2001

Scheduling flexible flowshops with sequence -dependent setup times

Kanchana Sethanan
West Virginia University

SCHEDULING FLEXIBLE FLOWSHOPS WITH SEQUENCE DEPENDENT SETUP TIMES

Kanchana Sethanan

Dissertation submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Decision Sciences and Production System

Wafik H. Iskander, Ph.D., Chair
Alan R. McKendall, Jr., Ph.D.
John L. Harpell, Jr., D.B.A.
Majid Jaraiedi, Ph.D.
Ralph W. Plummer, Ph.D.

Department of Industrial and Management Systems Engineering

Morgantown, West Virginia
2001

Keywords: Flexible Flowshop, Hybrid Flowshop, Dependent Setup Times, Tabu Search, Heuristics

ABSTRACT

Scheduling Flexible Flowshops with Sequence Dependent Setup Times

Kanchana Sethanan

This dissertation addresses the scheduling problem in a flexible flowshop with sequence-dependent setup times. The production line consists of *S* production stages, each of which may have more than one non-identical (uniform) machines. Prior to processing a job on a machine at the first stage, a setup time from idling is needed. Also sequence dependent setup times (SDST) are considered on each machine in each stage. The objective of this research is to minimize the makespan. A mathematical model was developed for small size problems and two heuristic algorithms (Flexible Flowshop with Sequence Dependent Setup Times Heuristic (FFSDSTH) and Tabu Search Heuristic (TSH)) were developed to solve larger, more practical problems. The FFSDSTH algorithm was developed to obtain a good initial solution which can then be improved by the TSH algorithm. The TSH algorithm uses the well-known Tabu Search metaheuristic. In order to evaluate the performance of the heuristics, two lower bounds (Forward and Backward) were developed. The machine waiting time, idle time, and total setup and processing times on machines at the last stage were used to calculate the lower bound. Computational experiments were performed with the application of the heuristic algorithms and the lower bound methods. Two quantities were measured: (1) the performance of the heuristic algorithms obtained by comparing solutions with the lower bounds and (2) the relative improvement realized with the application of the TSH algorithm to the results obtained with the FFSDSTH algorithm. The performance of the heuristics was evaluated using two measures: solution quality and computational time. Results obtained show that the heuristic algorithms are quite efficient. The relative improvement yielded by the TSH algorithm was between 2.95 and 11.85 percent.

ACKNOWLEDGEMENTS

I am deeply grateful to my dissertation advisor, Dr. Wafik Iskander, who spent numerous hours to share his knowledge and intelligence. He continuously provided valuable guidance, comments, and encouragement throughout this work. He has always been available for help and advice in a friendly atmosphere that inspired creativity and motivation. Without his help, I would have never finished this research.

I am also grateful to my dissertation committee, Dr. John Harpell, Dr. Alan McKendall, Dr. Majid Jaraiedi, and Dr. Ralph Plummer, for their positive comments and suggestions, which greatly improved the quality of this research.

I am profoundly grateful to my dearest friends in Thailand who always gave me excellent encouragement throughout my graduate studies in the USA. I also thank Thai students and fellow graduate students in the Industrial and Management Systems Engineering Department at West Virginia University, who made my life and stay in Morgantown such a joyful and truly exceptional experience.

Special thanks to the department of Industrial and Management Systems Engineering, West Virginia University, which furnished hospitality for learning and conducting research and provided me with financial support throughout my graduate years in the department.

Finally, my deep appreciation goes to my parents, sisters, and brother, no matter how far you were, you were always there for me. Your endless love, confidence, great support, and excellent encouragement were crucial to my accomplishments and my well-being.

Dedicated to Luang Por Prarajchabhavanavisuthi,

my parents Sunee and Pichai Sethanan,

my sisters Wachiraporn and Amornrat Sethanan,

and my brother Nithi Sethanan.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1. Background

1.1.1 Scheduling

Scheduling is defined as the determination of relative position of jobs with respect to a processing machine, including the assignment of definite times at which processing occurs (Nawaz et al., 1983). Another view of scheduling is defined as the "allocation of limited resources to jobs over time to perform a number of tasks" (Baker, 1974, p. 2). Examples of resources include machines, operators, facilities, computers, and transporters.

The problem of scheduling *n* jobs on *m* machines is one of the classical problems in flowshop manufacturing that have been studied by researchers for many years. Additionally, scheduling plays an essential role in the entire manufacturing system. Production scheduling problems exist frequently in production environments whenever resources are required to perform a set of operations on jobs, and also when each operation can be accomplished in more than one way (Randhawa & Kuo, 1997). Normally, there are two categories of constraints that are commonly found in scheduling problems. First, there are restrictions on the capacity of available resources and, second, there are technological limits on the order in which jobs can be performed. Resource constraints generally refer to processor capacities and limitations. Technological constraints include alternative routing and precedence relationships. Alternate routing means that the product can be produced on more than one processor, while precedence constraints mean that the processor cannot process a specific job if some other job is not completed. Scheduling problems involve the assignment of

machines to various jobs and determination of the order in which the jobs will be performed in order to optimize some criteria while satisfying the shop constraints. Generally, there are three issues concerned with scheduling jobs on a set of machines (Cheng & Sin, 1990):

1. What machine should be allocated to which job?
2. How to sequence the jobs in order to obtain the best schedule and meet the constraints?
3. How can the reasonableness of a schedule be rationalized?

Hence, the scheduler wishes to optimize some measures of effectiveness (such as minimization of makespan, mean flow time, lateness, or inventory) which may vary from one situation to another, and to satisfy the production constraints (e.g. production requirements, resource capacities, or operation procedures).

There are three issues that need to be specified when defining a scheduling problem. These three issues, as presented by Cutright (1990), are:

1. Length of planning horizon,
2. Nature of tasks that will be scheduled, and
3. Criteria used to determine the best schedule.

Planning Horizon

Planning (time) horizons are usually classified as long-term, intermediate-term (or medium-term), and short-range. Long-term planning typically involves capacity and strategic issues and is the responsibility of the top management. Management formulates policy-related questions such as gross labor-hours, machine-hours, floor space, customer policies, new product development, research funding, and company goals (Vollmann et al., 1992). Normally, the length of the

long-term planning horizon is at least five years. This research assumes that all long-term decisions have been made.

Once the long-term planning is made, operation managers begin intermediate-range planning in order to meet the objectives of the firm, subjected to a set of constraints imposed by the long-range planning decisions. Intermediate planning involves activities such as the determination of production plans, workforce levels, and forecasting product demand. Typically, the time horizon of short range planning is in months. It is also assumed in this research that all of these decisions have been determined and that workforce levels are fixed.

Short-range planning is dependent on both long and intermediate-range planning decisions. Operations managers make these plans in conjunction with supervisors and foremen who desegregate the intermediate plan into weekly, daily, or hourly schedules. Short-range planning uses the production plan and workforce level from the intermediate planning stage to determine job scheduling through the resources in order to meet the criteria. The time horizon of short-range planning is usually in days.

Nature of the tasks in the shop-floor system

The nature of tasks (or jobs) to be scheduled involves the following issues and questions:

1. Can a job be split in case there are more than one processors capable of performing it?
2. Are there several processors that can perform the same job?, or
3. Is the order of operations the same for each job?

Scheduling Criteria

Scheduling criteria are always a function of completion time of the jobs and may also be a function of the due date. Examples include minimization of flow time, lateness, or tardiness.

1.1.2 The Place of Scheduling within an Organization

The scheduling function must interface with many other important functions in the manufacturing systems (e.g. production planning, master production planning, material and capacity planning, etc.) as shown in the information flow diagram in Figure 1.1. In order to provide the departments in an organization access to the necessary scheduling information and enable the departments to provide the scheduling system with relevant information (e.g. changes in jobs' data and status of machines), a management information system (MIS) or a decision support system (e.g. forecasting, aggregate planning, and master production scheduling) is probably needed (Chen, 1997). The process of scheduling begins with capacity planning (also called long-term planning) which involves facility and equipment acquisition. Intermediate planning includes aggregate and master production planning. In the aggregate planning stage, decisions regarding the use of facilities, people, and inventories are made. The master schedule then desegregates the aggregate planning and develops an overall schedule for outputs. Short-term schedules then translate capacity decisions, intermediate planning, and master schedules into job sequences, specific assignments of personnel, machinery, and material.

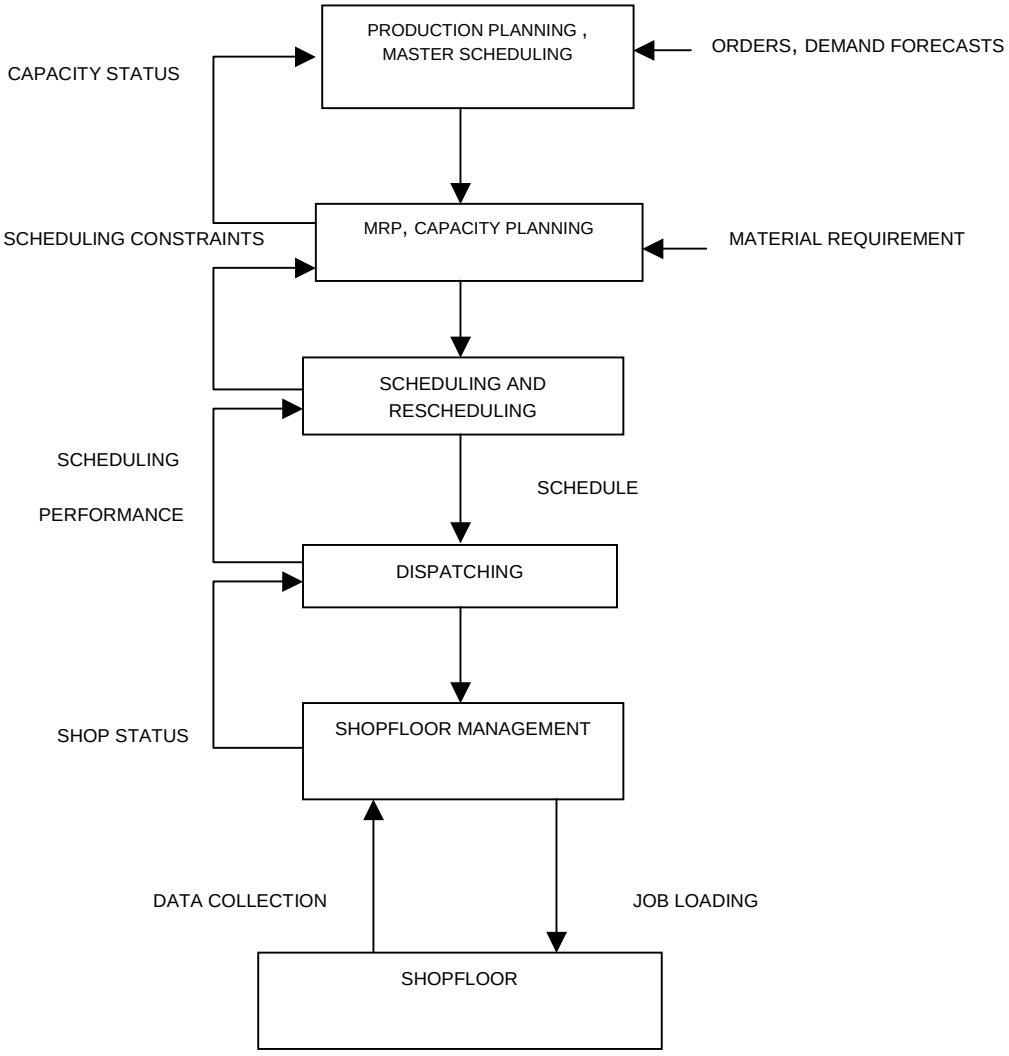

Figure 1.1: Information Flow Diagram in a Manufacturing System (Pinedo,1995)

1.1.3 Classification of Sequencing Problems

To classify the major scheduling models, it is necessary to characterize the configuration of resources and the nature of tasks. For instance, a model may contain one resource type (single-stage problems) or several resource types (multistage problems). If the set of tasks available for scheduling does not change over the time, the system is called *static*. Conversely, if new tasks arise over time, the system is called *dynamic* (Baker, 1974).

