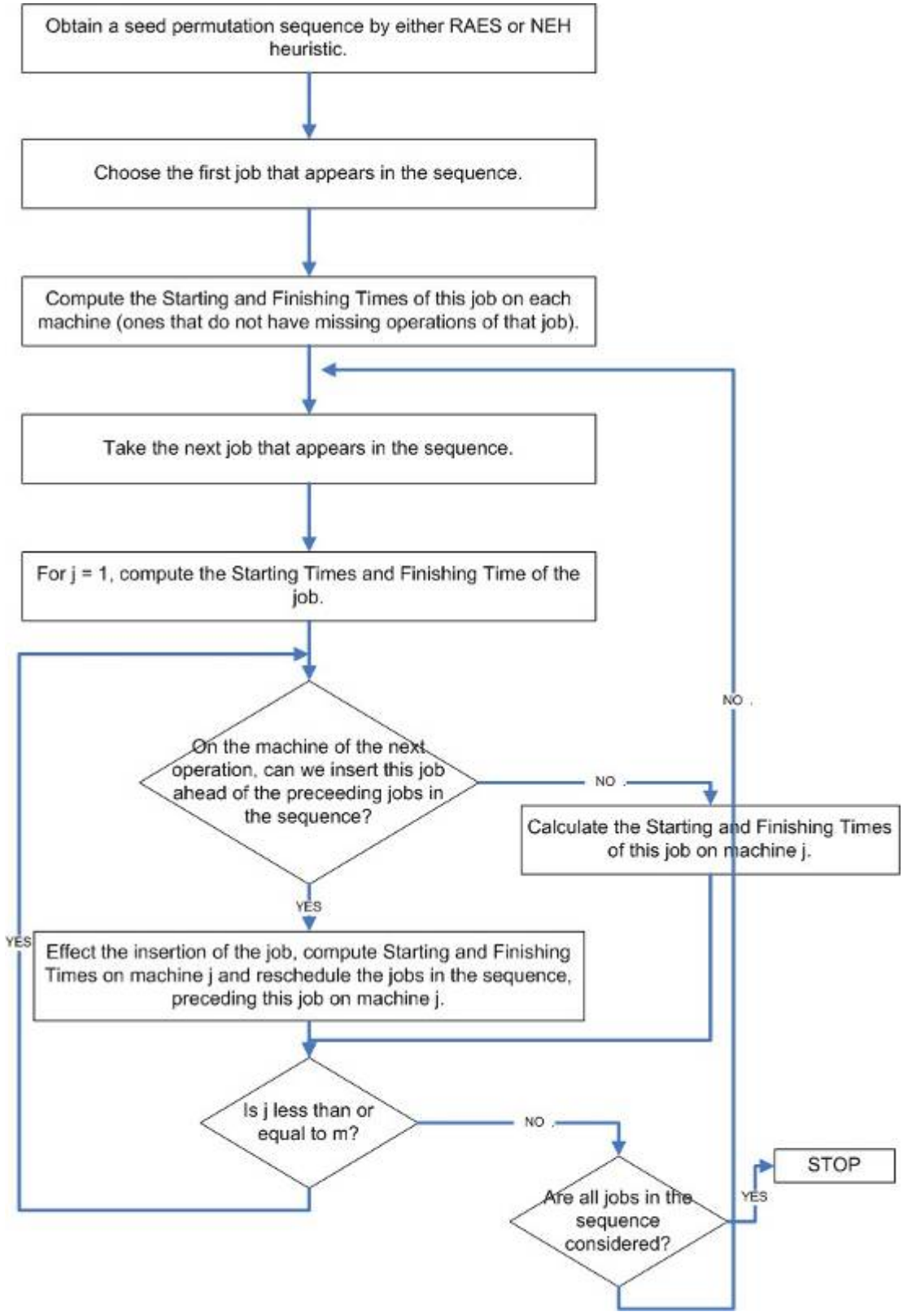

Figure 4.4: Main schematic of the heuristic procedure proposed to generate NPS

US_j	Set of unscheduled jobs on machine j
CT_i	Completion time of the last operation of job i
MS	Makespan of the scheduled job-set
w_i	Weight (or relative importance or relative holding cost) for job i
F_π	Total flowtime of jobs in π (i.e. sum of CT_i's over all scheduled jobs)
W_π	Total weighted flowtime of jobs in π (i.e. sum of $(w_i \times CT_i)$'s over all scheduled jobs)

As depicted in Figure 4.4, the main principle of the heuristic procedure is that, one at a time jobs are taken from the permutation sequence (without violating the order assigned by the permutation sequence generating heuristic, namely RAES or NEH), and find a suitable place for that job in the earliest idle time interval ahead (which is sufficient for the completion of the job selected) in order to process the inserted job.

4.2.1. INITIALIZATION

The key steps involved in the proposed heuristic method are listed below. To initialize the procedure at time zero, we first set s_0^j for $j = 1,2,3.....m$ as ϕ, the null job, i.e. $\{Seq^j\} = \{\phi\}$ for all machines. Let $ST(\phi, j) = FT(\phi, j) = 0$ and $RST(0, j) = RFT(0, j) = 0$ for all machines. Let $n_j = 0$, for $j = 1,2,3,...m,$ (note that it is job ϕ used in ST and FT, and "0" is used in RST and RFT in order to denote the job position number).

A permutation sequence is obtained by using RAES (Dannenbring 1977) or NEH (Nawaz et al. 1983) in order to have the initial sequence prior to the execution of the procedure.

4.2.2. THE HEURISTIC PROCEDURE

The heuristic procedure based on the initializing parameters in the previous section is as follows:

STEP 1: With respect to the NPS-set, let $\{Seq^j\}$ be the sequence in which jobs are already scheduled on machine j. Having the initial permutation sequence at hand, take the unscheduled jobs, say job-set i, (unscheduled with respect to the NPS-set) be now taken up for scheduling in the NPS-set.

STEP 2: Set $j = 1$ and $T = 0$.

STEP 3: If $t(i, j) > 0$, then proceed to the next step; else go to STEP 10 (i.e., the job has a missing operation on machine j).

STEP 4: Set $k = -1$. (4.1)

STEP 5: Let

$k = k + 1$ (4.2)

(k is the job position in the sequence $\{Seq^j\}$ on machine j)

If $k \neq n_j$ (the position in the sequence is not the same as the number of jobs processed on machine j)

then go to the next step; else go to STEP 9.

STEP 6: At this step, check for the possible insertion of the chosen job i, i.e., processing job i ahead of the preceding jobs (preceding in the permutation sequence) on machine j.

If $(\max\{T; RFT(k, j)\} + t(i, j) \leq RST(k+1, j)$ (4.3)

Then proceed to the next step; else return to STEP 5 for checking

possible insertion of job i in the next position.

STEP 7: Feasible insertion of job i, in $\{Seq^j\}$ is done appropriately and all $RST(k, j)$'s and $RFT(k, j)$'s for all preceding jobs is done.

For $p = 0$ to k, do the following:

{

$$s'^{j}_{p} = s^{j}_{p}, \tag{4.4}$$

$$RST'(p, j) = RST(p, j) \tag{4.5}$$

and

$$RFT'(p, j) = RFT(p, j) \tag{4.6}$$

}

Let

$$s'^{j}_{k+1} = i, \tag{4.7}$$

$$RST'(k+1\,j) = \max\{RFT(k, j); T\}, \tag{4.8}$$

$$RFT'(k+1\,j) = RST'(k+1, j) + t(i, j), \tag{4.9}$$

$$ST(i, j) = RST'(k+1, j), \tag{4.10}$$

$$FT(i, j) = RFT'(k+1, j), \tag{4.11}$$

and

$$t = FT(i, j), \tag{4.12}$$

For $p = (k+1)$ to n_j do the following:

{

$$s'^{j}_{p+1} = s'_{p} \tag{4.13}$$

$$RST'(p+1\,j) = RST(p, j) \tag{4.14}$$

$$RFT'(p+1\,j) = RFT(p, j) \tag{4.15}$$

}

Set

$$n_j = n_j + 1. \tag{4.16}$$

STEP 8: For $p = 0$ to n_j do the following:

{

$$s_p^j = s_p'^j \tag{4.17}$$

$$RST(p, j) = RST'(p, j) \tag{4.18}$$

and

$$RFT(p, j) = RFT'(p, j) \tag{4.19}$$

}

Reshaping the sequence of jobs on machine j.

Then, go to STEP 11.

STEP 9: This step involves the computation of starting time and finishing time of job i on machine j is done without any insertion ahead being feasible.

$$RST(n_j + 1, j) = \max\{T; RFT(n_j, j)\}, \tag{4.20}$$

$$RFT(n_j + 1, j) = RST(n_j + 1, j) + t(i, j), \tag{4.21}$$

$$ST(i, j) = RST(n_j + 1, j), \tag{4.22}$$

$$FT(i, j) = RFT(n_j + 1, j) \tag{4.23}$$

and

$$T = FT(i, j). \tag{4.24}$$

Increment

$$n_j = n_j + 1 \tag{4.25}$$

Set

$$s_{n_j}^j = i. \tag{4.26}$$

Go to STEP 11.

STEP 10: Set

$$ST(i, j) = FT(i, j) = \mathrm{B} \tag{4.27}$$

where B is a large number implying that there is no operation of job i on machine j.

STEP 11: Increment,

$$j = j + 1 \tag{4.28}$$

If $j \leq m$ then go back to STEP 3; else proceed to STEP 12.

STEP 12: Compute the schedule performance measures as scheduling of job i is done.

Set

$$CT_i = T, \tag{4.29}$$

$$MS = \max\{CT_i\} \tag{4.30}$$

over all scheduled jobs in the NPS-set.

STEP 13: STOP if all jobs are scheduled in the NPS-set; else go to STEP 1.

4.2.3. THE PROPOSED HEURISTIC PROCEDURE

A further improvement can be made over the performance of this procedure by making a slight shift in the main logic lying behind this algorithm (see Figure 4.5). For the case of the former procedure, the lacking thing was that at each stage, while creating the NPS-set of jobs to be scheduled for the remaining earliest idle time span of a given machine, the heuristic procedure strictly obeys to the previously formed permutation sequence coming from the previous machine. Namely, when the unscheduled jobs are being assigned to the earliest idle time span, the one with the preceding position in the permutation sequence is considered first. In other words, the jobs that will go through STEPS 3, 4, 5, 6, and finally 7 is already determined by the initial permutation sequence, as long as the computation of starting time and finishing time of job i on machine j is done without any insertion ahead being feasible (see STEP 7).