Day and Hottenstein (1970) depict a schema for classifying sequencing problem as presented in Figure 1. 2. The framework shows that the sequencing problems have been categorized according to the following components:

1. the nature of job arrivals, such as fixed batch size or continuous arrivals which are given by a probability density function.
2. the number of machines involved, for instance, single machine production (m = 1) or multi-machine production (m > 1), and
3. the nature of job route.

Further classification could be added to this figure which would include characteristics such as setup time (e.g. dependent or independent of job sequence on a given machine) and due date considerations.

This research focuses on a static scheduling problem: A flexible (hybrid) flowshop with dependent setup times, which minimizes the maximum completion time of all jobs. The jobs are available at time zero and have sequence dependent setup times on machines at each production stage. All parameters such as processing and setup times are assumed to be known with certainty.

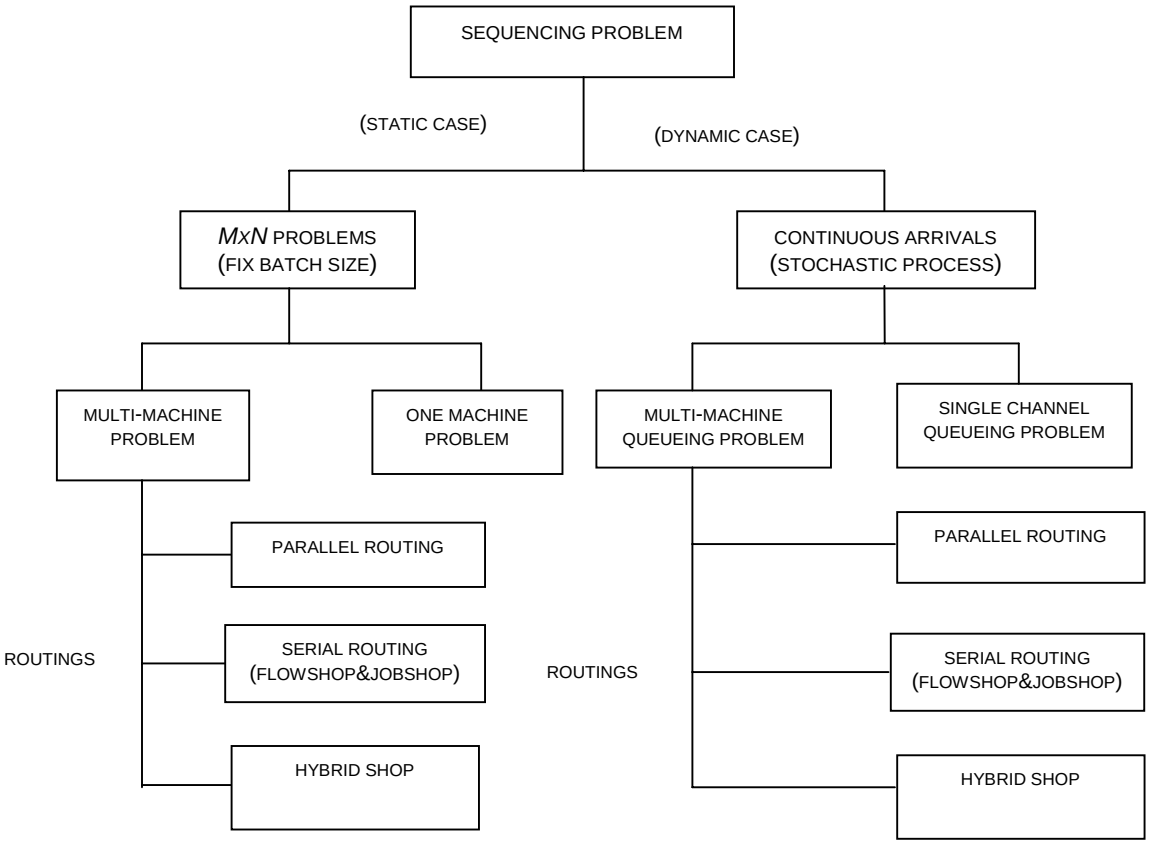

Figure 1.2: A Classification of Sequencing Problems

1.1.4 The General Flowshop Scheduling Problem

Flowshop scheduling problems can be classified into two categories: general flowshop and permutation flowshop (Pinedo, 1995; Chen,1997). For the permutation flowshop, each of the *n* jobs is processed on the machines (*m* =1, 2, ..., *M*) in the same order (Osman & Potts, 1989). On the other hand, the processing sequences of jobs on machines from one stage to another could be different in the general flowshop. In addition, flowshop scheduling may be classified as static or

dynamic. In general, a static scheduling problem specifies a number of *n* jobs and an optimal schedule is to be found with respect to the *n* jobs only (Dudek et al., 1992), while a dynamic scheduling problem specifies that jobs are constantly entering and leaving the job file according to some probability distribution in the stochastic process (Day & Hotenstein, 1970).

The majority of the research published has thus far been devoted to the static problem. The early work started with Johnson (1954) for the two-machine case. Johnson's algorithm finds an optimal sequence that minimizes the maximum flow time (called makespan) for all jobs. The simplicity of Johnson's method encouraged other researchers to extend his idea in order to find optimal sequences for the *M*-machine problem. For the *M* machine case, the Campbell, Dudek, and Smith's (1970) heuristic (CDS), which extends Johnson's algorithm, is considered to be a very effective and robust heuristic (Ho & Chang, 1991). Generally, the static flowshop problems have the following characteristics (Baker, 1974; Gupta, 1977; Stafford and Tseng, 1990; and Sarin& Lefoka, 1993, and Pinedo, 1995).

1. Each machine can process at most one job at a time.
2. Each job can be processed on at most one machine at a time.
3. Preemption and splitting of any particular job are not allowed.
4. Jobs are processed on each machine in the same order.
5. All *N* jobs are available for processing at time zero.
6. All machines are available at time zero and are independent.
7. The processing time of each job on each machine is a known value.
8. Jobs are independent of one another.

1.1.5 A Flexible Flowshop Environment (FFS)

A flexible flowshop (FFS) is a generalization of the flowshop and the parallel processor environments. A flexible flowshop is alternatively called a hybrid flowshop or multiprocessor flowshop. In the most general setting of a flexible flowshop environment, there are multiple stages (*S* stages), each of which consists of *m(s*) (*s* = 1, 2, 3,…,*S*) parallel processors). A schematic representation of a flexible flowshop environment is given in Figure 1.3. The processors in each stage may be identical, uniform, or unrelated. Machines are uniform if the time to process a job on any machine is a constant ratio of its processing time on other machines. In other words, uniform machines are identical processors that do not have equal speeds. Unrelated machines are machines for which the time to process a job on any machine has no particular relationship of its processing time on any other machine (Cheng & Sin, 1990). In a FFS environment, each job is processed first at stage 1, then at stage 2, and so on. Normally, a job requires only one machine at each stage and any machine can process any job.

1.1.6 Dependent Setup Times

Setup time is the time used to prepare the process of jobs on machines (Allahverdi et al., 1999). Consequently, the requirements of setup times of jobs are very common in many real manufacturing situations. This includes setting up tools such as jigs and fixtures, cleanup, inspecting material, and positioning the jobs. The issue of setup time has been of much interest in the past few decades. According to the Goldratt Theory Of Constraint (TOC) (Goldratt, 1990), setup reduction efforts can improve performance, but only if concentrated on production bottlenecks or constraints. The total time for a machine can be classified as either production time, setup time, idle time (i.e., time not used for setup or processing), or

waste time (i.e., time spent processing material that cannot be converted into throughput; for instance, time to process products for which there is no demand). It is possible to improve the efficiency or capacity of a resource by reducing idle time and waste time, cutting or reducing the total setup time, and reducing the production time per unit of the product.

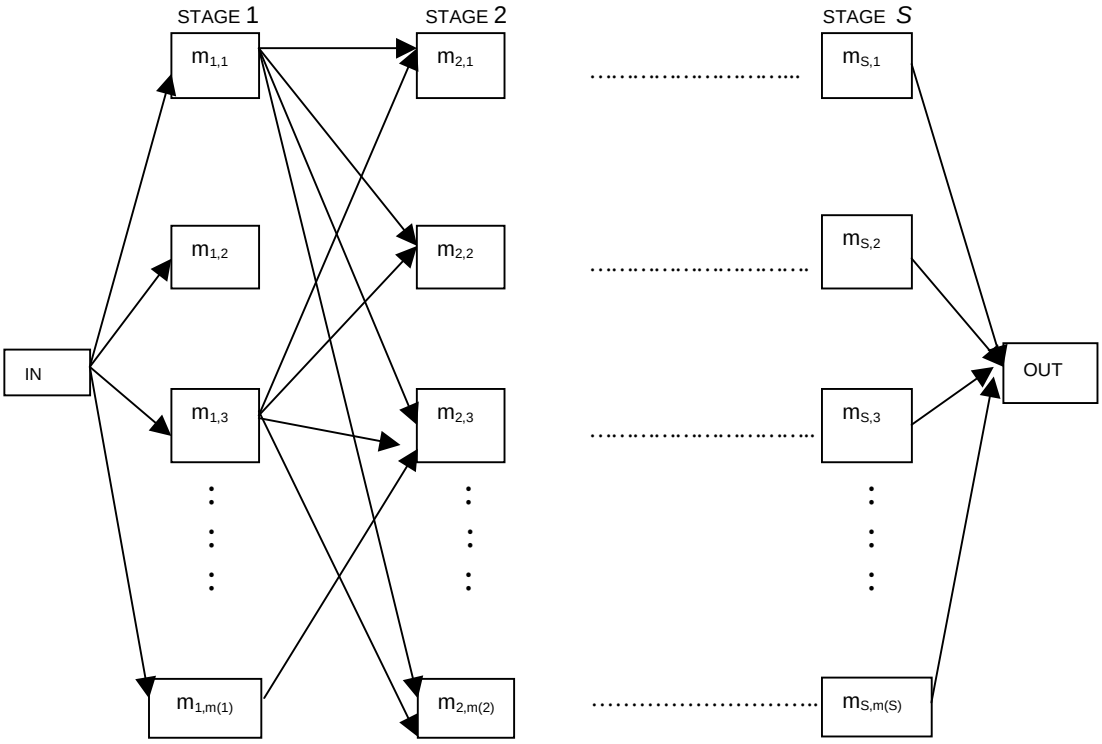

Figure 1.3: A Schematic Representation of a Flexible Flowshop Environment

Typically, there are two categories of setup times. In the first category, setup time is sequence independent. That is, i.e., it depends only on the job to be processed. In the second, setup time is sequence dependent as it depends on both the job to be processed and the preceding job. Another view of setup time classification adopted by Randhawa and Kuo (1997) includes: (1) processor dependent, (2) product dependent, and (3) both processor and product dependent. Processor dependent setup time deals with the setup time that depends only on the processor, regardless of the job type, while product dependent setup time refers to the setup time that depends only on the product, regardless of the machine type.