In order to overcome this situation, the logic of the heuristic procedure is shifted, to enable the simultaneous consideration of unscheduled jobs (with missing operations) and assigning them to the earliest inserted idle time by a sequencing rule, a dispatching rule or by employment of a flowshop heuristic generating permutation sequences. The procedure applied at this step is intended to make the overall NPS schedule more robust enabling it to further minimize makespan by careful consideration of this criterion during the job-passing phase.

As depicted in Figure 4.5 the procedure now considers all the candidates for job-passing into any inserted idle time existing in the initial permutation schedule based on a so called dispatching rule working under the limitation of the Finishing Time of the first job ahead which the jobs move. With this new rule, STEPS 4 through 7 within for the primal procedure undergoes the following change.

STEP' 3 For jobs with $t(i, j) > 0$, then proceed to the next step; else go to STEP 10 (i.e., the jobs have a missing operation on machine j).

STEP 4: Set $k = -1$. (4.31)

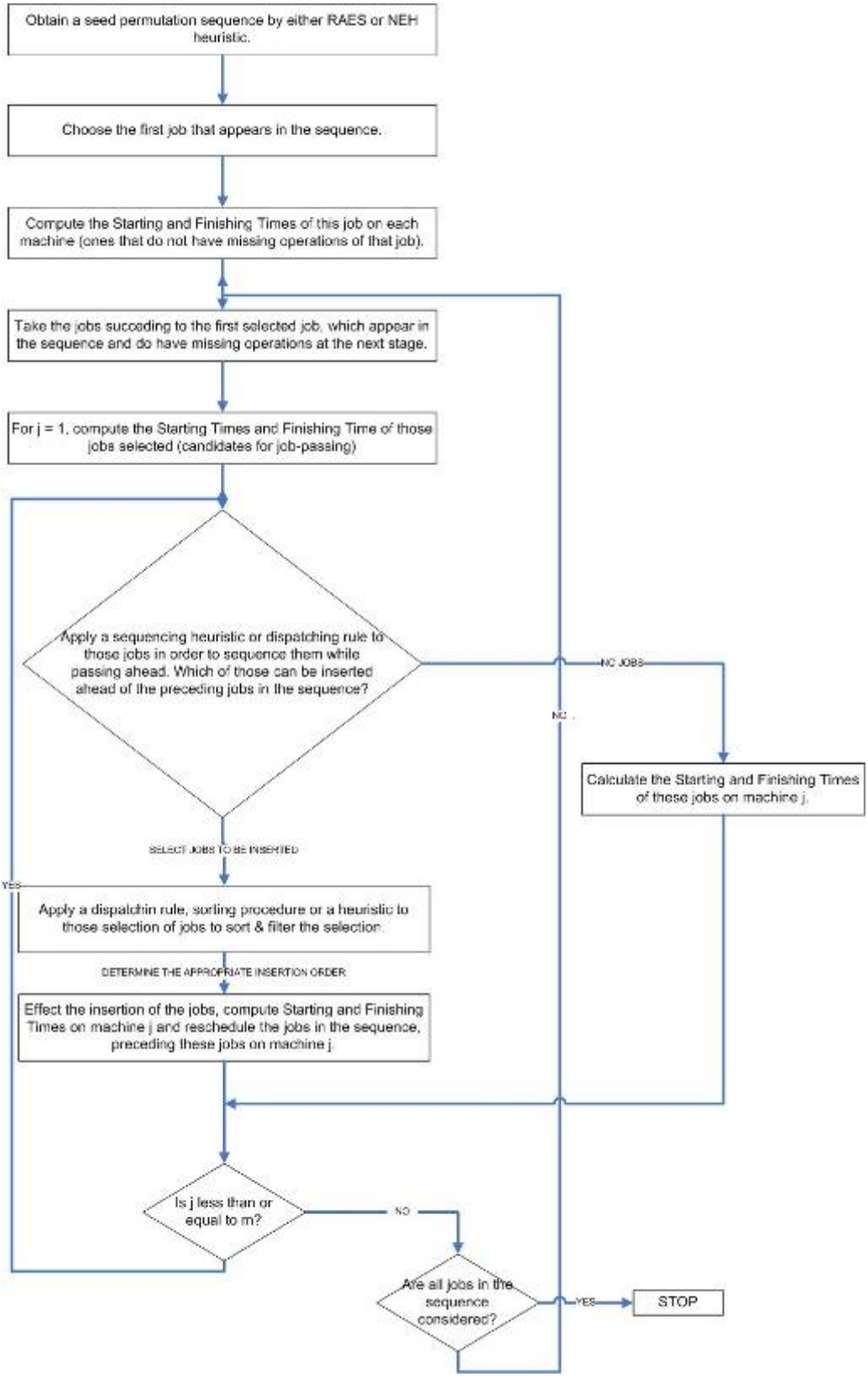

Figure 4.5: The reorganized procedure for generating NPS-set at each stage

STEP 5: Let

$$k = k + 1 \tag{4.32}$$

(k is the job position in the sequence $\{Seq^j\}$ on machine j)

If $k \neq n_j$ (the position in the sequence is not the same as the number of jobs processed on machine j)

then go to the next step; else go to STEP 9.

STEP 6: At this step, check for the possible insertion of the chosen jobs, i.e., processing jobs ahead of the preceding jobs (preceding in the permutation sequence) on machine j.

Apply the appropriate rule for sequencing those jobs within themselves.

For all $i \in US^*$

$$\text{If } (\max\{T; RFT(k, j)\} + \sum t(i, j) \leq RST(k+1, j) \tag{4.33}$$

Then proceed to the next step; else cut off the jobs beginning from reverse order in the sequence till (4.33) holds, and return to STEP 5 for checking possible insertion of remaining (could not be inserted ahead) jobs in the next available position.

/* Set of candidate jobs which are sorted due to a rule before being inserted into the idle time ahead */

STEP 7: Feasible insertion of jobs for all $i \in US^*$, in $\{Seq^j\}$ is done appropriately and all $RST(k, j)$'s and $RFT(k, j)$'s for all preceding jobs is done.

For $p = 0$ to k, do the following:

{

$$s'^j_p = s^j_p, \tag{4.34}$$

$$RST'(p, j) = RST(p, j) \tag{4.35}$$

and

$$RFT'(p, j) = RFT(p, j) \tag{4.36}$$

}

Let

$$s'^{j}_{k+1} = i\,, \tag{4.37}$$

$$RST'(k+1\,j) = \max\{RFT(k, j); T\}\,, \tag{4.38}$$

$$RFT'(k+1\,j) = RST'(k+1, j) + t(i, j)\,, \tag{4.39}$$

$$ST(i, j) = RST'(k+1, j)\,, \tag{4.40}$$

$$FT(i, j) = RFT'(k+1, j)\,, \tag{4.41}$$

and

$$t = FT(i, j)\,, \tag{4.42}$$

For $p = (k+1)$ to n_j do the following:

{

$$s'^{j}_{p+1} = s'_p \tag{4.43}$$

$$RST'(p+1\,j) = RST(p, j) \tag{4.44}$$

$$RFT'(p+1\,j) = RFT(p, j) \tag{4.45}$$

}

Set

$$n_j = n_j + 1 \quad . \tag{4.46}$$

Chapter 5 includes the evaluation of the new heuristic procedure together with the conventional heuristic developed by Pugazhendhi et al. (2002). While comparing the heuristics, the upgraded form employs the following sequencing rules and/or algorithms in order to derive NPS from a given PS:

- generation of initial PS → NEH heuristic for minimizing makespan and Ho's heuristic for minimizing total flowtime,
- intermediary sorting/dispatching rule → NEH heuristic for makespan problems, and Shortest Processing Time (SPT) Rule for problems based on total flowtime as the performance criterion

CHAPTER 5

COMPUTATIONAL RESULTS

In this chapter the new heuristic procedure first has been demonstrated with some illustrative examples in order to show the performance enhancement provided by the new approach for simplistic cases. The examples are than formalized with the generation of a large set of example problems by varying the number of jobs, number of machines and percentage of missing operations respectively in order to come up with an experimental design. The problem instances are then solved by using different heuristic methods together with the newly developed procedure and computational results have been tabularized at the end of the chapter based on both makespan and the total flowtime as the performance criterion.

5.1. EXPERIMENTATION OF HEURISTIC PROCEDURES

5.1.1. ILLUSTRATIVE EXAMPLES

In order to visualize the conventional heuristic and the one derived from it, the following illustrative example would be beneficial. Consider the following 4-job, 5-machine problem as viewed in Table 5.1 with job processing times given below.

Table 5.1: Illustrative 4-job, 5-machine problem

Job *i*	**Machine** *j*				
	1	**2**	**3**	**4**	**5**
1	13	0	50	12	12
2	17	0	0	12	10
3	34	50	0	12	20
4	0	200	0	12	18

An entry of zero processing time indicates the missing operations for the job on the corresponding machine. Let us take the initial permutation sequence {4312} which has a makespan value of 304 (see Table 5.2) and a total flowtime value of 1110. When the conventional heuristic algorithm for generating non-permutation schedules is implemented over the permutation sequence {4312}, it is seen that non-permutation schedules are generated with the timetable given in Table 5.3 (entries indicate the start time (ST) and finish time FT of the job found in the i-th position in the sequence {4312}, i.e. job $[i]$ on machine j, when job $[i]$ is appended. These times are shown before and after the sign "/" respectively.

Table 5.2: Starting and finishing time of jobs for the permutation sequence

Job position	**Job**	**Machine**				
		1	**2**	**3**	**4**	**5**
1	**4**	-	0/200	-	200/212	212/230
2	**3**	0/34	200/250	-	250/262	262/282
3	**1**	34/47	-	47/97	262/274	282/294
4	**2**	47/64	-	-	274/286	294/304
Job loading sequence		{312}	{43}	{1}	{4312}	{4312}

Table 5.3: Starting and finishing time of jobs when NPS are adopted

	Machine				
	1	**2**	**3**	**4**	**5**
	-	0/200	-	200/212	212/230
	0/34	200/250	-	250/262	262/282
	34/47	-	47/97	97/109	109/121
	47/64	-	-	64/76	76/86
Job loading sequence	{312}	{43}	{1}	{2143}	{2143}

The resultant NPS are also presented in Tables 5.4 and 5.5. The makespan is observed to be 282 and the total flowtime has been drastically lowered down to 719, resulting in an improvement in the values of the performance measures obtained earlier by the adoption of the permutation sequence. Values of $RST(k, j)$ and $RFT(k, j)$ for the scheduled job-set $\{43\}$ can be seen in Table 5.4.