Sequence dependent properties (e.g. setup times or costs) are considered to be important factors in the manufacturing environment, especially, when a shop floor is operated at or near its full capacity (Wilbrecht & Prescott, 1969). Sequence dependent setups are commonly found both in a single machine type or a multiple machine type. Even though there exists an enormous amount of research on the flowshop scheduling problem, research study has rarely been conducted in the case where setup times are sequence dependent (Simon Jr., 1992; Allahverdi, 1999). Hence, the results of these research studies lack a practical solution for applications that require the treatment of setup times. For this reason, dependent setup times cannot be neglected and hence are considered in this research.

Sequence dependent setups occur especially in process industry operations, where machine setup time is significant and is needed when products change. The magnitude of setup time depends on the similarity in technological processing requirements (routing and precedence relationships) for the successive jobs (Srikan & Ghosh, 1986). Normally, similar technological requirements for two consecutive jobs would require lesser setup. For example, if the previous and the

current products processed on the machine are from the same family that consists of a set of similar products (or jobs) in terms of processing, then the changeover time between those two products is small. The changeover times depend on the family of products. This type of production system can be found in many industries such as pharmaceutical, cosmetic, chemical, and food and brewing industries. The following are real life examples of dependent setup times:

1. In printing industry, the cleaning and setting of presses are dependent on the color of ink and size of paper.
2. In textile industry, weaving and dyeing setup operations depend on jobs.
3. In brewing and food industry (for container and bottling section), settings are changed when the containers or bottle sizes change.

This research focuses on the scheduling problem in a flexible flowshop with sequence dependent setup times. A complete description of the problem is given in Chapter 2.

CHAPTER 2

STATEMENT OF THE PROBLEM

2.1 Introduction

Nowadays, manufacturing companies are faced with market demands for a variety of high quality products. These companies must, therefore, make their production systems more flexible, reduce costs related to production, and respond rapidly to demand fluctuations. Hence, companies need to have advanced techniques and an increasingly high degree of automation.

Production and operation management has been an interesting topic in manufacturing, especially in such areas as job scheduling and system control. The development of production schedules is a remarkably important task in industry. Many scheduling researchers have focused their research on sequencing and timing the scheduling of multiple non-identical jobs through one or more machine stations (Egbelu, 1991). A challenge facing many manufacturing and service industries is job assignment to parallel processors (e.g., workers or machines). Parallel processing is the situation where a job can be processed by more than one processor, but only one processor can actually work on one job. This type of production system where multiple products are processed on parallel, non-identical machines is common in both manufacturing and service industries. For instance, airline companies may assign one of several types of airplanes to service a route. In industries such as semiconductor manufacturing, it is common to find newer or more modern machines running side by side with older and less efficient machines. Even though the older machines are less efficient, they may be kept in the production lines because of their high replacement costs. The older machines may perform the same operations as the newer ones, but would generally require longer processing time for the same operations. Other examples include textile

plants assigning jobs to looms and paper plants assigning products to different paper machines (Randhawa & Smith, 1995). So, even though those resources may be of similar type, their production rates may be different. This research will focus on scheduling non-identical jobs in a flexible flowshop (or hybrid) environment with sequence dependent setup times as described in the following section.

2.2 Manufacturing Background

Nature of the Tasks in the Shop-floor System

In this research, production is restricted by resource and technological constraints. Processors (or machines) can process the same jobs but differ in their speeds. Thus, the production rate for the same job may be different between machines at the same stage, which results in different production costs per unit of the product.

This research deals with the general flexible flowshop, with *S* production stages, in which the job sequence may not be the same on each machine at each stage. The problem on hand has several distinct product families, and within each family there are different product types. Each production stage may be composed of more than one machine. If a stage has multiple machines, all machines would be similar in function but different in their performance. All products may be processed on any of the machines in a stage. It is assumed that the slowest machine in each stage has the lowest production performance for all products. The problem hence will be developed and solved for the parallel processing case with uniform processors.

Each product *i* of family *j* requires *PTime(j,i,s,m)* units of processing time on machine *m* of stage *s*. A production line requires a setup time to change over from one product to another. Machine changeover is needed when the product changes both within a family and between families. In this research, two types of machine changeovers (minor and major changeovers) are identified. A minor changeover time is

the changeover time required if the previous product belongs to the same family. On the other hand, if the previous product was of a different family, a major changeover time would be required. The changeover time for machine *m* of stage *s* between product *i* of family *j* and product *p* of family *q* is denoted *by ch(j,i,q,p,s)*. If $j = q$, then this changeover time is minor, but if $j \neq q$, then it is a major one. The changeover time in this research is assumed to be asymmetric. This means that *ch(j,i,q,p,s)* may not be necessarily equal to *ch(q,p,j,i,s)*. It is also assumed that changeover times are equal for all machines in the same stage of a production line when changing from one product to another, but the changeover time may be different between stages.

The processing on all stages is not preemptive, which means that a new product cannot enter into the stage until the previous product has been completely processed.

2.3 Problem Statement

This research addresses the problem of scheduling jobs in a flexible flowshop in which machines are uniform. A job used in this study is synonymous with an order and represents an individual, distinct demand for a product. Each production stage may be composed of more than one machine. Prior to processing a job on a machine in a production line, there is an associated setup time. Setup times are considered significant and typically depend on the sequence of the jobs through the processors. The problem considered in this study is complex in three ways:

1. Even though the flexible flowshop scheduling problems have been studied by several previous researchers, very few of them have considered both products and families in their models. This research addresses products which are grouped into families to be processed in a flexible flowshop environment. There are different products within each family, and there are many families to be considered.

2. Both major and minor setup times are considered. A major setup time is required if a machine at any stage switches from one family to another. On the other hand, a minor setup time is needed if the previous product belongs to the same family.
3. The system consists of *S* stages of production. Each production stage may consist of more than one non-identical (uniform) machines. The production line may have different number of machines in each stage. The system can produce a number of products and families, and all products and families can be produced on every processor.

This research addresses the problem of scheduling all products on the machines at the different stages in order to minimize the makespan.

2.4 Assumptions

The assumptions made in formulating the problem are as follows:

1. It is assumed that the decisions about production plans, workforce levels, and layout of the facility have been made from the long and intermediate-range planning.
2. Production is make-for-stock; hence, there are no due dates associated with batches or products.
3. All jobs and machines are available at the beginning of the scheduling process (at time zero).
4. There are many stages in the flowshop production line. Each stage may have several non-identical but uniform machines.
5. Jobs may not be necessarily scheduled in the same order in all stages.
6. Jobs can wait between two production stages (or stations) and the intermediate storage is unlimited.

7. Within the same product family, minor changeover times may not be equal between products. Likewise, major setup times may not be equal between families.
8. Setup times for jobs on each machine are dependent on the order in which jobs are processed, but it is also assumed that setup times are equal for all machines in the same stage when changing from one product to another.
9. No job splitting is allowed. A job must be completely finished on one machine before it can be manufactured on the succeeding machine.
10. There is no job preemption.

2.5 Research Objectives

The major objectives of this research are:

1. To formulate a mathematical model to solve the problem and to produce an optimal schedule in order to minimize the total makespan.
2. To develop efficient scheduling heuristics to find approximate solutions for large-size problems.
3. To evaluate the heuristics developed by comparing their results to good lower bounds.

CHAPTER 3

LITERATURE REVIEW

3.1 Introduction and Overview

This research focuses on a static sequencing of a flexible flowshop (FFS) environment. In a FFS environment, there are *S* production stages with one or more machines at each stage. Sequence dependent setup times (SDST) are considered on each machine. A review of previous work on flowshop scheduling is performed, along with a review of the SDST flowshop literature. Also, a review of the literature on the application of the Tabu search (TS) algorithm relevant to this study is presented.

A popular notation used in scheduling problems has the form of $\alpha/\beta/\gamma$. The first parameter (α) describes the machine environment and contains a single entry. The second parameter (β) is a field providing the details of processing characteristics and constraints. The β field may contain no entry, a single entry, or multiple entries. The last parameter (γ) contains the objective to be minimized and usually contains a single entry. Flowshop problems deal with *m* stages in series and with one machine in each stage, and are denoted, in general, as $Fm//C_{max}$ when makespan is to be minimized. If there are several processors in each stage and all of them are identical, the problem becomes a flexible flowshop, denoted as $FFs(Pm_1,Pm_2,\ldots,Pm_S)//C_{max}$. If the machines are uniform in the flexible flowshop, then Pm_s are replaced with Qm_s for s =1,2,…,S. When setup times are involved, the notation becomes $FFs(Pm_1,Pm_2,\ldots,Pm_S)/s_{ip}/C_{max}$ and $Fm/s_{ip}/C_{max}$ for the flexible flowshop and regular flowshop problems, respectively. In addition, if the setup time between job *i* and *p* depends on the machine, then the subscript *m* is added, that is, it becomes s_{ipm}. A complete list of the notation used in this study is presented in Appendix A.

Before reviewing the literature on flowshop scheduling, a review of the methodology for solving sequencing problems in general is presented in the following section.

3.2 Solution Methodologies for Scheduling Problems

After determining the context in which scheduling is being defined, the methodology for selecting a "good" schedule solution is determined. Day and Hottenstein (1970) state that there are four common approaches used to solve the static scheduling problem. These approaches are described below:

3.2.1 Combinatorial approach

Combinatorial approaches are based on the changing of one permutation to another by switching jobs around in order to optimize a given objective function.

3.2.2 Enumerative optimal methods

The most general techniques are mathematical formulations (including linear programming, dynamic programming, integer programming, or mixed integer programming), and branch and bound methods.

Scheduling problems are typically represented as an optimization problem subject to a set of constraints. The problem takes the form of a mathematical model that expresses the desired objective subject to the constraints set forth in the problem. However, there are many difficulties in formulating mathematical models. These difficulties include the complexity of the interactions among many variables in a system, the difficulty in the attempt to optimize the schedule from the system, and the difficulty in gaining an agreement among these variables on what is essential for the good of the system (Cutright, 1990).

Typically, the mathematical model for the problem is either too difficult or too time-consuming to solve in reasonable time. Since the development of a mathematical model is a time-consuming task and requires a thorough understanding of the system being represented, it is necessary to find solution techniques that are easy to implement even though they may not always lead to an optimal solution. These techniques include heuristic approaches and Monte Carlo sampling which are described below.

3.2.3 Heuristic approach

Generally, difficulties arise in solving scheduling problems. Exact solution procedures may not exist or may be too expensive to apply for large-sized problems. One then has to use procedures that yield good (but not necessarily optimal) solutions. These methods are termed heuristics. Heuristic approaches can be divided into:

1. exact solution to a relaxed problem such as LP relaxation and Lagrangian relaxation,
2. local search procedures including search techniques such as tabu search (TS), genetic algorithm (GA), or simulated annealing (SA), and
3. ad hoc decision rules.

3.2.4 Monte Carlo sampling

Monte Carlo method is a technique for the solution of a model using random (or pseudo random) numbers. For this approach, a scheduling problem is solved by taking random samples of feasible solutions and using the best of these solutions. Ideally, the number of samples would be as large as possible.

3.3 Flowshop Scheduling Models

In order to discuss relevant research in the area of flowshop scheduling, the topics reviewed are divided into three categories: (1) models without SDST consideration, (2) models which explicitly consider SDST, and (3) previous work concerned with TS application to solve the flowshop scheduling problems.

3.3.1 Flowshop Scheduling Models without SDST Considerations

3.3.1.1 G*eneral Flowshop Scheduling (Fm/ /C_{max})*

The flowshop scheduling problem with no setup times has been researched extensively over the past five decades. Work on these problems was pioneered by Johnson (1954), who presented a simple algorithm for solving the F2//C_{max} problems to optimality in a polynomial time. A wealth of research then followed but will not be covered here as it is not relevant to the problem at hand.

3.3.1.*2 Flexible Flowshop Scheduling* (*FFs/ /C_{max})*

A flexible flowshop environment consists of *S* production stages, each of which having *m(s)* parallel machines, *s* =1,2,…,*S.* The machines in each stage may be identical, uniform, or unrelated. This section reviews previous work performed in a flexible flowshop environment without SDST considerations.

3.3.1.2.1 Exact Approaches

<u>Two-stage cases</u>: FF2(Pm_1,Pm_2)//C_{max}

Arthanary and Ramaswamy (1971) were the first to develop the FFS problem (Soewandi, 1998). They proposed a branch and bound algorithm for the two-stage FFS problem in which there are *m* identical machines in stage 1 but only one machine in stage 2, FF2(Pm_1, Pm_2 =1)//C_{max}. They could optimally solve problems with up to 10 jobs with reasonable computational effort.

According to Gupta (1988), the two-stage flowshop problem in which each stage consists of identical multiple machines, FF2(P*m_1,Pm_2*)//C_{max}, is

NP-complete. He proposed a heuristic to solve a special case when there is only one machine in the second stage in order to minimize the makespan, FF2(Pm_1,Pm_2=1)//C_{max}. Computational experiments showed that the effectiveness of the proposed heuristic increases as the problem-size increases.

Gupta and Tunc (1991) considered the FFs(Pm_1=1,Pm_2)//C_{max} and established approximate solution algorithms. They also developed a branch and bound algorithm using the heuristic solution as an upper bound on makespan. Their results showed that when the number of machines at stage 2 is equal to or greater than the total number of jobs, the Longest Processing Time (LPT) scheduling rule yields optimal solutions. For the case in which the total number of jobs is greater than the number of machines in stage 2, they developed two heuristics to minimize the makespan. Computational results indicated that the effectiveness of the algorithms increases with the increase of the total number of jobs. For the cases in which the deviations of the heuristic makespans were relatively large from the lower bounds, an improved branch and bound algorithm was developed. The maximum number of jobs reported in their work was only eight jobs.

Multiple stage cases (FFs(Pm_1,Pm_2,…,Pm_S)//C_{max})

Brah and Hunsucker (1991) and Ragendran and Chaudhuri (1992) developed branch and bound algorithms for the FFs(Pm_1,Pm_2,…,Pm_S)//C_{max}. Both studies can solve only small-sized problems. Portmann et al. (1998) also studied the FFs(Pm_1,Pm_2,…,Pm_S)//C_{max} problem. They improved the lower bound of Brah's and reduced the number of branches used in the search tree. They also used a genetic algorithm (GA) approach to improve the search. Their computational experiments indicated that optimal solutions using their branch and bound approach were more often reached using the GA approach. They

could solve problems with up to five stages (3, 3, 1, 2, and 2 machines in stages 1 through 5, respectively) and 15 jobs with an average deviation of 3% from the results of the branch and bound algorithm.

Moursli (1995) also investigated on the FFs(Pm_1,Pm_2,…,Pm_S)//C_{max} problem. He derived three improvements from Brah's algorithm and three new lower bounds. His computational experiments showed that his algorithm could solve problems with up to 20 jobs to optimality. Both number of nodes investigated and running time were drastically reduced in his approach. Another study was done by Vignier et al. (1996). They developed a branch and bound approach to solve FFs(Pm_1,Pm_2,…,Pm_S)//C_{max} and solve problems with up to 15 jobs.

3.3.1.2.2 Heuristic Approaches

Two stage cases (FF2(Pm_1,Pm_2)//C_{max})

Lee and Vairaktarakis (1994) developed five new lower bounds for the FF2(Pm1,Pm2)//C_{max} problem. They also proposed a heuristic to solve the FF2(Pm1,Pm2,…,PmS)//C_{max} problem. However, their results were not reported. In 1996, Guinet et al. studied the scheduling for the FF2(Pm1,Pm2)//C_{max} problems. They developed a heuristic and three lower bounds. The computational results showed that the average gap compared between the heuristic solution and lower bounds are less than 0.73%. Another study was done by Haouari and Hallah (1997). They developed a new lower bound and used the Simulated Annealing (SA) and TS approaches to solve the problems. According to the solutions of these problems, the TS based heuristic yielded an optimal solution for 35 % of the cases and an average relative error of only 0.82%. In 1998, Soewandi developed a new procedure, which he termed "Improved, Modified Johnson's Order" to solve the FF2(Pm_1,Pm_2)//C_{max} and

$FF3(Pm_1,Pm_2,Pm_3)//C_{max}$ problems. He also considered the two-stage FFS with uniform machines at each stage ($FF2(Qm_1,Qm_2)//C_{max}$) and developed a solution procedure adapted from Johnson's rule. Additionally, he proved that his heuristic has a worst case performance Bound[1] (w.c.p.b) for the $FF2(Qm_1,Qm_2)$ problem as $1+\max\{\frac{(m_1-1)(v_{m1},1)}{\sum_{n=1}^{m1} v_{n,1}}, \frac{(m_2-1)(v_{m2},2)}{\sum_{n=1}^{m2} v_{n,2}}\}$ where v_m is the speed of machine m, and m_s is the number of machines in stage s. Further, he developed two heuristics for $FF3(Pm_1=1,Pm_2,Pm_3=1)//C_{max}$. Riane and Artiba (1997) and Riane et al. (1998) studied $FF3(Pm_1,Pm_2,Pm_3)//C_{max}$ problems, and developed two heuristics to cope with realistic problems. The experimental results indicated that their heuristics can solve problems with up to 130 jobs with a relative errors less than 1% of the lower bound.

Multiple stage cases : $FFs(Pm_1,Pm_2, \ldots, Pm_S)//C_{max}$

In 1994, Ding and Kittichartphayak developed three heuristics for scheduling in $FFs(Pm_1,Pm_2, \ldots, Pm_S)//C_{max}$. The computational results showed that one of their heuristics, called the combined approach, is the best and can solve problem sets with number of jobs up to 8 with an average error less than 3% of the optimal solutions.

Multiple stage cases : $FFs(Qm_1,Qm_2, \ldots, Qm_S)//C_{max}$

A multi-stage FFS scheduling problem in which jobs are identical and machines are uniform at each stage was considered by Verma and Dessouky (1999) with the objective of minimizing the makespan. They compared the Latest Start Time (LST) rule with other heuristics: the Fastest Available Machine

[1] An index that indicates the deviation of the performance values yielded by an algorithm, in the worst case, from the optimal solution for a given problem, or in some cases, from the values of the best known solutions or lower bounds.

Heuristic (FAMH), the Earliest Completion Time Heuristic (ECTH), and the Mix Heuristic (MH). Their results indicated that the FAMH had a worst case absolute bound that was twice as large as the ECTH, LSTH, and MH heuristics.

3.3.2 Flowshop Scheduling Models with SDST Consideration

3.3.2.1 General Flowshop Scheduling (*Fm/ s_{ipm} /C_{max}*)

Allahverdi et al. (1999) presented a review of scheduling problems involving setup considerations. They classified scheduling into batch and non-batch, sequence-dependent, and sequence-independent setup. They also summarized the results from the existing research and provided guidelines for future research.

3.3.2.1.1Exact Approaches

Two-machine cases (F2/ s_{ipm} /C_{max})

Prior to the research of the multiple machine problem, the two-machine scheduling problem had been investigated by several researchers (e.g. Corwin & Esogbue, 1974; Gupta, 1986, etc.). Corwin and Esogbue (1974) considered two different flowshop scheduling problems with one of the machines having no setup times. The objective of their study was to find the minimum makespan. After establishing the optimality of permutation schedules, they solved the problem using a dynamic programming formulation. Their findings showed that, from computational standpoint, their formulation was comparable to that of the traveling salesman problem (TSP). On the other hand, Gupta (1986) formulated the Fm/s_{ipm},no wait/C_{max} problem as a TSP for the case in which jobs are processed continuously through the shop. He showed that the flowshop scheduling problem with SDST is NP-hard for the cases of limited or infinite intermediate storage space available to store partially completed jobs. The

results from the TSP formulation of the continuous processing case were used to describe an approximate solution for the cases in which the storage spaces were limited or finite.

In addition to Corwin & Esogbue's and Gupta's studies, one of the studies of Szwarc and Gupta (1987) was in terms of a special flowshop scheduling problem with sequence dependent additive setup times. They developed a polynomially bounded approximate method with the objective of minimizing makespan.

Multiple machine cases (Fm/ s_{ipm} /C_{max})

Excellent efforts to solve the SDST for the *m*-machine flowshop problem to optimality were performed by Srikar and Ghosh (1986). They developed a method to reduce the number of constraints and binary variables in a MILP formulation of the *m*-machine flowshop in order to minimize the makespan. They could solve problems with up to six machines and six jobs; however, the time required to solve problem was too large (22 minutes of CPU on a Prime 550 computer). Stafford and Tseng (1990) later discovered an error in Srikar and Ghosh's model. They corrected it and solved the problem using LINDO. They developed new MIP formulations for the regular flowshop problem and for the no intermediate queues (NIQ) flowshop problem.

Exact optimization schemes are mostly based on the application of a branch and bound (B&B) algorithm. The important part of a successful B&B procedure lies in the computation of the lower bounds. In 1997, Rios-Mercado developed several inequalities for two MIP formulations of the Fm/s_{ipm}/C_{max} problem. He used a branch and cut (B&C) procedure and found that this procedure is effective compared to a branch and bound (B&B) algorithm. The

main difference between the B&C and B&B procedures is that B&C algorithms reduce the problem size (or a set of unevaluated nodes) by adjoining valid inequalities (cutting planes or cuts). This, in turn, provides a stronger linear programming-representation.

Recently, Rios-Mercado (1997) and Rios-Mercado and Bard (1999) presented a branch and bound scheme for the SDST permutation flowshop scheduling problem in order to minimize the makespan. Their algorithm included the implementation of lower bounds and upper bounds and a dominance elimination criterion, and yielded a significantly better performance over previous work. They also could solve 100%, 43%, and 23% of 10-, 15-, and 20-job problems, respectively, within a 1 % optimality gap. Gupta (1982) proposed a branch and bound algorithm for the solution of the SDST flowshop with the objective of minimizing the total setup times of machines. Unfortunately, the computational results from the experiments were not reported. Because of the complexity of the multiple machine scheduling problem, thus far no approach has been found to solve the SDST flowshop to optimality for large-size problems.

3.3.2.1.2 Heuristic Approaches

Heuristic algorithms for the $Fm/s_{ipm}/C_{max}$ problem were developed by Simons (1992), Rios-Mercado (1997), and Rios-Mercado (1999). Simons (1992) developed four heuristics and compared them with three existing approaches (or benchmark) that represent generally practiced approaches to scheduling in this environment. However, only two of their proposed heuristics (called SETUP and TOTAL) produced better results than the other heuristics tested. In addition, computational experiments showed that problems with up to 15 machines and 15 jobs could be solved.

Evidently, the most relevant work on heuristics for the $Fm/s_{ipm}/C_{max}$ problem was conducted by Rios-Mercado (1997; 1999). They developed two heuristics called HYBRID and GRASP to solve the problem. Experimental results showed that the HYBRID heuristic outperforms GRASP when the number of machines is small and when setup time fluctuations are large.