Further we have $s_1^j = 4$ for $j = 2,4$ and 5 (implying that job 4 is the first job to be scheduled on machines 2, 4, and 5), $s_1^j = 3$ for $j = 1$ (implying that job 3 is the first job to be scheduled on machine 1), and $s_2^j = 3$ for $j = 2,4$, and 5 (implying that job 3 is the second job to be scheduled on machines 2, 4 and 5). A good example to job-insertion, i.e. passing of a job ahead of the preceding job(s) can be seen when we consider job 1 for scheduling on machine 4. Corresponding to $j = 4$, invoking Step 5 of the proposed algorithm, we look out for the possible insertion of job 1 ahead of the preceding jobs in the permutation schedule, namely jobs 4 and 3. We check for the following invoking Step 6:

$$\text{If } (\max\{T; RFT(0,4)\} + t(1,4) \le RST(1,4); \text{ i.e. if } \max\{97;0\} + 12 \le 200 \text{ (see 4.33).}$$

Actually, what the heuristic procedure does is that processing of job 1 on machine 4 can be commenced and ended in between time units 97 and 109, respectively, because machine 4 remains idle during the time-span of 0 to 200. The fact that this check is satisfied indicates that job 1 can be processed prior to job(s) scheduled ahead of it on machine 4. This insertion brings re-computation of RST and RFT values of the jobs scheduled on the machine under consideration, leading to re-sequencing of the partial job set $\{431\}$ on machine 4 and resulting in the generation of NPS. Steps 7 through 10 have been essentially developed for this purpose (see equations 4.4 through 4.27 (for just one job insertion at a time) and equations 4.34 trough 4.46 (for passing ahead of multiple jobs at a time).

Table 5.4: Values of $RST(k, j)$ and $RFT(k, j)$ for the scheduled job-set {43}

Job position k	**Machine**				
	1	**2**	**3**	**4**	**5**
0	0/0	0/0	0/0	0/0	0/0
1	0/34	0/200	-	200/212	212/230
2	-	200/250	-	250/262	262/282

Table 5.5: Values of $ST(i, j)$ and $FT(i, j)$ for the scheduled job-set {43}

Job i	**Machine**				
	1	**2**	**3**	**4**	**5**
4	-	0/200	-	200/212	212/230
3	0/34	200/250	-	250/262	262/282
Job loading sequence	{3}	{43}	$\{\phi\}$	{43}	{43}

Similarly, applying the upgraded heuristic procedure brings us to the resultant NPS given in Table 5.6. For this case, the makespan value is kept as the level of 282 -still demonstrating improvement compared to the permutation schedule at hand- while the total flowtime is further improved (i.e. decreased), reaching a value of 649.

Table 5.6: Starting and finishing time of jobs when new NPS are adopted

	Machine				
	1	**2**	**3**	**4**	**5**
	-	0/200	-	200/212	212/230
	0/13	-	13/63	63/75	75/85
	13/30	-	-	30/42	42/52
	30/64	200/250	-	250/262	262/282
Job loading sequence	{132}*	{43}	{1}	{2143}	{2143}

* More than one insertion has been allowed at this stage.

5.1.2. GENERATION OF EXAMPLE PROBLEMS

The example problems are generated by varying the number of jobs, number of machines and the percentage of missing operations respectively as indicated in Table 5.7. For combination of number of jobs, number of machines and percentage of missing operations (n, m and p) 10 instances are created in order to obtain mean values for each instance and also to ensure homogeneity of variance and independence of the outliers. In total, $(10) \times 4 \times 5 \times 4 = 800$ problem instances have been formed in order to be optimized using different heuristic techniques.

Table 5.7: Values that each parameter takes per each experiment

	n =	m =	p =
$10\times$ instances for each combination of parameters	10	10	20%
	20	20	30%
	30	30	40%
	40	40	50%
		50	

At this stage, the notion of Taillard's (1993) benchmarking problems for flowshop schedules has been extensively employed. The computer code given in Appendix A is a modification of the C++ code used for generating Taillard's example problems. The difference is in terms of the ranges of the values that parameters n, m and p take; in the sense that instead of Taillard's range of 20 to 500 for number of jobs, the range of 10 to 40 have been employed in order to be conservative in terms of computation time of the proposed heuristic. The heuristic procedure, which is employing a sequencing rule at each stage (i.e. machine) in order to decide the sequence and number of jobs to be passed ahead, will be prone to high values of n because of the growing completion time of jobs.

The performance measues have been the maximum completion time (i.e. makespan) of jobs and secondarily the total flowtime of jobs flowing through machines. For the

total flowtime as the secondary criterion of assessment, the weights associated with each job is taken constant as 1, as no differentiation of relative holding cost has been assigned to any of the jobs.

All computer codes have been developed in Visual C++ 6.0 and computational experiments are conducted on an Intel Pentium II 3.192 MHz PC (total physical memory 512 mb) under the Windows XP operating system.

While evaluating the heuristic approaches, the following performance measures have been used:

i) The relative performance improvement (RPI) of the NPS-set over a permutation schedule, with respect to makespan is given by $RPI(MS) = (M - M') \times 100 / M'$, where M and M' respectively are the values of makespan, as computed by the permutation sequence obtained by Nawaz et al.'s (1983) NEH heuristic and the NPS-set obtained by implementing the proposed heuristic. The reason that the comparison is made with NEH heuristic is that, it is by far the best heuristic among a large group of heuristic methods (Ruiz & Maroto, 2005) providing permutation schedules to improve the makespan as the primary performance criterion. NEH heuristic is selected as the best performer among SPT Rule, LPT Rule, Johnson's Rule, Page's Heuristic, Palmer's Heuristic, Campbell, Dudek & Smith's CDS Heuristic, Gupta's Algorithm, Dannenbrings RA, RACS, and RAES and Ho and Chang's heuristic (all focusing on makespan criterion). $RPI_1(MS)$ corresponds to the RPI of the heuristic developed by Pughazhendhi et al. (2002) and $RPI_2(MS)$ stand for the RPI of newly developed heuristic allowing intermediary sorting of multiple jobs prior to passing ahead of the other jobs in the permutation sequence.

ii) The relative performance improvement (RPI) of the NPS-set over a permutation schedule, with respect to total flowtime is given by

$RPI(TFT) = (F - F') \times 100 / F'$, where F and F' respectively are the values of total flowtime, as computed by the permutation sequence obtained by Ho's algorithm (Ho, 1995) and the NPS-set obtained by implementing the proposed heuristic. For the objective of minimizing total flow time of jobs, heuristics have been developed by Miyazaki et al. (1978), Rajendran (1993) and Ho (1995). Of these heuristics, Ho's heuristic is the best performing heuristic (Ho, 1995). This heuristic generates a seed sequence by arranging the jobs in the ascending $\sum_{j=1}^{m}(m - j + 1) \times t_{ij}$ values (Rajendran & Ziegler, 2001) .

Later, improvement schemes based on pairwise interchange and insertion are employed until no significant improvement in the total flowtime of jobs is obtained. This heuristic is computationally more cumbersome than the heuristics of Miyazaki et al. And Rajendran, but it is more effective in minimizing the total flowtime of jobs than the other two heuristics.

iii) Mean RPI is the average of 10 values of RPI for a given problem set specified by (n, m and p), whereas Max RPI indicates the maximum RPI value out of the 10 problems in each set.

iv) Another performance measure is the number of problems (N), out of 10, for which an improvement in the performance criterion is realized for a given problem set defined by the parametric variables n, m and p.

5.1.3. EVALUATION OF THE HEURISTICS

The evaluation of the heuristics and computational experimentations are done separately for two criterions, namely makespan and the total flowtime. The comparisons of the heuristics for each of the performance criterion has been then presented within the outcome section of this chapter. Results are summarized and tabularized based on differen values of n, m and p.

5.1.3.1. MAKESPAN AS THE PRIMARY CRITERION

Tables 5.8 through 5.11 illustrate the relative performance of the two proposed heuristics by taking NEH heuristic as the base in terms of getting the best permutation schedule due to "makespan" as the performance criterion.

Table 5.8: Performance $(M = \max\{C_i\})$ evaluation of both NPS-sets for $p = 20\%$

n	m	**Pugazhendhi et al. (2002)**			**New heuristic procedure**		
		Mean RPI_1	Max RPI_1	N_1	Mean RPI_2	Max RPI_2	N_2
10	10	0,40	2,56	2	0,27	2,48	**2***
	20	0,61	2,75	4	0,53	2,84	3
	30	0,64	2,71	6	0,59	2,72	5
	40	0,66	4,16	6	0,82	4,11	5
	50	0,57	2,36	6	0,36	2,12	**7**
20	10	0,53	2,65	5	0,42	3,24	**6**
	20	0,16	2,56	4	0,17	1,95	3
	30	0,42	2,42	4	0,28	1,83	3
	40	0,31	1,50	5	0,35	0,96	3
	50	0,55	1,70	7	0,57	2,21	**7**
30	10	0,35	1,38	2	0,25	1,23	**2**
	20	0,30	2,29	3	0,23	2,33	2
	30	0,38	1,97	6	0,23	2,15	**7**
	40	0,47	1,03	8	0,47	1,26	**9**
	50	0,56	3,24	9	0,62	2,79	**10**
40	10	0,22	0,83	2	0,25	1,34	**3**
	20	0,36	1,77	5	0,35	1,48	**6**
	30	0,42	1,25	6	0,33	1,09	**7**
	40	0,42	1,18	8	0,44	1,57	7
	50	0,36	2,12	7	0,45	1,42	**7**

* For $N_2 \geq N_1$, values in the column are highlighted.

Table 5.9: Performance ($M = \max\{C_i\}$) evaluation of both NPS-sets for $p = 30\%$

n	m	**Pugazhendhi et al. (2002)**			**New heuristic procedure**		
		Mean RPI_1	Max RPI_1	N_1	Mean RPI_2	Max RPI_2	N_2
10	10	0,46	4,89	2	0,62	3,10	1
	20	0,94	4,51	5	1,09	3,36	4
	30	0,87	4,59	7	0,85	2,98	6
	40	1,15	3,38	8	1,34	4,71	**8**
	50	0,63	4,78	6	0,75	2,89	**6**
20	10	0,84	4,04	6	0,69	2,50	5
	20	0,89	4,87	6	1,00	4,19	**6**
	30	0,91	3,65	8	1,05	4,35	7
	40	0,97	3,40	9	1,08	4,70	8
	50	1,07	2,56	9	0,95	3,96	**9**
30	10	0,25	0,87	3	0,38	1,48	**3**
	20	0,70	3,18	9	0,72	3,88	**9**
	30	0,72	2,84	7	0,77	4,51	6
	40	0,72	1,42	8	0,77	2,76	**8**
	50	1,06	2,75	9	0,85	4,08	**9**
40	10	0,18	2,05	3	0,04	1,34	**3**
	20	0,71	2,66	9	0,98	4,12	7
	30	0,93	3,20	7	1,04	5,41	**7**
	40	0,95	2,34	10	1,10	4,09	**10**
	50	1,19	2,31	9	1,01	4,06	**10**

Table 5.10: Performance $(M = \max\{C_i\})$ evaluation of both NPS-sets for $p = 40\%$

n	m	**Pugazhendhi et al. (2002)**			**New heuristic procedure**		
		Mean RPI_1	Max RPI_1	N_1	Mean RPI_2	Max RPI_2	N_2
10	10	0,74	6,63	2	0,81	2,48	**2**
	20	0,32	3,26	4	1,51	6,24	**5**
	30	1,14	9,63	6	2,07	7,43	**6**
	40	1,55	4,28	9	2,16	6,11	**9**
	50	1,22	0,08	6	1,36	4,12	**6**
20	10	0,59	1,02	3	0,39	3,24	**4**
	20	1,05	3,76	8	1,06	2,95	7
	30	1,35	1,09	7	1,48	5,83	**9**
	40	0,10	3,35	9	1,62	3,96	**9**
	50	0,63	6,51	10	2,39	6,21	8
30	10	0,25	1,34	3	0,15	1,23	2
	20	0,30	3,08	8	0,83	2,33	**9**
	30	0,70	2,01	10	1,33	3,15	**10**
	40	1,04	2,38	10	1,74	3,26	9
	50	0,28	0,97	10	1,88	4,79	**10**
40	10	0,08	1,49	3	0,16	1,34	2
	20	0,96	2,14	8	0,84	1,48	**8**
	30	0,97	3,07	8	1,36	4,09	**9**
	40	1,01	2,92	10	1,30	4,57	9
	50	1,12	3,14	9	1,98	4,42	**10**

Table 5.11: Performance $(M = \max\{C_i\})$ evaluation of both NPS-sets for $p = 50\%$

n	m	**Pugazhendhi et al. (2002)**			**New heuristic procedure**		
		Mean RPI_1	Max RPI_1	N_1	Mean RPI_2	Max RPI_2	N_2
10	10	0,89	3,71	3	0,81	1,48	**4**
	20	1,43	2,96	4	1,51	3,84	**5**
	30	2,40	5,72	9	2,07	4,72	7
	40	2,34	4,68	10	2,16	5,11	**10**
	50	1,43	3,26	10	1,36	3,12	8
20	10	0,29	0,98	4	0,39	1,24	**4**
	20	1,26	3,71	8	1,06	2,95	**9**
	30	1,82	4,97	9	1,48	4,83	**9**
	40	1,53	4,26	7	1,62	4,96	**8**
	50	2,52	5,32	10	2,39	5,21	9
30	10	0,39	1,03	3	0,15	1,23	**4**
	20	0,95	2,10	10	0,83	2,33	9
	30	1,26	2,32	10	1,33	5,15	9
	40	2,10	2,54	6	1,74	3,26	**6**
	50	1,97	4,11	9	1,88	4,79	**10**
40	10	0,32	1,01	3	0,16	1,34	2
	20	0,82	2,32	9	0,84	2,48	**10**
	30	1,56	4,35	10	1,36	3,09	9
	40	1,31	4,59	10	1,30	3,57	**10**
	50	2,15	5,12	10	1,98	4,42	9

The computational results gained from each run per different values of n, m and p based on makespan criteria provide important insights. The discussions upon the results can me summarized as follows:

a) both heuristics start to dominate the NEH heuristic in terms of makespan as the performance criterion, when the percentage of missing operations is increased, namely over 20%; implying that for the flowshops with percentage of missing operations less than 20%, there is no significant need for obtaining NPS due to makespan as the primary performance criterion, parallel to what has been drawn out by Liao et al. (2006) in their very recent study,

b) increasing the number of jobs, n, generally results in a lessened percentage improvement (in terms of makespan) by both heuristic procedures compared to NEH heuristic; implying that the need for obtaining NPS does not increase for a flowshop as the number of jobs increase,

c) increasing the number of machines, m, while keeping the number of jobs constant, generally results in a higher percentage improvement in terms of the makespan as the primary criterion; implying that more number of machines with higher percentage of missing operations leads to higher need for obtaining NPS for a flowshop,

d) when the proposed heuristic is compared to the heuristic method developed by Pugazhendhi et al. (2002) in terms of the Mean RPI, and Maximum RPI for each problem instance, it has been observed that the newly proposed approach does not provide clear dominance on the existing NPS-generating heuristic developed by Pugazhendhi et al. (2002), even for different levels of percentage of missing operations; implying that the two heuristics perform more or less the same in terms of makespan as the primary performance criterion (see table 5.16).

Similar comparisons have been made for flowtime as the secondary performance criterion, in order to assess the outcomes obtained by the introduction of the new heuristic approach.

5.1.3.2. TOTAL FLOWTIME AS THE SECONDARY CRITERION

Tables 5.12 through 5.15 illustrate the relative performance of the two proposed heuristics by taking Ho's heuristic as the base in terms of getting the best permutation schedule due to "total flowtime" as the performance criterion.

Table 5.12: Performance $(F = \sum C_i)$ evaluation of both NPS-sets for $p = 20\%$

n	m	**Pugazhendhi et al. (2002)**			**New heuristic procedure**		
		Mean RPI_1	Max RPI_1	N_1	Mean RPI_2	Max RPI_2	N_2
10	10	0,35	3,87	4	0,37	3,05	**5***
	20	0,94	2,50	5	0,74	2,98	**4**
	30	0,69	3,93	6	0,37	3,45	**6**
	40	0,66	3,75	7	0,50	3,32	**6**
	50	0,62	2,86	8	0,35	2,10	**8**
20	10	0,18	2,53	4	0,25	2,04	**6**
	20	0,73	3,75	8	0,64	2,89	**8**
	30	0,90	3,93	8	0,66	2,72	**10**
	40	0,65	3,09	8	0,94	4,62	**10**
	50	0,86	2,91	10	0,80	4,03	**10**
30	10	0,32	2,50	8	0,55	2,55	**8**
	20	0,55	2,55	7	0,76	2,73	**7**
	30	0,37	3,58	9	0,67	1,53	**10**
	40	0,58	2,34	10	0,72	3,52	**10**
	50	0,78	1,67	8	0,95	2,29	**9**
40	10	0,11	1,16	8	0,52	1,09	**9**
	20	0,56	3,26	10	0,77	1,86	**10**
	30	0,70	3,02	10	0,88	3,43	**10**
	40	0,63	2,97	10	0,45	2,53	**10**
	50	0,82	1,61	10	0,74	3,00	**10**

* For $N_2 \geq N_1$, values in the column are highlighted.

Table 5.13: Performance $(F = \sum C_i)$ evaluation of both NPS-sets for $p = 30\%$

n	m	**Pugazhendhi et al. (2002)**			**New heuristic procedure**		
		Mean RPI_1	Max RPI_1	N_1	Mean RPI_2	Max RPI_2	N_2
10	10	0,61	7,82	3	0,43	9,46	**4***
	20	0,98	5,30	6	1,37	5,86	**7**
	30	1,27	5,40	9	1,21	4,46	8
	40	1,17	6,24	9	1,17	5,69	**9**
	50	1,09	4,89	8	1,18	5,68	**8**
20	10	0,67	4,24	8	0,98	5,28	**8**
	20	1,01	5,19	8	1,02	6,27	**10**
	30	1,19	5,00	10	1,33	6,45	**10**
	40	1,27	5,08	9	1,45	5,18	**9**
	50	1,52	5,15	9	1,80	8,08	**10**
30	10	0,62	1,43	9	0,31	1,86	**9**
	20	1,26	4,89	10	0,95	5,29	**10**
	30	1,56	3,51	9	1,25	5,38	**10**
	40	1,44	6,28	10	1,24	4,11	9
	50	1,06	6,49	9	1,39	6,76	**10**
40	10	0,49	3,51	10	0,57	4,19	**10**
	20	0,94	4,82	10	0,82	5,39	**10**
	30	1,56	3,85	9	1,28	6,42	**10**
	40	1,27	5,53	9	1,25	6,70	**9**
	50	1,44	5,38	9	1,52	6,33	**9**

Table 5.14: Performance $(F = \sum C_i)$ evaluation of both NPS-sets for $p = 40\%$

n	m	**Pugazhendhi et al. (2002)**			**New heuristic procedure**		
		Mean RPI_1	Max RPI_1	N_1	Mean RPI_2	Max RPI_2	N_2
10	10	0,61	3,21	3	0,67	3,38	**4***
	20	1,52	8,10	8	1,55	5,25	**8**
	30	1,95	5,76	8	1,97	5,07	**9**
	40	1,89	4,70	8	1,71	4,58	**9**
	50	1,56	5,34	9	1,68	7,72	**8**
20	10	0,56	2,19	5	0,44	3,00	**7**
	20	1,44	4,13	9	1,59	4,10	**10**
	30	1,68	6,02	10	1,99	8,05	**10**
	40	2,16	5,57	10	2,27	7,81	**10**
	50	2,45	7,47	9	2,59	6,76	**10**
30	10	0,51	2,43	7	0,47	2,26	**8**
	20	1,89	4,20	9	1,81	7,37	**10**
	30	2,43	5,72	9	2,24	9,71	**10**
	40	2,25	5,36	10	2,08	5,78	**10**
	50	2,33	5,39	10	1,92	9,27	**10**
40	10	0,55	1,55	9	0,75	2,88	**10**
	20	1,84	3,68	9	1,83	5,66	**10**
	30	2,10	4,46	10	2,32	7,21	**10**
	40	2,46	5,34	9	2,30	7,77	**10**
	50	2,01	4,03	10	2,14	5,82	**10**