Moreover, Rios-Mercado and Bard (1998) made a comparison between Simons's and Rios-Mercado and Bard's heuristics in relation to the $Fm/s_{ipm}/C_{max}$ problems and concluded that, in general, Rios-Mercado and Bard's heuristics outperformed Simons's SETUP heuristic. Nonetheless, in terms of better solutions for the cases in which both setup and processing times are identically distributed, Simons's SETUP heuristic is relatively superior to Rios-Mercado and Bard's algorithms.

Another performance measure investigated by several researchers is the minimization of the sum of weighted tardiness. Scheduling jobs on parallel machines with SDST considerations were considered by Lee and Pinedo (1997). They developed a three-phase heuristic, and a local search technique using SA that is applied at the last phase. Additionally, Randhawa and Smith (1995) investigated the factors that affected scheduling environments consisting of parallel and non-identical processors. These factors are the processing capacity relationships, sequencing and assignment rules, job sizes, and demand distributions. They measured the effects of variables by comparing the mean flow time, processor utilization spread, and proportion of tardy jobs. Computational experiments showed that, setup times and system loading parameters were important factors in the system performance.

3.3.2.2. Flexible Flowshop Scheduling ($FFs/s_{ipm}/C_{max}$)

To date, no literature in the flexible flowshop with sequence dependent setup time has been found. However, some literature is available on flexible flowshops with independent time for the $FFs(Pm_1,Pm_2,\ldots,Pm_S)//C_{max}$ problem as presented below.

Setup times may simply be included in the processing times in the situations where the entire batch of products is processed on one machine. Conversely, if the same batch of products is partly assigned to several machines, the same amount of setup time is still needed for the machines they are partly assigned to and cannot be simply added to the processing times.

Li (1997) considered a two-stage FFS with a single machine at the first stage and several identical machines at the second stage, and independent setup times with the objective of minimizing the makespan, $FF2(Pm_1=1,Pm_2)//C_{max}$. He developed two heuristics adapted from previous work to solve the problem. Gupta and Tunc (1994) developed polynomial heuristics for the two-stage FFS scheduling problems in which there is only one machine in stage 1 and identical machines in stage 2 but the number of machines at this stage is equal to or larger than the total number of jobs. They also considered setup and removal times independent from the processing times. The computational results indicated that the effectiveness of the proposed algorithms increases when the number of jobs increases. The contributions found in the literature for the FFS scheduling problem are summarized in Table 3.1.

Exact algorithms based on branch and bound (B&B) and mixed integer programming (MIP) were found in the literature to solve the problem. However, the results of the computational experiments showed that B&B algorithms become inefficient with more than 20 jobs. Also, the MIP models are impractical

because of their large size even for a small number of jobs and machines. Hence, approximation methods such as TS have been paid attention to recently.

Table 3.1: Summary of Previous Research on FFS Scheduling Problems.

Problem Type	References	Methodology	Problem size
FF2(Pm_1,Pm_2=1)//C_{max}	1. Arthanary and Ramaswany (1971) 2. Gupta (1988)	Branch and Bound (B&B) Hueristic (w.c.p.b)	6-8 jobs 3 - (2 / m)
FF2((Pm_1,Pm_2=1)//C_{max}	Gupta and Tunc (1991)	Heuristic	
FF2(Pm_1,Cm) //C_{max} (Cm= continuous flowshop)	Gupta (1997)	Heuristic (w.c.p.b)	2- (1 / m)
FF2(Pm_1,Pm_2)//C_{max}	1. Brah and Hunsucker (1991) 2. Lee and Vairaktarakis (1994) 3. Rajendran and Chaudhari (1992) 4. Moursli (1995) 5. Guinet et al. (1996) 6. Haouari and Hallah (1997) 7. Soewandi (1998)	B&B Heuristic (w.c.p.b) B&B B&B Heuristic Heuristic Heuristic (w.c.p.b)	≤8 jobs 2- (1/max{m_1,m_2}) ≤ 8 jobs ≤ 20 jobs 2 –(1/max{m_1,m_2})
FF2(1,Pm_2)/ /C_{max} (independent setup is considered)	Li (1997)	Heuristic	
FF2(1,Pm_2)/ /C_{max} (both independent setup and removal items are considered)	Gupta and Tunc (1994)	Heuristic	

Table 3.1: Summary of Previous Research on FFS Scheduling Problems (continued).

Problem Type	References	Methodology	Problem size
FFs(Pm_1,Pm_2,…,Pm_S)//C_{max}	1. Lee and Vairaktarakis (1994)	Heuristic (w.c.p.b)	S-(1/max{m_1,m_2}) - … - (1/max{m_{1-1},m_S}
	2. Moursli (1995)	B&B	≤ 6 jobs for 5 stages
	3. Vignier et al. (1996)	B&B	≤ 15 jobs
	4. Portmann et al. (1998)	B&B and B&B+GA	≤ 15 jobs for 5 stages
	5. Soewandi (1998)	Heuristic (w.c.p.b)	4- 1/max{m_1,m_2} – 1/m_3 for Proc. SP1 10/3- 1/max{m_1,m_2} – 1/3m_3 for Proc. SP2
	6. Ding and Kittchartphayak (1996)	Heuristic	8 jobs
	7. Novicki and Smutnicki (1996)	Heuristic (TS approach)	≤ 500 jobs , 20 machines
	8. Franca et al. (1996)	Heuristic (TS approach)	≤ 50 jobs , 5 machines
	9. Novicki and Smutnicki (1998)	Heuristic (TS approach)	≤ 150 jobs , 60 machines
FF2(Qm_1,Qm_2)//C_{max}	Soewandi (1998)	Heuristic (w.c.p.b)	{1+ (m-1)v_m}/V
FFs(Qm_1,Qm_2,…,Qm_S)//C_{max} (for jobs are identical only)	Verma and Dessouly (1999)	Heuristic	

3.3.3 Applications of Tabu Search (TS) to the Flowshop Scheduling Problem

3.3.3.1 Introduction and Overview

Tabu search is a heuristic designed for finding a near optimal solution for combinatorial problems. It is considered as a metaheuristic (Hertez and Werra 1989, 1990, and Skorin-Kapov and Vakharia, 1993). This heuristic was first proposed by Glover in 1989. It attempts to find a better solution than an initial. A key difference between TS algorithm and other hill-climbing algorithms is that TS is not trapped at local minima. The search process is provided with a mechanism that allows the objective function to deteriorate and, in a controlled way, allows it to escape from local minima.

Researchers have shown that many combinatorial problems are NP-hard; hence, near-optimal solutions are obtained. A heuristic method is often used to find an initial solution which is then improved in an effort to find a near-optimal solution. Basically, the application of TS is characterized by several components such as a move, neighborhood, memory, initial solution, tabu list, aspiration level, and stopping criteria.

A move, a neighborhood, and a tabu list

A *move* is a function that transforms one solution to another. The subset of moves applicable to a given solution generates a collection of solutions called the *neighborhood*. TS begins with an initial solution which may be obtained from a heuristic or from a random generation. At each step, the neighborhood of the current solution is examined in order to find an appropriate neighbor. Typically, there are two fundamental methods to examine an appropriate neighborhood. The first method is to examine the entire neighborhood and select the best neighbor. This method is appropriate for problems with small neighborhoods. The second method, which is useful with large neighborhoods, is to examine a smaller neighborhood determined by some appropriate technique. A trade-off exists between the effort spent in searching the neighborhood and the quality of the neighbor selected. The move that leads to this neighbor is

performed and the resulting solution becomes the new current solution to initiate the next iteration. The search allows for moves that yield solutions inferior to the best solution obtained so far in order to avoid being trapped at a local optimum.

Since the search always chooses the best new movement, it may well fall back into the local minimum from which it previously emerged. At any stage of the process, a tabu list of mutation that the procedure is not allowed to perform is kept. The goal of utilizing the tabu list is to exclude moves that would bring us back to the point where we were at some previous iterations and keep us trapped in a local minimum. To avoid cycling, the reverse of a movement that has been recently performed is forbidden (tabu) and inserted on the top of tabu list. All other entries are pushed down one position and the bottom entry is deleted. In other words, a tabu list is operated as a FIFO strategy. The length of the tabu list is an important parameter. If the number of entries in the tabu list is too small, cycling may occur. Conversely, if the number of entries is too large, the computation time may increase significantly. The tabu list may be of several types such as position of jobs or pairs of jobs that may not be interchanged (Tillard,1990).

Memory

Normally, there are three types of memories: short-term, intermediate, and long-term memories. A fundamental component of the TS algorithm is a short-term strategy called “simple TS” (Glover,1989; Glover, 1990; Werra & Hertz, 1989).

The fundamental memory structure in the simple TS algorithm is the so-called tabu list. As mentioned earlier, each move in a tabu list is memorized after each iteration. The best move is selected among the set of candidates which are not in the tabu list. Normally, a short-term memory is a method that keeps limited track of a search trajectory in order to guide the search out of a local optimum. The functions of intermediate and long-term memories are employed within tabu search to achieve regional intensification and global diversification of the search. When a region of the

solution space produces good solutions, then it is good to intensify the search in that region (intensification). Conversely, instead of inducing the search to focus more intensively on regions that contain good solutions previously found, the long-term memory (diversification) guides the process to regions that markedly contrast with those examined so far.

Aspiration level condition

An improvement can be realized in the TS is due to the fact that too many solutions may be forbidden. An aspiration level is defined as the value of the best schedule obtained so far. The aspiration level provides flexibility to choose good moves by allowing the tabu status of a move to be overridden, after comparing the values of the schedules, if it seems desirable to do so. Criteria for removing the tabu status will be expressed by aspiration level condition.

Stopping criteria

Stopping criteria are rules to stop the search. Some stopping rules are defined such as maximum number of iterations, maximum computation time, maximum CPU time, or the maximum number of iterations have been performed without improving the best solution obtained so far. Figures 3.1 and 3.2 describe the process of the tabu search with short-term memory (Glover, 1990).

3.3.3.2 Review of TS Applications

During the last two decades, the Tabu Search (TS) technique has been found to be a remarkably effective approach to solve combinatorial optimization problems. Barnes and Laguna (1993) reviewed some of the research related to TS applications in production scheduling and provided synthesis of the TS methods that have been employed. Some suggestions for future research were also provided in their study.

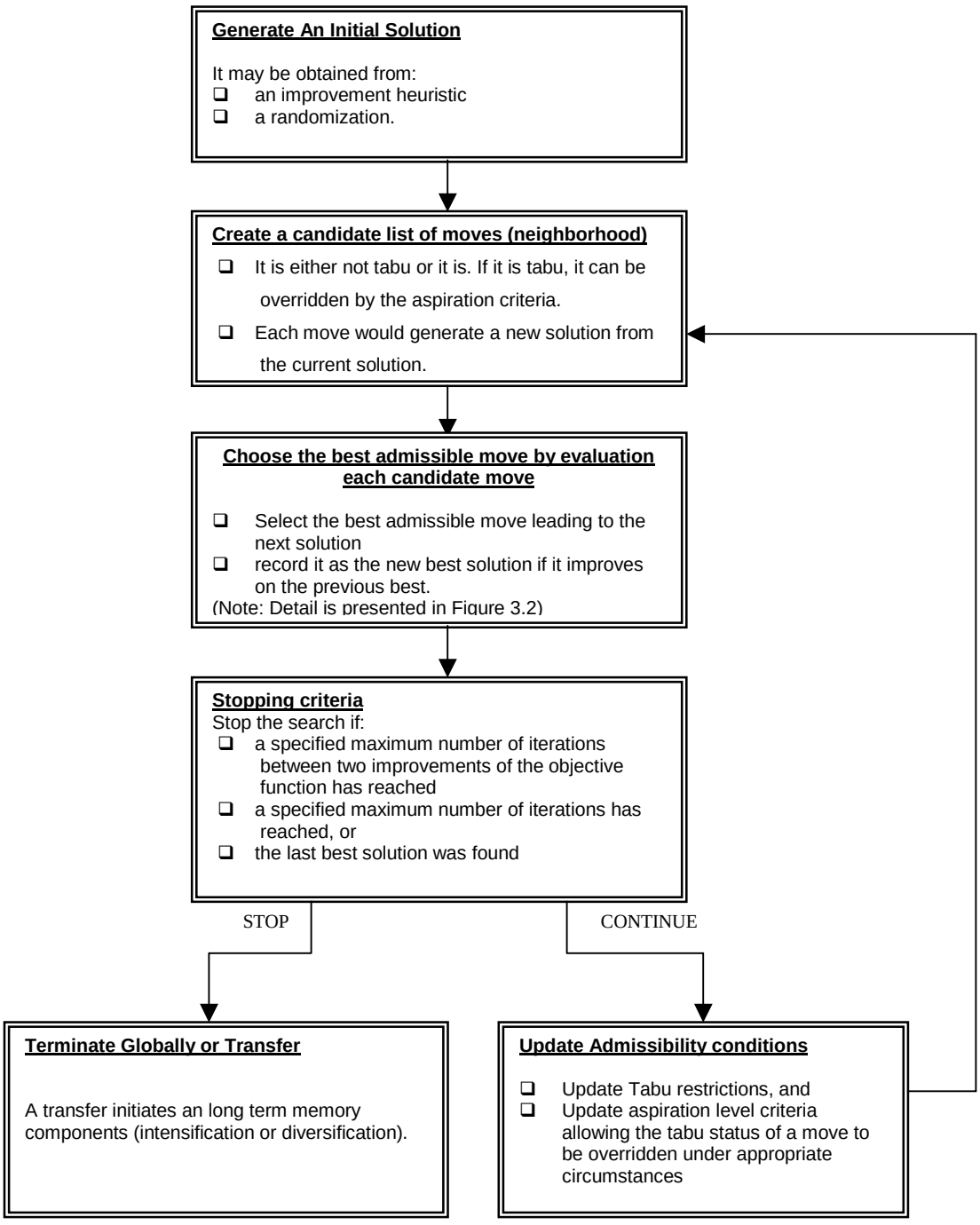

Figure 3.1 : The General Tabu Search Technique

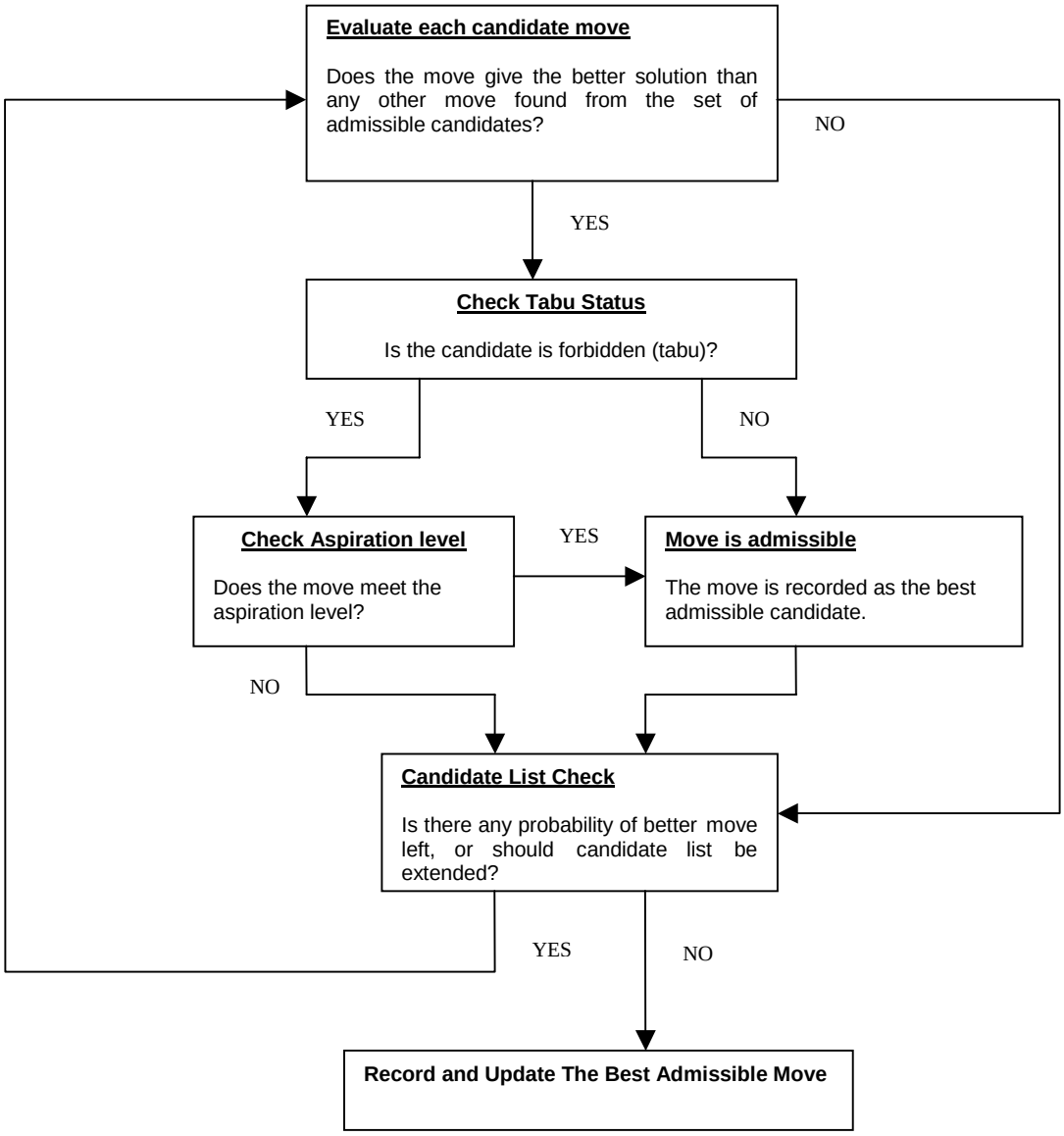

Figure 3.2: Selecting the Best Admissible Move

In 1993, Laguna et al. applied TS to a single machine problem in order to minimize the sum of the setup costs and linear delay penalties when N jobs, arriving at time zero, are to be scheduled for sequential processing on a continuously available machine. Their experimental results showed that the TS heuristic succeeded in finding optimal solutions to all problems (with up to 22 jobs) to which the solutions are known. A fast and easy approximation approach based on the TS technique was developed by Novicki and Smutnicki (1996) for the permutation flowshop problem with the objective of minimizing makespan. Their results showed that the algorithm was effective and could solve problems with up to 500 jobs and 20 machines. Also, Franca et al. (1996) proposed a three-phase heuristic for solving the scheduling problem with identical parallel processors in order to minimize the makespan. The TS algorithm was applied for solving the problem in phase 2 which improves the initial solution obtained from the first phase. They then attempted to further improve the solution in phase 3. The number of jobs and machines that their method could solve within reasonable running time were up to 50 jobs and 5 machines.

The best efforts to apply the TS algorithm for solving large-size FFs problems with identical parallel machines at each stage have been performed by Nowicki and Smutnicki (1998). They developed an algorithm to solve problems with the objective of minimizing makespan. They used their algorithm to solve problems with up to 150 jobs and 60 machines. Another study was done by Norman (1999), who investigated flowshop scheduling problems with both sequence-dependent setup times and finite buffers by applying the TS approach. His findings showed that a TS heuristic procedure can give a good solution for problems with up to 200 jobs and 20 machines.

Because of the reported success of the TS in previous research with similar problems, it has been selected for application in this research.

CHAPTER 4

EXACT ALGORITHM

4.1 Introduction

Even though the flexible flowshop problem with sequence dependent setup times is difficult to solve optimally for large-size problems, an exact procedure using a mathematical programming formulation, is generally accepted for solving small-size problems. There are two main reasons for formulating a mathematical programming model:

- The mathematical programming formulation provides a better understanding of the problem, which will be useful in formulating relaxed problems and in developing heuristic solution procedures.
- Even though existing computing devices cannot solve large problems in an acceptable time, development of these devices is improving with a fast pace. Faster computers are developed, with larger memories, and may be able to solve practical size problems in the near future.

4.2 Mathematical Formulation

A brief description of the problem is reviewed in order to help in understanding the mathematical formulation. The problem involves the scheduling of multiple products in a flexible flowshop environment with sequence dependent setup times ($FFS(Qm_1,Qm_2,\ldots, Qm_S)/s_{ipm}/C_{max}$). In this research, there is only one production line considered. The production line consists of many stages, which may have one or more non-identical (uniform) parallel machines. In each stage, machines can process all products but they differ in their performances, and the machines cannot process a new product until the previous product has been completely finished.

The products have to be manufactured on only one of the machines in each stage, and the processing of products cannot start until the products are completed in the previous stage. Each product, e.g., product *i* of family *j*, requires *PTime*(j,i,s,m) units processing time on machine *m* of stage *s*. Machine setup times are needed between any two products. In this study, it is assumed that setup times are equal for all machines in the same stage when changing from one product to another.

This chapter presents a 0-1 mixed integer programming model with the objective of minimizing makespan for the problem. The model is presented below with a brief explanation of each constraint. Parameters and decision variables used in formulating the model are defined in Table 4.1.

The objective function:

Min E

Constraints:

- *Completion time forcing constraints:*

  This set of constraints ensures that all products are scheduled and the completion time of any product on any machine of the first stage is at least the sum of setup time from idling and processing time required for the product on that machine.

$$FT(j,i,1,m) \geq ch(0,0,j,i,s) + \{PTime(j,i,1,m) \cdot x(j,i,1,m)\} \quad (1)$$

$j = 1,2,\ldots,N$; $i = 1,2,\ldots,f_j$; and $m=1,2,\ldots,m(1)$

Table 4.1: The Notation Used in the Mixed Integer Programming Model

Type of Variables	Notation	Explanation
Decision variables	FTime(j,i,s,m)	Finish time of product *i*, family *j* on machine *m* of stage *s*
	E	The makespan
Binary decision variables	x(j,i,s,m)	= 1 , if product *i*, family *j* is assigned to machine *m* of stage *s* = 0 , otherwise
	w(j,i,q,p,s,m)	= 1 , if product *i*, family *j* immediately precedes product *p*, family *q* on machine *m* of stage *s* = 0 , otherwise
Parameters	*i,p*	Product indices
	j,q	Family indices
	s	Stage index
	m	Machine index
	f_j	The number of products in family *j*
	m(s)	The number of machines in stage *s*
	N	Total number of families
	M(s)	The set of machines in stage *s* ; M(s) = {1,2,..,m(s)}
	S	The number of stages in the production line
	PTime(j,i,s,m)	The processing time of product *i*, family *j* on machine *m* of stage *s*
	ch(j,i,q,p,s)	The number of time units required to changeover from product *i*, family *j* to product *p*, family *q* at stage *s*

- *Stage link constraints:*

Constraints (2) ensure that the completion time of product *i* of family *j* produced on machine *m* in the current stage (stage *s*) must be greater than its completion time in a previous stage (stage *s-1*). The difference must be equal to or greater than the amount of processing time required in the current stage.

$$FTime(j,i,s,m) \geq FTime(j,i,s-1,mp) + \{PTime(j,i,s,m) \cdot x(j,i,s,m)\} \quad (2)$$

$j = 1,2,\ldots,N$, $i = 1,2,\ldots,f_j$; $s = 2,3,\ldots,S$, $m = 1,2,\ldots,m(s)$, and $mp = 1,2,\ldots,m(s-1)$

Setup times are not considered here because the machine in the current stage may be setup for the product while the product is being processed in the previous stage.