Table 5.15: Performance $(F = \sum C_i)$ evaluation of both NPS-sets for $p = 50\%$

n	m	**Pugazhendhi et al. (2002)**			**New heuristic procedure**		
		Mean RPI_1	Max RPI_1	N_1	Mean RPI_2	Max RPI_2	N_2
10	10	1,03	4,26	6	0,67	3,35	**8***
	20	1,73	5,89	7	1,93	4,14	**8**
	30	2,29	6,42	10	2,19	4,84	9
	40	1,90	4,10	10	2,45	5,96	**10**
	50	1,88	6,98	9	1,93	6,72	**10**
20	10	1,08	4,03	8	1,19	7,18	**9**
	20	1,36	4,45	9	1,47	5,80	**10**
	30	2,37	5,35	10	2,32	8,41	**10**
	40	2,23	5,98	10	2,24	6,47	**10**
	50	2,41	8,10	10	2,68	7,89	**10**
30	10	0,45	3,45	8	0,15	3,01	**10**
	20	1,48	4,09	10	1,60	4,23	**10**
	30	2,55	5,22	9	2,44	5,23	**10**
	40	2,40	7,51	10	2,53	6,99	**10**
	50	1,96	3,90	10	1,99	3,83	**10**
40	10	0,80	3,33	10	0,54	3,38	**10**
	20	1,96	6,54	10	1,71	4,01	**10**
	30	2,45	8,52	10	2,15	8,79	**10**
	40	2,39	7,21	10	2,16	7,27	**10**
	50	2,40	6,88	10	2,41	6,38	**10**

The computational results gained from each run per different values of n, m and p based on makespan demonstrate significant improvement due to flowtime as the secondary performance criterion, compared to makespan as the primary performance indicator. The discussions upon the results can me summarized as follows:

a) increasing p values more than %20 leads to both NPS-generating heuristics dominating NEH heuristic in terms of flowtime as the secondary performance criterion; implying that both NPS-generating heuristics, especially the proposed heuristic provides much better results due to flowtime as the secondary performance criterion for flowshops compared to permutation schedules,

b) increasing number of jobs, n does not significantly affect the percentage improvements made by both heuristics, in other words there is no correlation in between the number of jobs and the percentage improvement by both heuristic approaches, based on the flowtime criteria; however, increasing number of machines, m leads to higher percentage improvement by both heuristic methods in terms of developing NPS for flowshops based on flowtime criteria,

c) when the proposed heuristic is compared to the heuristic method developed by Pugazhendhi et al. (2002) in terms of the Mean RPI, and Maximum RPI for each problem instance, it has been observed that the newly proposed approach does provide considerable improvement for each of the problem instances; implying that the percentage improvement by introduction of the new NPS-generating heuristic is clear by increasing percentage of missing operations.

The summary of section 5.1.3.1 and section 5.1.3.2 can be made with a simple table (Table 5.16) in order to assess the overall performance of the heuristic developed by Pugazhendhi et al. (2002) and the modification of it allowing multiple job-passing at a time. Table 5.16 includes the sum of the values of N_1 and N_2's for each distinct value of p. It can be seen that for the case of flowshop problems with p values lower than 20%, the new heuristic as well as the heuristic of Pugazhendhi do not improve the makespan better than NEH heuristic does. However, the performance

enhancement by these heuristics is significant for p values greater than or equal to 30%.

Table 5.16: Comparison of domination of heuristics over NEH*

	value of p	**New heuristic over NEH**	**Pugazhendhi over NEH**
1st Criterion:	20%	104/96	**107/93****
	30%	132/68	**140/60**
Makespan	40%	142/58	**145/55**
	50%	151/49	**153/47**
	value of p	**New heuristic over Ho's heuristic**	**Pugazhendhi over Ho's heuristic**
2nd Criterion:	20%	**166/34**	158/42
	30%	**179/21**	173/27
Flowtime	40%	**183/17**	171/29
	50%	**194/6**	186/14

* Over 200 instances for each value of p.

** Bold values indicate the dominance of that approach over other two methods.

$ # of instances the heuristic dominates / # of instances NEH or Ho's heuristic dominates.

For the case of total flowtime, the new heuristic outperforms Ho's heuristic (especially for higher values of p) also dominating the heuristic developed by Pugazhendhi et al. with significant improvement, which is getting larger as the percentage of missing operations is increased.

5.1.3.3. COMPARISON OF CPU TIMES

The mean CPU time taken for implementing the heuristics that generate the permutation schedules (namely NEH and Ho's heuristic) as well as for implementing the new heuristic also has been tracked and analyzed for each specific problem instance corresponding to p taking values 20% and 30% respectively (as number of missing operations increase, the CPU time will decrease for any algorithm) without a need to consider the instances for p = 40% and 50%. Table 5.17 shows that the total

computational for implementing the proposed heuristic is negligible. Actually what the proposed heuristic does is a simple sequencing rule with at most $O(nm + n\log n)$ complexity. Compared to the NEH heuristic with complexity $O(n^3 m)$ the new heuristic has a theoretical complexity of $O((nm + n\log n)^3 m) = O(n^3 m^4)$ with the sequencing rule inserted into it (remaining as a polynomial time algorithm).

Table 5.17: Comparison of CPU times* of both heuristics for each problem instance

n	m	p = 20%			p = 30%		
		NEH	Pugaz et al.	New Heuristic	NEH	Pugaz et al.	New Heuristic
10	10	0,313	0,043	1,253	0,234	0,036	0,921
	20	0,375	0,017	2,207	0,286	0,017	2,039
	30	0,387	0,047	0,797	0,391	0,047	0,748
	40	0,419	0,039	0,978	0,485	0,032	0,729
	50	0,472	0,026	0,092	0,379	0,025	0,074
20	10	0,385	0,046	0,777	0,384	0,043	0,595
	20	0,446	0,008	0,845	0,474	0,007	0,681
	30	0,461	0,005	3,029	0,390	0,005	2,905
	40	0,524	0,004	2,485	0,522	0,004	2,206
	50	0,548	0,031	2,912	0,506	0,026	2,879
30	10	0,817	0,121	4,129	0,886	0,098	3,738
	20	0,822	0,065	2,778	0,733	0,055	2,003
	30	0,831	0,095	1,164	0,861	0,077	1,145
	40	0,963	0,096	5,377	1,023	0,091	5,031
	50	1,006	0,131	5,927	0,921	0,127	4,421
40	10	1,290	0,168	9,365	1,274	0,139	6,847
	20	1,529	0,219	4,613	1,575	0,197	4,170
	30	1,601	0,095	4,220	1,647	0,093	3,063
	40	1,849	0,108	8,111	1,753	0,101	7,098
	50	1,994	0,203	10,970	2,038	0,190	8,986

* Time unit is seconds.

CHAPTER 6

CONCLUSIONS

In this study, a new heuristic procedure for flowshop problems has been proposed, with the intention to generate non-permutation schedules. The makespan and flowtime as the primary and secondary criterion of performance respectively have been considered with the newly proposed heuristic method.

The new heuristic based on a primal version selected from the literature imposed good results for flowshops especially when the number of missing operations is relatively high. For the problem instances where the percentage of missing operations is lower than 20%, the new approach as well as the existing heuristic procedure of Pugazhendhi et al. (2002) does not improve the performance of the flowshop schedule thinking in terms of the "makespan" as the primary criterion. The permutation schedules generated by NEH heuristic for each example provide better makespan values up to 20% for percentage of missing operations. Over 20%, the newly proposed heuristic improves the makespan value compared to NEH heuristic, however the improvement is not higher than that of Pugazhendhi et al.'s heuristic.

For the case of the secondary performance criterion, namely the total flowtime of jobs; the effects of the number of jobs, the number of machines, and the overall percentage of missing operations have also been analyzed. Basically, evident results have been gathered showing that with increasing percentage of missing operations and/or increasing number of jobs and/or increasing number of machines in a flowshop, the proposed heuristic outperforms all other permutation and non-permutation schedule generating heuristics. The performance of the new heuristic procedure having intermediary sorting rules for multiple jobs bypassing the stages at a time was compared with the significant methods selected from the literature in

terms of solution performance. According to the outcomes of this thesis study and interpretations made upon those outcomes, the suggestions for further research directions might be as follows:

a. The outcomes of this thesis study clearly demonstrate that there is room for improvement in terms of flowtime as the criterion of performance. Moreover, the computational results also show that even for low percentage of missing operations, non-permutation schedules provide better results. One has to analyze the percentage improvement provided by introduction of NPS for problems having percentage of missing operations less than 15% down to 0%. This task has been partially performed by Liao et al. (2006), however much clear evaluation is needed.

b. Heuristic procedures that are capable of solving problems of larger sizes, presumably with the use of metaheuristics especially for large problems, have to be developed in order to obtain higher percentage improvement by bringing non-permutation schedules into the scene.

c. Further studies have to look for simple procedures or heuristics which are capable of generating non-permutation schedules by a similar approach with the newly proposed heuristic method, providing good performance in terms of makespan as the primary criterion, for the case of flowshops with missing operations. Other due date based criteria might be brought into the discussion for flowshops with missing operations, where the need for obtaining NPS is much more significant (Liao et al., 2006).

d. More extensive use of dispatching rules at the intermediary steps for allowing job-passing might produce effective methodologies for generation of non-permutation schedules based on various performance criteria.

REFERENCES

Baker, K.R. (1995). Introduction to Sequencing and Scheduling. John Wiley & Sons, New York.

Bonney, M. and Gundry, S. (1976) "Solutions to the Constrained Flowshop Sequencing Problem," *Operational Research Quarterly* 27 (4), 869-883.

Campbell, H.G., Dudek, R.A., and Smith, M.L. (1970) "A Heuristic Algorithm for the n-Job, m-Machine Sequencing Problem," *Management Science* 16, 630-637.

Chen, C.-L, Vempati, V.S., and Aljaber, N. (1995) "An Application of Genetic Algorithms for Flow Shop Problems," *European Journal of Operational Research* 80, 389-396.

Conway, R.W., Maxwell, W.L., and Miller, L.W. (1967) *Theory of Scheduling.* Addison-Wesley, Reading, MA.

Dannenbring, D. (1977) "An Evaluation of Flow Shop Sequencing Heuristics," *Management Science* 23, 1174-1182.

Davoud Pour, H. (2001) "A New Heuristic for the n-Job, m-Machine Flow-shop Problem," *Production Planning and Control* 12 (7), 648-653.

Dumolien, W.J., and Santen, W.P. (1983) "Cellular Manufacturing Becomes Philosophy of Management at Component Facility," *Industrial Engineering* 34, 72-76.

Framinan, J.M., Leisten, R., Rajendran, C. (2003) "Different Initial Sequences for the Heuristic of Nawaz, Enscore, and Ham to Minimize Makespan, Idle-time or Flowtime in the Static Permutation Flowshop Sequencing Problem," *International Journal of Production Research* 41 (1), 121-148.

Graves, S.C. (1981) "A Review of Production Scheduling," *Operations Research* 29, 646-675.

Gupta, J.N. (1971) "A Functional Heuristic Algorithm for the Flowshop Scheduling Problem," *Operational Research Quarterly* 22 (1), 39-47.

Gupta, J.N.D. (1972) "Heuristic Algorithms for Multistage Flowshop Scheduling Problem," *AIEE Transactions* 4 (1), 11-18.

Gupta, J.N.D., Tseng, F.T., Stafford, E.F., (2004) "An Empirical Analysis of Integer Programming Formulations for Permutation Flowshop," *OMEGA, The International Journal of Management Science* 32, 285-293.

Ho, J.C., Chang, Y.-L. (1991) "A New Heuristic for the n-Job, M-Machine Flow-Shop Problem," *European Journal of Operational Research* 52, 194-202.

Ho, J.C. (1995) "Flowshop Sequencing with Mean Flowtime Objective," *European Journal of Operational Research* 81, 571-578.

Hundal, T.S. and Rajpogal, J. (1988) "An Extension of Palmer's Heuristic for the Flow Shop Scheduling Problem," *International Journal of Production Research* 26 (6), 1119-1124.

Ishibuchi, H., Misaki, S. and Tanaka, H. (1995) "A Modified Simulated Annealing Algorithm for the Flowshop Sequencing Problem," *European Journal of Operational Research* 81, 388-398.

Johnson, S.M. (1954) "Optimal Two-and Three-Stage Production Schedules with Setup Times Included", *Naval Research Logistics Quarterly* 1, 61-68.

King, J.R. and Spachis, A.S. (1980) "Heuristics for Flow-Shop Scheduling," *International Journal of Production Research* 18 (3), 345-357.

Koulamas, C. (1998) "A New Constructive Heuristic for the Flow-Shop Scheduling Problem," *European Journal of Operational Research* 105, 66-71.

Lahiri, S., Rajendran, C., and Narendan, T.T. (1993) "Evaluation of Heuristics for

Scheduling in a Flowshop: A Case Study," *Production Planning and Control* 4, 153-158.

Leisten, R. and Kolbe, M. (1998) "A Note on Scheduling Jobs with Missing Operations in Permutation Flowshops," *International Journal of Production Research* 36 (9), 2627-2630.

Miyazaki, S. and Nishiyama, N., and Hashimoto, F. (1978) "An Adjacent Pairwise Approach to the Mean Flowtime Scheduling Problem," *Journal of the Operations Research Society of Japan* 21, 287-299.

Murata, T., Ishibuchi, H., and Tanaka, H. (1996) "Genetic Algorithms for Flowshop Scheduling Problems," *Computers and Industrial Engineering* 30 (4), 1061-1071.

Nawaz, M., Enscore, E., and Ham, I. (1983) "A Heuristic Algorithm for the m -Machine, n -Job Flow-shop Sequencing Problem," *OMEGA, The International Journal of Management Science* 11, 91-95.

Ogbu, F. and Smith, D. (1990) "The Application of the Simulated Annealing Algorithms to the Solution of the $n/m/C_{\max}$ Flowshop Problem," *Computers and Operations Research* 17 (3), 243-253.

Osman, I. and Potts, C.N. (1989) "Simulated Annealing for Permutation Flow-Shop Scheduling," *OMEGA, The International Journal of Management Science* 17, 551-557.

Page, E.S. (1961) "An Approach to the Scheduling of Jobs on Machines," *Journal of the Royal Statistical Society, B Series* 23 (2), 484-492.

Palmer, D. S. (1965) "Sequencing Jobs Through a Multi-Stage Process in the Minimum Flow Time – A Quick Method of Obtaining a Near Optimum," *Operational Research Quarterly* 16, 101-107.

Park, Y., Pegden, C.D., and Enscore, E.E. (1984) "A Survey and Evaluation of Static Flowshop Scheduling Heuristics," *International Journal of Production Research* 22, 127-141.

Pinedo, M. (2002) Scheduling: *Theory, Algorithms and Systems*. Second Ed. Prentice-Hall, Englewood Cliffs, NJ.

Ponnambalam, S.G., Aravindan, P., and Chandrasekaran, S. (2001) "Constructive and Improvement Flow Shop Scheduling Heuristics: An Extensive Evaluation," *Production Planning and Control* 12 (4), 335-344.

Potts, C.N., Shmoys D.B., and Williamson D.P. (1991) "Permutation vs. Non-permutation Flow Shop Schedules," *Operations Research Letters* 10, 281-284.

Pugazhendhi, S., Thiagarajan, S., Rajendran, C., and Anantharaman, N. (2002) "Performance Enhancement by Using Non-permutation Schedules in Flowline-based Manufacturing Systems," *Computers and Industrial Engineering* 44, 133-157.

Pugazhendhi, S., Thiagarajan, S., Rajendran, C., and Anantharaman, N. (2004) "Relative Performance Evaluation of Permutation and Non-Permutation Schedules in Flowline-Based Manufacturing Systems with Flowtime Objective," *International Journal of Advanced Manufacturing Technology* 23, 820-830.

Rajendran, C. (1993) "Heuristic algorithm for scheduling in a flowshop to minimize total flow time," *International Journal of Production Economics* 29, 65-73.

Rajendran, C. and Ziegler, H. (2001) "A Performance Analysis of Dispatching Rules and a Heuristic in static Flowshops with Missing Operations of Jobs" *European Journal of Operational Research* 131, 622-634.

Reeves, C.R. (1993) "Improving the Efficiency of Tabu Search for Machine Sequencing Problems," *Journal of the Operational Research Society* 44, 375-382.

Reeves, C.R. (1995) "A Genetic Algorithm for Flow-Shop Sequencing," *Computers and Operations Research* 22, 5-13.

Ruiz, R. and Maroto, C. (2005) "A Comprehensive Review and Evaluation of Permutation Flowshop Heuristics," *European Journal of Operational Research* 165, 479-494.

Sridhar, J. and Rajendran, C. (1993) “Scheduling in a Cellular Manufacturing System: a Simulated Annealing Approach,” *International Journal of Production Research* 31, 2927-2945.

Suliman, S. (2000) “A Two-phase Heuristic to the Permutation Flow Shop Scheduling Problem,” *International Journal of Production Economics* 64, 143-152.

Taillard, E. (1990) “Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem,” *European Journal of Operations Research* 47, 65-74.

Taillard, E. (1993) “Benchmarks for Basic Scheduling Problems,” *European Journal of Operational Research* 64, 278-285.

Tandon, E., Cummings, P.T., and LeVan, M.D. (1991) “Flowshop Sequencing with Non-Permutation Schedules,” *Computers & Chemical Engineering* 15, 601-607

Wagner, H.M. (1959) “An Integer-Linear Programming Model for Machine Scheduling,” *Naval Research Logistics Quarterly* 6, 131-140.

Wang, L., and Zheng, D.Z. (2003) “An Effective Hybrid Heuristic for Flow Shop Scheduling,” *The International Journal of Advanced Manufacturing Technology* 21, 38-44.

Widmer, M. and Hertz, A. (1989) “A New Heuristic Method for the Flow Shop Sequencing Problem,” *European Journal of Operational Research* 41, 186-193.

Zegordi, S.H., Itoh, K., and Enkawa, T. (1995) “Minimizing Makespan for Flowshop Scheduling by Combining Simulated Annealing with Sequencing Knowledge,” *European Journal of Operational Research* 85, 515-531.

APPENDICES

APPENDIX A: C++ CODE FOR GENERATING EXAMPLE PROBLEMS

```
/*
     C++ code generating the 200 flow shop instances for each
     p value (p: percentage of missing operations).

     Processing times vary within the interval {0,99}.

*/

#define ANSI_C 0     /* 0:   K&R function style convention */
#define VERIFY 0     /* 1:   produce the verification file */
#define FIRMACIND 0  /* 0,1: first machine index           */

#include <stdio.h>
#include <math.h>

struct problem {
  long rand_time;      /* random seed for jobs */
  short num_jobs;      /* number of jobs */
  short num_mach;      /* number of machines */
};

#if VERIFY == 1

struct problem S[] = {
  {         0,  0, 0},
  { 873654221, 10, 10},
  {         0,  0, 0}};

#else /* VERIFY */

struct problem S[] = {
{         0,     0,  0},
                    /* 10 jobs  10 machines */
{ 873654221,    10,  10},
{ 379008056,    10,  10},
{ 1866992158,   10,  10},
{ 216771124,    10,  10},
{ 495070989,    10,  10},
{ 402959317,    10,  10},
{ 1369363414,   10,  10},
{ 2021925980,   10,  10},
{ 573109518,    10,  10},
```

```
{ 88325120,     10,  10},
                /* 20 jobs  10 machines */
{ 587595453,    20, 10},
{ 1401007982,   20, 10},
{ 873136276,    20, 10},
{ 268827376,    20, 10},
{ 1634173168,   20, 10},
{ 691823909,    20, 10},
{ 73807235,     20, 10},
{ 1273398721,   20, 10},
{ 2065119309,   20, 10},
{ 1672900551,   20, 10},
                /* 30 jobs 10 machines */
{ 479340445,    30, 10},
{ 268827376,    30, 10},
{ 1958948863,   30, 10},
{ 918272953,    30, 10},
{ 555010963,    30, 10},
{ 2010851491,   30, 10},
{ 1519833303,   30, 10},
{ 1748670931,   30, 10},
{ 1923497586,   30, 10},
{ 1829909967,   30, 10},
                /* 40 jobs  10 machines */
{ 1328042058,   50,  5},
{ 200382020,    50,  5},
{ 496319842,    50,  5},
{ 1203030903,   50,  5},
{ 1730708564,   50,  5},
{ 450926852,    50,  5},
{ 1303135678,   50,  5},
{ 1273398721,   50,  5},
{ 587288402,    50,  5},
{ 248421594,    50,  5},
                /* 10 Jobs 20 machines */
{ 1958948863,   10, 20},
{ 575633267,    10, 20},
{ 655816003,    10, 20},
{ 1977864101,   10, 20},
{ 93805469,     10, 20},
{ 1803345551,   10, 20},
{ 49612559,     10, 20},
{ 1899802599,   10, 20},
{ 2013025619,   10, 20},
{ 578962478,    10, 20},
                /* 20 jobs 20 machines */
{ 1539989115,   20, 20},
{ 691823909,    20, 20},
{ 655816003,    20, 20},
{ 1315102446,   20, 20},
{ 1949668355,   20, 20},
```

```
{ 1923497586,   20, 20},
{ 1805594913,   20, 20},
{ 1861070898,   20, 20},
{ 715643788,    20, 20},
{ 464843328,    20, 20},
                /* 30 jobs  20 machines */
{ 896678084,    30, 20},
{ 1179439976,   30, 20},
{ 1122278347,   30, 20},
{ 416756875,    30, 20},
{ 267829958,    30, 20},
{ 1835213917,   30, 20},
{ 1328833962,   30, 20},
{ 1418570761,   30, 20},
{ 161033112,    30, 20},
{ 304212574,    30, 20},
                /* 40 jobs 20 machines */
{ 1539989115,   40, 20},
{ 655816003,    40, 20},
{ 960914243,    40, 20},
{ 1915696806,   40, 20},
{ 2013025619,   40, 20},
{ 1168140026,   40, 20},
{ 1923497586,   40, 20},
{ 167698528,    40, 20},
{ 1528387973,   40, 20},
{ 993794175,    40, 20},
                /* 10 jobs 30 machines */
{ 450926852,    10, 30},
{ 1462772409,   10, 30},
{ 1021685265,   10, 30},
{ 83696007,     10, 30},
{ 508154254,    10, 30},
{ 1861070898,   10, 30},
{ 26482542,     10, 30},
{ 444956424,    10, 30},
{ 2115448041,   10, 30},
{ 118254244,    10, 30},
                /* 20 jobs 30 machines */
{ 471503978,    20, 30},
{ 1215892992,   20, 30},
{ 135346136,    20, 30},
{ 1602504050,   20, 30},
{ 160037322,    20, 30},
{ 551454346,    20, 30},
{ 519485142,    20, 30},
{ 383947510,    20, 30},
{ 1968171878,   20, 30},
{ 540872513,    20, 30},
                /* 30 jobs 30 machines */
{ 2013025619,   30, 30},
```

```
{ 475051709,    30, 30},
{ 914834335,    30, 30},
{ 810642687,    30, 30},
{ 1019331795,   30, 30},
{ 2056065863,   30, 30},
{ 1342855162,   30, 30},
{ 1325809384,   30, 30},
{ 1988803007,   30, 30},
{ 765656702,    30, 30},
                /* 40 jobs 30 machines */
{ 1368624604,   40, 30},
{ 450181436,    40, 30},
{ 1927888393,   40, 30},
{ 1759567256,   40, 30},
{ 606425239,    40, 30},
{ 19268348,     40, 30},
{ 1298201670,   40, 30},
{ 2041736264,   40, 30},
{ 379756761,    40, 30},
{ 28837162,     40, 30},
                /* 10 jobs 40 machines */
{ 450926852,    10, 40},
{ 1462772409,   10, 40},
{ 1021685265,   10, 40},
{ 83696007,     10, 40},
{ 508154254,    10, 40},
{ 1861070898,   10, 40},
{ 26482542,     10, 40},
{ 444956424,    10, 40},
{ 2115448041,   10, 40},
{ 118254244,    10, 40},
                /* 20 jobs 40 machines */
{ 471503978,    20, 40},
{ 1215892992,   20, 40},
{ 135346136,    20, 40},
{ 1602504050,   20, 40},
{ 160037322,    20, 40},
{ 551454346,    20, 40},
{ 519485142,    20, 40},
{ 383947510,    20, 40},
{ 1968171878,   20, 40},
{ 540872513,    20, 40},
                /* 30 jobs 40 machines */
{ 2013025619,   30, 40},
{ 475051709,    30, 40},
{ 914834335,    30, 40},
{ 810642687,    30, 40},
{ 1019331795,   30, 40},
{ 2056065863,   30, 40},
{ 1342855162,   30, 40},
{ 1325809384,   30, 40},
```

```
{ 1988803007,   30, 40},
{ 765656702,    30, 40},
                /* 40 jobs 40 machines */
{ 1368624604,   40, 40},
{ 450181436,    40, 40},
{ 1927888393,   40, 40},
{ 1759567256,   40, 40},
{ 606425239,    40, 40},
{ 19268348,     40, 40},
{ 1298201670,   40, 40},
{ 2041736264,   40, 40},
{ 379756761,    40, 40},
{ 28837162,     40, 40},

                /* 10 jobs 50 machines */
{ 450926852,    10, 50},
{ 1462772409,   10, 50},
{ 1021685265,   10, 50},
{ 83696007,     10, 50},
{ 508154254,    10, 50},
{ 1861070898,   10, 50},
{ 26482542,     10, 50},
{ 444956424,    10, 50},
{ 2115448041,   10, 50},
{ 118254244,    10, 50},
                /* 20 jobs 50 machines */
{ 471503978,    20, 50},
{ 1215892992,   20, 50},
{ 135346136,    20, 50},
{ 1602504050,   20, 50},
{ 160037322,    20, 50},
{ 551454346,    20, 50},
{ 519485142,    20, 50},
{ 383947510,    20, 50},
{ 1968171878,   20, 50},
{ 540872513,    20, 50},
                /* 30 jobs 50 machines */
{ 2013025619,   30, 50},
{ 475051709,    30, 50},
{ 914834335,    30, 50},
{ 810642687,    30, 50},
{ 1019331795,   30, 50},
{ 2056065863,   30, 50},
{ 1342855162,   30, 50},
{ 1325809384,   30, 50},
{ 1988803007,   30, 50},
{ 765656702,    30, 50},
                /* 40 jobs 50 machines */
{ 1368624604,   40, 50},
{ 450181436,    40, 50},
{ 1927888393,   40, 50},
```

```
{ 1759567256,   40, 50},
{ 606425239,    40, 50},
{ 19268348,     40, 50},
{ 1298201670,   40, 50},
{ 2041736264,   40, 50},
{ 379756761,    40, 50},
{ 28837162,     40, 50},

{          0,    0,  0}};
#endif /* VERIFY */

/* generate a random number uniformly between low and high */
int z=0;

#if ANSI_C == 1
int unif (long *seed, short low, short high)
#else
short unif (long *seed, short low, short high)
//long *seed; short low, high;
#endif
{
 static long m = 2147483647, a = 16807, b = 127773, c = 2836;
 double  value_0_1;

 long k = *seed / b;
 *seed = a * (*seed % b) - k * c;
 if(*seed < 0) *seed = *seed + m;
 value_0_1 =  *seed / (double) m;


 return (short) (low + floor(value_0_1 * (high - low + 1)));

}

/* Maximal 40 jobs and 50 machines are provided. */

short d[21][501];                   /* duration */

#if ANSI_C == 1
void generate_flow_shop(short p)         /* Fill d and M according to S[p] */
#else
void generate_flow_shop(short p)
//short p;
#endif
{
 short i, j;
 long time_seed = S[p].rand_time;

 for(i = 0; i < S[p].num_mach; ++i)      /* determine a random duration */
   for (j = 0; j < S[p].num_jobs; ++j)   /* for all operations */
     d[i][j] = unif(&time_seed, 0, 99);  /* 99 = max. duration of op. */
```

```
}

#if ANSI_C == 1
void write_problem(short p)  /* write out problem */
#else
void write_problem(short p)
//short p;
#endif
{
 short i, j;
 FILE *f = NULL;
 char name[6];

 sprintf(name,"p%03d", p);              /* file name construction */
 if(!(f = fopen(name,"w"))) {           /* open file for writing  */
  fprintf(stderr,"file %s error\n", name);
  return;
 }
 fprintf(f,"%d %d\n", S[p].num_jobs, S[p].num_mach); /* write header line */

 for(j = 0; j < S[p].num_jobs; ++j) {
  for(i = 0; i < S[p].num_mach; ++i) {
    fprintf(f,"%2d %2d ", i+FIRMACIND, d[i][j]);   /* write machine and job */
  }
  fprintf(f,"\n");                      /* newline == End of job */
 }
 fclose(f);                             /* close file */
}

int main()
{
 short i = 1;
 while(S[i].rand_time) {                /* for i == 1 up to NULL entry */
  generate_flow_shop(i);                /* generate problem i  */
  write_problem(i);                     /* write out problem i */
  ++i;                                  /* increment i */
 }
 return 0;
}
```

APPENDIX B: C++ CODE FOR GENERATING NPS WITH THE NEW APPROACH

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>

// generating NPS with the makespan criterion
// maximal size of problem: 50 machines, 40 jobs
// current size of problem: m machines, n jobs
// structure of pi[] is the same as for pibest[]
// processing times p[1..m][1..n]; cells p[0][..] and p[..][0] are not used
// completion times C[1..n][1..m] are the same as p[][]; cells C[0][..] and C[..][0] are set to
zero

#define N    50                    // maximal number of jobs
#define M    10                    // maximal number of machines
#define MAXL 6000000L              // max long value

int pi[N+1];
long p[M+1][N+1],C[M+1][N+1];
int pibest[N+1];

FILE *fe;                          // output file to be traced

// templates
// returns max(x,y)
template <class number>
  number max(number x, number y) { return (x>y)?x:y; }

// returns min(x,y)
template <class number>
  number min(number x, number y) { return (x<y)?x:y; }

// returns sign of number x
template <class number>
  int sign(number x) { return (x<=0)?-1:1; }

// swaps two elements
template <class vect>
  void swap(vect *a, vect *b) { vect c=*a; *a=*b; *b=c; }

// quicksort; returns permutation such that a[pi[i]]<=a[pi[i+1]] for a[n..m]
template <class vect>
  void sort(int n, int m, vect a[], int pi[])
  {  int i,j; vect x;
     if (m <= n) return;
     i=n; j=m; x=a[pi[(i+j)/2]];
```

```
    do
    {  while (a[pi[i]] < x) i++;
     while (x < a[pi[j]]) j--;
     if (i <= j) { swap(pi+i,pi+j); i++; j--; }
    }  while (i < j);
    sort(n,j,a,pi); sort(i,m,a,pi);
  }

// quicksort; returns permutation such that a[pi[i]]>=a[pi[i+1]] for a[n..m]
template <class vect>
  void _sort(int n, int m, vect a[], int pi[])
  {  int i,j; vect x;
     if (m <= n) return;
     i=n; j=m; x=a[pi[(i+j)/2]];
     do
     {  while (a[pi[i]] > x) i++;
      while (x > a[pi[j]]) j--;
      if (i <= j) { swap(pi+i,pi+j); i++; j--; }
     }  while (i < j);
     _sort(n,j,a,pi); _sort(i,m,a,pi);
  }

// returns makespan for permutation pi[]
long Cmax(int n, int m, long p[][N+1], int pi[])
{ int i,j;

  C[1][1]=p[1][pi[1]];
  for (j=2;j<=n;j++) C[1][j]=C[1][j-1]+p[1][pi[j]];
  for (i=2;i<=m;i++)
  { C[i][1]=C[i-1][1]+p[i][pi[1]];
    for (j=2;j<=n;j++) C[i][j]=max(C[i][j-1],C[i-1][j])+p[i][pi[j]];
  }
  return C[m][n];
}

// johnson's algorithm for F2//Cmax problem
void johnson(int n, long aa[], long bb[], int pi[])
{ int a=0,b=n+1,j;

  for (j=1;j<=n;j++) { if (aa[j]<=bb[j]) pi[++a]=j; else pi[--b]=j; }
  if (a>0) sort(1,a,aa,pi);
  if (b<=n) _sort(b,n,bb,pi);
}

// insertion of Ho's algorithm for F//Cmax problem
void campbell(int n, int m, long p[][N+1], int pi[])
{ int i,j,pp[N+1];
  long ax[N+1],bx[N+1],cp,cmx=MAXL;

  for (j=1;j<=n;j++) ax[j]=bx[j]=0;
  for (i=1;i<m;i++)
```

```
  { for (j=1;j<=n;j++) { ax[j]+=p[i][j]; bx[j]+=p[m-i+1][j]; }
    johnson(n,ax,bx,pp); cp=Cmax(n,m,p,pp);
    if (cp<cmx) { cmx=cp; for (j=1;j<=n;j++) pi[j]=pp[j]; }
  }
}

// insertion of gupta's algorithm for F//Cmax problem; algorithm 0
void gupta(int n, int m, long p[][N+1], int pi[N+1])
{ float eps = 0.001;
  int i,j;
  float ax[N+1],a;
  long s;

  for (j=1;j<=n;j++)
  { s=MAXL; pi[j]=j;
    for (i=1;i<m;i++) s=min(s,p[i][j]+p[i+1][j]);
    s=sign(p[1][j]-p[m][j]);
    if (!s) ax[j]=a/eps; else ax[j]=a/s;
  }
  _sort(1,n,ax,pi);
}

long alpha(int j, int k, int l)
  { int i; long s=0;  for (i=k;i<=l;i++) s+=p[i][j]; return s; }

long beta(int j, int k, int l) { return alpha(j,k+1,l+1); }

float f(int j, int k, int l)
{ long u=min(alpha(j,k,l),beta(j,k,l));
  return u?sign(alpha(j,k,l)-beta(j,k,l))/((float)u):sign(alpha(j,k,l)-beta(j,k,l))*MAXL;
}

// (NEH) Nawaz, Enscore and Ham's algorithm for F//Cmax
// problem, complexity O(n^3*m)
void nawaz(int n, int m, long p[][N+1], int pi[])
{ int i,j,k,t;
  long c,cp,s;
  long sump[N+1];

  for (j=1;j<=n;j++) { s=0; for (i=1;i<=m;i++) s+=p[i][j]; sump[j]=s; }
  for (j=1;j<=n;j++) pi[j]=j; _sort(1,n,sump,pi);

  for (k=2;k<n;k++)
  { cp=Cmax(k,m,p,pi); i=k;                    // insert on k
    for (j=k;j>1;j--)
    { swap(pi+j,pi+j-1);                       // shift left
      c=Cmax(k,m,p,pi);                        // set new cmax
      if (c<cp) { cp=c; i=j-1; }               // store best location
    }
    t=pi[1]; for (j=1;j<i;j++) pi[j]=pi[j+1]; pi[i]=t;  // adjust pi[]
  }
```

```
}

// Nawaz, Enscore, Ham's algorithm for F//Cmax problem,
// efficient implementation from Taillard's paper, complexity // O(n^2*m)
void NEH(int n, int m, long p[][N+1], int pi[])
{ int i,j,k,l,t;
  long c,cp,s;
  long r[M+1][N+1],q[M+2][N+2],d[M+1];
  long sump[N+1];

  for (j=1;j<=n;j++) { s=0; for (i=1;i<=m;i++) s+=p[i][j]; sump[j]=s; }
  for (j=1;j<=n;j++) pi[j]=j; _sort(1,n,sump,pi);

  for (i=0;i<=m;i++) r[i][0]=0;                    // r[][0] edge values
  for (j=0;j<=n;j++) r[0][j]=0;                    // r[0][] edge values
  d[0]=0;                                          // d[0] edge value

  for (k=2;k<n;k++)
  {
    for (i=1;i<=m;i++) for (j=1;j<=k;j++)
      r[i][j]=max(r[i][j-1],r[i-1][j])+p[i][pi[j]];     // set new r[][]

    for (i=0;i<=m;i++) q[i][k]=0;                  // q[][k] edge values
    for (j=0;j<=k;j++) q[m+1][j]=0;                  // q[m+1][] edge values
    for (i=m;i>=1;i--) for (j=k-1;j>=1;j--)
      q[i][j]=max(q[i][j+1],q[i+1][j])+p[i][pi[j]];     // set new q[][]

    cp=r[m][k]; i=k; t=pi[k];                      // insert on k
    for (j=k-1;j>=1;j--)
    { for (l=1;l<=m;l++) d[l]=max(d[l-1],r[l][j-1])+p[l][t]; // set d[]
      c=d[1]+q[1][j];
      for (l=2;l<=m;l++) c=max(c,d[l]+q[l][j]);         // set new cmax
      if (c<cp) { cp=c; i=j; }                     // store best location
    }
    for (j=k;j>i;j--) pi[j]=pi[j-1]; pi[i]=t;          // adjust pi[]
  }
}

long power(int x, int n) { int i; long s=1; for (i=1;i<=n;i++) s*=x; return s; }

long eps(int x, int n) { int i; long s=0; for (i=1;i<=n;i++) s+=power(x,i-1); return s; }

// ad hoc generator of processing times
void generator(int n, int m, long p[][N+1])
{ int i,j;
  for (i=1;i<=m;i++)
    for (j=1;j<=n;j++) p[i][j]=1+random(99);
}

// Taillard's uniform generator [low,high]
int unif(long *seed, int low, int high)
```

```
{ static const long m=2147483647l,a=16807l,b=127773l,c = 2836l;
  long k;
  double value_0_1;

  k=*seed/b; *seed=a*(*seed%b)-k*c;
  if (*seed<0) *seed+=m;
  value_0_1=*seed/(float)m;
  return low+(int)(value_0_1*(high-low+1));
}

// Taillard's generator of instances with different p values
void generate(int n, int m, int k, long p[][N+1])
{ int i,j,t;
  long x,y,r,q,sp,lbb,td,lb;
  FILE *u;
  long _seed;
  char path[50];
  sprintf(path,"c:\\tcpp\\my\\recipes\\dane\\%d_%d.gen",n,m);
  u=fopen(path,"rt");
  for (i=1;i<=k;i++) fscanf(u,"%ld%ld%ld",&_seed,&td,&lbb);
  fclose(u);

  for (i=1;i<=m;i++) for (j=1;j<=n;j++) p[i][j]=unif(&_seed,1,99);
/*
  lb=0;
  for (i=1;i<=m;i++)
  { r=MAXL; q=MAXL; sp=0;
    for (j=1;j<=n;j++)
    { x=0; for (t=1;t<i;t++) x+=p[t][j];
      y=0; for (t=i+1;t<=m;t++) y+=p[t][j];
      if (x<r) r=x; if (y<q) q=y;
      sp+=p[i][j];
    }
    if ((r+sp+q)>lb) lb=r+sp+q;
  }

  for (j=1;j<=n;j++)
  { x=0; for (i=1;i<=m;i++) x+=p[i][j];
    if (x>lb) lb=x;
  }
*/
}

// shows permutation and non-permutation schedules
void show_pi(int n, int pi[])
{ int j;
  clrscr(); for (j=1;j<=n;j++) printf("%4d",pi[j]); printf("\n"); getch();
}

void main()
{  int i,j,n,m;
```

```
  long copt;
  n=20; m=5;

  generate(n,m,7,p);

//  clrscr();
//  nawaz(n,m,p,pi);
//  clrscr(); printf("%ld\n",Cmax(n,m,p,pi)); getch();
//  show_pi(n,pi);

//  NEH(n,m,p,pi);
//  clrscr(); printf("%ld\n",Cmax(n,m,p,pi)); getch();
//  show_pi(n,pi);

//  campbell(n,m,p,pi);
//  clrscr(); printf("%ld\n",Cmax(n,m,p,pi)); getch();
//  show_pi(n,pi);
//  getch();

//  rand_pi(n,pi);

  clrscr();
  fe=fopen("trace.txt","wt");
  sa(n,m,p,pi,&copt,pibest);
  fclose(fe);
  show_pi(n,pi);

//  clrscr();
//  copt=Cmax(n,m,p,pibest);

}
```