- *Constraints about product sequencing on all the S stages:*

$$FTime(j,i,s,m)-FTime(q,p,s,m)-ch(q,p,j,i,s)+(V)(1-w(q,p,j,i,s,m)) \geq \{PTime(j,i,s,m) \cdot x(j,i,s,m)\} \quad (3)$$

$j = 1,2,\ldots,N$, $q = 1,2,\ldots,N$, $i = 1,2,\ldots,f_j$; $p = 1,2,\ldots,f_q$, $s = 1,2,\ldots,S$, $m = 1,2,\ldots,m(s)$, and V is a very large positive number.

If product *p* of family *q* is processed on machine *m* at stage *s* immediately before product *i* of family *j,* then the value of *w(q,p,j,i,s,m)* equals to one. Hence, the completion time of product *i,* family *j* must be greater than the completion time of product *p*, family *q*. The difference must be equal to or greater than the sum of the setup time from product *p,* family *q* to product *i,* family *j* and the required processing time of product *i,* family *j* on that machine.

- *Sequence completion time constraint:*

These constraints are needed to ensure that the makespan is equal to or greater than the completion time of each of the products in the last stage.

$$FT(j,i,S,m) \leq E \quad (4)$$

$j = 1,2,\ldots,N$, $i = 1,2,\ldots,f_j$; and $m = 1,2,\ldots,m(S)$

- *Constraints (5) ensure that each product is processed on exactly one machine in each stage.*

$$\sum_{m=1}^{m(s)} x(j,i,s,m) = 1 \quad (5)$$

j = 1,2,…,N, i = 1,2,…, f_j ; and s = 1,2,…,S

- *Except for the first product, a product scheduled on a machine must be immediately preceded by exactly one different product.*

$$x(q,p,s,m) - w(0,0,q,p,s,m) - \sum_{j=1}^{N}\sum_{i=1}^{fj} w(j,i,q,p,s,m) = 0 \quad (6)$$

q = 1,2,…,N; p = 1,2,…, f_q; s = 1,2,…,S; and m = 1,2,…,m(s)

- *Except for the last product, a product scheduled on a machine must be immediately followed by exactly one product.*

$$x(j,i,s,m) - w(j,i,0,0,s,m) - \sum_{q=1}^{N}\sum_{p=1}^{fj} w(j,i,q,p,s,m) = 0 \quad (7)$$

j = 1,2,…,N; i = 1,2,…,f_j ; s = 1,2,…,S; and m = 1,2,…,m(s)

- *A machine can have exactly one first and one last product:*

$$\sum_{q=1}^{N}\sum_{p=1}^{fj} w(0,0,q,p,s,m) = 1 \quad (8)$$

s = 1,2,…,S; and m = 1,2,…,m(s)

$$\sum_{j=1}^{N}\sum_{i=1}^{fj} w(j,i,0,0,s,m) = 1 \quad (9)$$

s = 1,2,…,S; and m = 1,2,…,m(s)

The above formulation is illustrated using the following simple problem. The problem data are shown below and the computer model is listed in Appendix B.

Number of families: N = 2

Number of stages: S = 3

Number of products: $f_j = 2$; j = 1,2

Number of machines: m(1) = 1, m(2) = 1, and m(3) = 2

Speed of machines at each stage ~ U(0.85,1.15), resulting in the speeds shown in Table 4.2.

Processing time of each product on the standard machine at each stage ~ U(10,50), resulting in the processing times shown in Table 4.3.

Setup time from idling in stage 1 for each product, as a percentage of the processing time ~ U(5%,15%), resulting in the setup times shown in Table 4.4.

Changeover time between two products at each stage (ch(q,p,j,i,s)), as a percentage of the processing time ~U(10%,40%) and ~U(5%,15%) for major and minor setup times respectively, resulting in the times shown in Table 4.5.

Table 4.2: Speeds of Machines at Each Stage

	Speed of Machine	
	m=1	m=2
Stage 1	1.1	-
Stage 2	1.15	-
Stage 3	1.0	0.98

Table 4.3: Processing Time of Each Product at Each Stage on the Standard Machine

Stage	Family	Processing Time of Products	
		Product 1	Product 2
s =1	j = 1	18.28	33.81
	j = 2	48.95	25.1
s =2	j = 1	31.57	28.24
	j = 2	26.09	17.39
s =3	j = 1	23.68	44.87
	j = 2	19.09	49.26

Table 4.4: Setup Time from Idling for Each Product in Stage 1

Family	Setup Time from Idling of products	
	ch(0,0,j,1,1)	ch(0,0,j,1,2)
j = 1	3.89	1.52
j = 2	2.26	3.99

Table 4.5: Changeover Times between Products at each Stage (ch(q,p,j,i,s))

Stage s	Family j	Product i	Changeover time from product p of family q to product i of family j (ch(q,p,j,i,s))			
			Family q=1		Family q=2	
			p=1	p=2	p=1	p=2
1	1	1	0.00	4.24	5.88	4.90
		2	2.49	0.00	6.54	4.52
	2	1	5.33	6.22	0.00	4.20
		2	5.91	5.51	3.48	0.00
2	1	1	0.00	2.96	6.45	10.29
		2	2.20	0.00	11.25	7.87
	2	1	9.39	10.24	0.00	2.41
		2	10.58	6.98	3.07	0.00
3	1	1	0.00	3.22	6.64	10.63
		2	3.13	0.00	10.17	6.15
	2	1	9.79	9.95	0.00	3.46
		2	6.63	11.50	1.65	0.00

The model has 188 constraints, 556 continuous variables, and 96 integer variables for this problem. It is necessary to use a software which can handle a large number of variables. The MPL/CPLEX software was used to solve this problem. The makespan of this solution is 172.32 time units. The optimal product sequences on the different machines and stages are presented below.

Stage 1: Machine1: (1,2)-> (1,1) -> (2,2) -> (2,1)

Stage 2: Machine 1: (1,2)-> (1,1) -> (2,2) -> (1,1)

Stage 3: Machine 1: (1,2)-> (1,1) -> (2,1)

Machine 2: (2,2)

where (j,i) is product i of family j.

Although an optimal solution was obtained for the problem, the computational time was excessive. Attempts to solve larger problems were unsuccessful as they required too much CPU times. Hence, a heuristic algorithm is developed to obtain a near-optimal solution for realistic sized problems.

CHAPTER 5

HEURISTIC ALGORITHMS

The flexible flowshop problem with sequence-dependent setup times is known to be NP-hard (Allahverdi et. al, 1999). In general, the computational effort required to find an optimal solution grows exponentially with the size of the problem. In an effort to find a near optimal solution for problems with average or large sizes, a two-phase algorithm was developed. The first phase consists of a constructive heuristic developed to obtain an initial solution. This heuristic will be termed as the "Flexible Flowshop with Sequence Dependent Setup Times Heuristic" (FFSDSTH). The second phase, referred to as the Tabu Search Heuristic (TSH), uses the well-known Tabu Search meta-heuristic to improve on the solution obtained from the first phase. The algorithm process flow for both phases is shown in Figure 5.1.

The detailed description of the two-phase heuristic is presented in the following three sections. A detailed description of the FFSDSTH is presented in Section 5.1, and is followed by a numerical illustration in Section 5.2. The TSH is described in Section 5.3.

5.1 Phase 1: Obtaining an Initial Solution Using the FFSDSTH Algorithm

The heuristic developed in this phase schedules one family at a time on the machines of the first stage. The algorithm then proceeds by scheduling products to the machines of all other stages. Prior to the presentation of the FFSDSTH algorithm, the notation and variables used are defined.

Notation:

Let

i,p = product indices

j,q = family indices

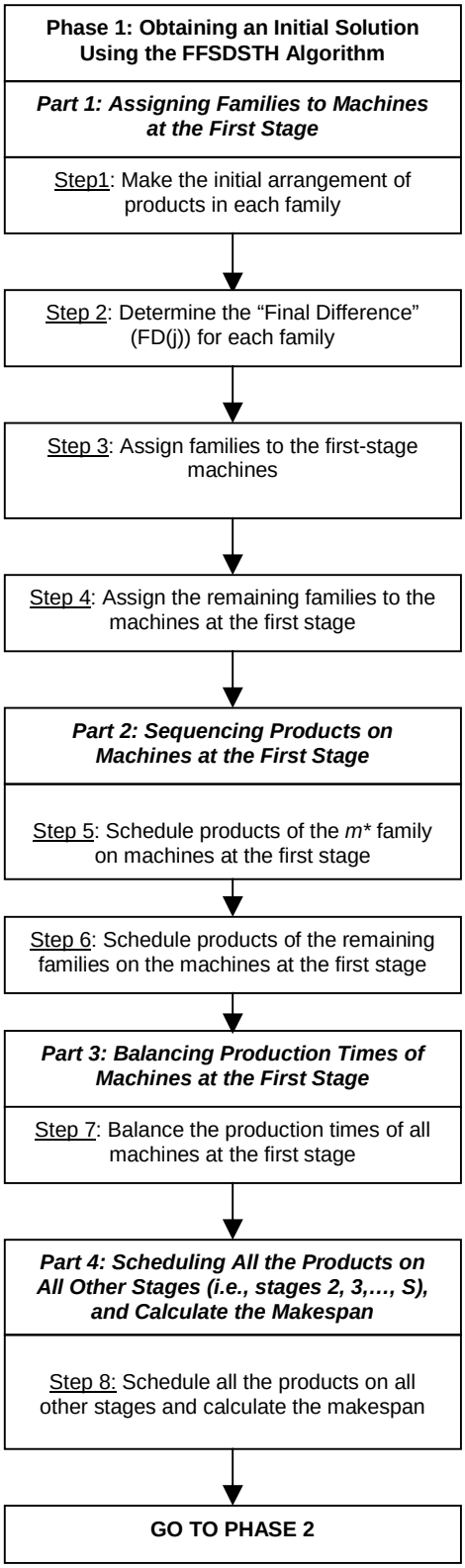

Figure 5.1: A Process Flow of the FFSDSTH and TSH Algorithms

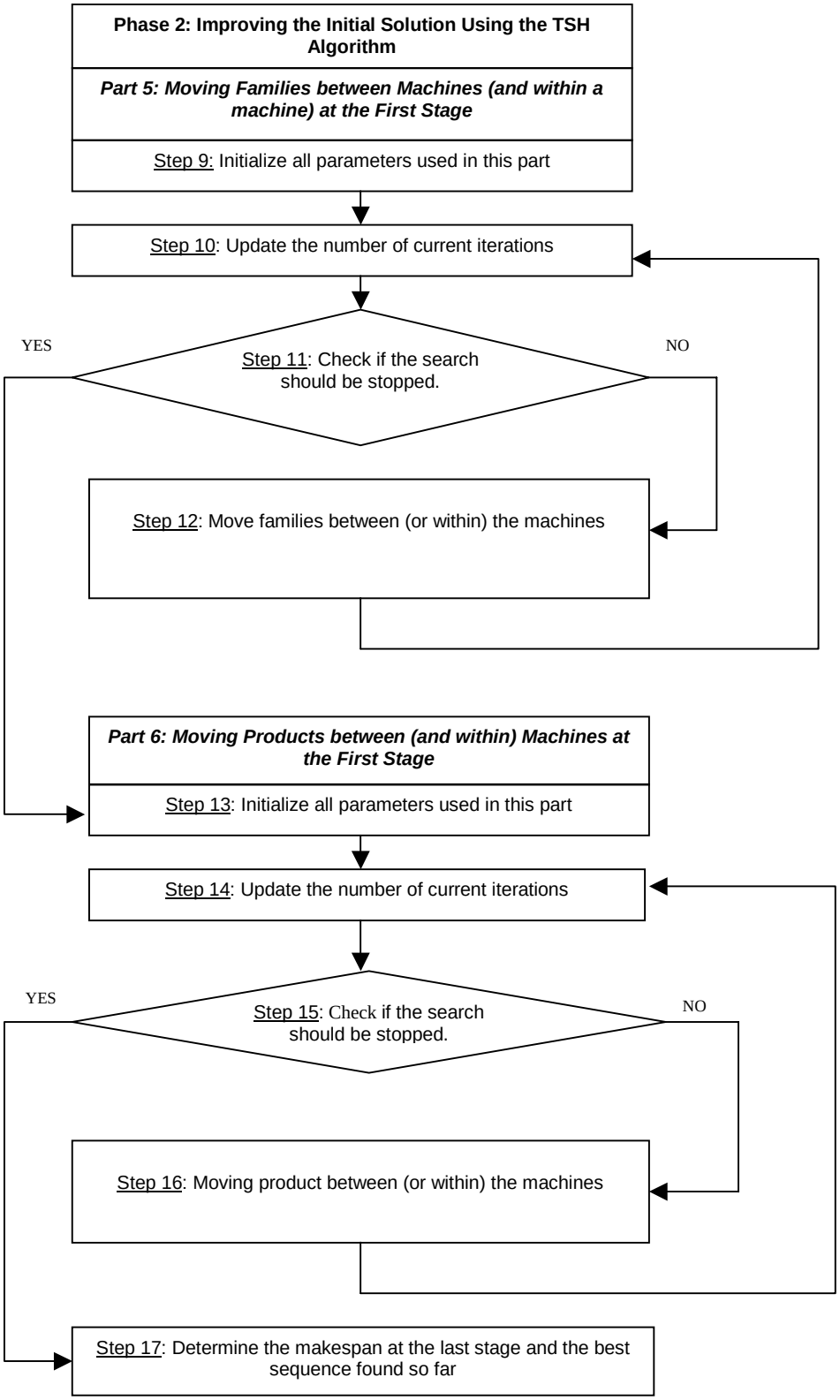

Figure 5.1: A Process Flow of the FFSDSTH and TSH Algorithms (continued)

s	=	stage index
J	=	set of all families; $J = \{1,2,\ldots,N\}$
F_j	=	set of products in family j; $j \in J$
	=	$\{1,2,\ldots,f_j\}$
f_j	=	number of products in family j; $j \in J$
Ψ	=	set of stages in a production line
	=	$\{1,2,\ldots,S\}$
$m(s)$	=	number of machines in stage s; $s \in \Psi$
$M(s)$	=	set of machines in stage s
	=	$\{1,2,\ldots,m(s)\}$
$v_{s,m}$	=	speed of machine m at stage s
ch(q,p,j,i,s)	=	The number of time units required to changeover from product i of family j to product p of family q at stage s
$STime(j,i,s,m)$	=	start time of product i of family j on machine m of stage s. There are 8 possible ways of determining the value of $STime(j,i,s,m)$. A detailed description of these ways is presented in section 5.1.1.
$PTime(j,i,s,m)$	=	processing time of product i *of* family j on machine m of stage s; $j \in J$, $i \in F_j$, $s \in \Psi$, and $m \in M(s)$.
$T(j,i)$	=	processing time of product i *of* family j on the standard machine in stage 1
$FTime(j,i,s,m)$	=	finish time of product i, family j on machine m of stage s. This time is equal to the sum of its start time and processing time.

FTime(j,i,s,m)	=	*STime(j,i,s,m) + PTime(j,i,s,m); j* $\in$ *J*, *i* $\in$ F_j, *s* $\in$ Ψ, and *m* $\in$ *M(s)*
$\lfloor x \rfloor$	=	the largest integer less than or equal to x
avg_proc/prd/mc(j)	=	the average processing time per product per machine for family *j* at the first stage; *j* $\in$ *J*
COT(q,j)	=	the average changeover time from family *q* to family *j; q, j* $\in$ *J* and *q* $\neq$ *j.* The value of *COT(q,j)* is obtained by calculating the average of the total changeover times from all products of family *q* to all products of family *j*.
avg_COT(j)	=	the average changeover time from all other families to family *j; j* $\in$ *J.* The value of *avg_COT(j)* is obtained by dividing the sum of the average changeover times from all other families to family *j* by *N-1*.
W_{1j}	=	the ratio between *avg_proc/prd/mc(j)* and $\min_{q \in J}$ *avg_proct/prd/mc(q); j* $\in$ *J*
W_{2j}	=	the ratio between *avg_COT(j)* and $\min_{q \in J}$ *COT(q); j* $\in$ *J*
*m**	=	the minimum value of *m(s)*; *m** = $\min_{s \in \psi}$ *m(s)*
FD(j)	=	the "final difference" value for family *j; j* $\in$ *J,* calculated as *{W_{1j} x avg_proc/prd/mc(j)} – {W_{2j} x avg_COT (j)}.* This value is used to assign the first *m** families to machines in the first stage at the start of scheduling.
R	=	set of the *m** families with the lowest values of *FD(j)*, |*R*| = *m** and *R* = { f_1, f_2, …, f_{m^*}} where f_j is the family with

the j^{th} lowest *FD* value. These families will be assigned first at the start of the scheduling.

WKL(j) = the total processing time and setup time (or workload) of family *j* in stage 1, $j \in R$, using a standard machine (i.e., speed = 100%). The value of *WKL(j)* is obtained by summing the processing times of all products on a standard machine at the first stage and the setup times of all products using the products order determined in Step 1, as explained in section 5.1.2.

GT = the sum of the total processing and setup times of all families in set *R* at the first stage, i.e.,

$$GT = \sum_{j \in R} WKL(j)$$

avg_GT(1) = The average processing time per machine for families in set *R*, using standard machines at stage 1; *i.e., avg_GT(1) = GT/ m(1)*

num_mc(j) = total number of machines needed to process family *j* in stage 1; $j \in R$. The value of *num_mc(j)* is obtained by simply dividing the total processing time of family *j* at the first stage (*WKL(j)*) by *avg_GT(1)*.

Min_mach(j) = the minimum number of machines needed to process family *j* in stage 1; $j \in R$. This value of is obtained as follows:

$$Min\_mach(j) = \max\{1, \lfloor num\_mc(j) \rfloor\}$$

min_used = the minimum total number of machines needed to process all families in set *R* in stage 1;

min_used = $\sum_{j \in R} Min\_mach(j)$

K = set of shared machines at the first stage. These machines are the slowest *m(1) – min_used* machines. Shared machines are those to which more than one family has been assigned.

quota_time(j) = the limited production time of family *j* on the shared machine; $j \in G$

Prior to the presentation of the FFSDSTH algorithm, the procedure used to determine the start time of a product on a machine is presented below.

5.1.1 Start Time Determination

There are eight possible ways to determine the value of the start time (*STime(j,i,s,m)*) as described below.

5.1.1.1 If *j* = the first family processed on machine *m* at the first stage; $j \in J$, *i* = the first product scheduled in family *j*; $i \in F_j$, and $m \in M(1)$, *then:*

STime(j,i,1,m) = ch(0,0,j,i,1)

5.1.1.2 If *j* = the first family scheduled on machine *m* at the first stage, $i \neq$ the first product in family *j* processed on the machine, then:

STime(j,i,1,m) = FTime(j,p,1,m) + ch(j,p,j,i,1)

where,

p = the product that precedes product *i* on machine *m* in the first stage

and $j \in J$, *i*, $p \in F_j$, $m \in M(1)$

5.1.1.3 If $j \neq$ the first family scheduled, *i* = the first product scheduled in family *j* on machine *m* at the first stage. Then:

$STime(j,i1,m) = FTime(q,p,1,m) + ch(q,p,j,i,1)$

where,

q = the family that precedes family j on machine m of stage s

p = the last product of family q scheduled on machine m of stage s

and $j,q \in J,\ i \in F_j,\ p \in F_q,\ m \in M(1)$

5.1.1.4 If $j \neq$ the first family scheduled, $i \neq$ the first product in j processed on machine m at the first stage. Then:

$STime(j,i,1,m) = FTime(j,p,1,m) + ch(j,p,j,i,1)$

where,

p = the product in family j that precedes product i on machine m at the first stage

and $j \in J,\ i,\ p \in F_j\ m \in M(1)$

5.1.1.5 If j = the first family scheduled, i = the first product in j processed on machine m in stage s: $s \in \{2,3,\ldots,S\}$. Then:

$STime(j,i,s,m) = FTime(j,i,s\text{-}1,mp)$

where, $j \in J,\ i \in F_j\ m \in M(s)$, mp is the machine in stage s-1 on which product i of family j was processed

5.1.1.6 If j = the first family scheduled, $i \neq$ the first product in family j processed on machine m in stage s:$s \in \{2,3,\ldots,S\}$. Then:

$STime(j,i,s,m) = max\ \{FTime(j,p,s,m) + ch(j,p,j,i,s),\ FTime(j,i,s\text{-}1,mp)\}$

where,

p = the product in family j that precedes product i on machine m stage s

$j \in J,\ i,\ p \in F_j$

$m \in M(s)$

mp is defined as above.

5.1.1.7 If $j \neq$ the first family scheduled, i = the first product in family j processed on machine m in stage s:$s \in \{2,3,\ldots,S\}$. Then:

$STime(j,i,s,m) = max\{FTime(q,p,s,m)+ch(q,p,j,i,s), FTime(j,i,s-1,mp)\}$

where,

q = the family that precedes family j on machine m of stage s

p = the last product of family q scheduled on machine m of stage s

and $j,q \in J$, $i \in F_j$, $p \in F_q$, $m \in M(s)$, mp is defined earlier.

5.1.1.8 If $j \neq$ the first family scheduled, $i \neq$ the first product in family j processed on machine m in stage s:$s \in \{2,3,\ldots,S\}$. Then:

$STime(j,i,s,m) = max\{FTime(j,p,s,m)+ch(j,p,j,i,s), FTime(j,i,s-1,mp)\}$

Where,

p = the product in family j that precedes product i on machine m stage s

$j \in J$, i, $p \in F_j$

$m \in M(s)$

mp is defined earlier.

If there is any change in the schedule, then the start time of all products and families affected by the change are recalculated.

5.1.2 A Detailed Description of the FFSDSTH Algorithm

The detailed description of the FFSDSTH is presented below in Parts 1 through 4.

Part 1: Assigning Families to Machines at the First Stage

In order to assign families to machines at the first stage, the algorithm starts by sorting products in an initial order within each family and calculating some production and setup parameters for each family, as detailed in the following steps.

Step 1: Make the initial arrangement of products in each family

Since this problem involves uniform machines, define a machine with a standard speed (speed = 100%) for the first stage. Determine the processing time of each product on this standard machine. For each family, arrange the products as follows:

Calculate for each product the sum of its setup time from idling and its processing time on the standard machine. Select as the first product in the sequence the product with the lowest sum. Then calculate for each remaining product the sum of its setup time from the previous product and its processing time on the standard machine. Selected as the second product in the sequence the product with the lowest sum. Repeat this procedure until all products in each family have been completely ordered. Details of the above procedure are given below.

1.1 Find the first product in the sequence.

Find *i'* with:

$$\min_{i \in F_j} (T(j,i) + ch(0,0,j,i,1);\ j \in J$$

1.2 Update $F_j = F_j \setminus \{i'\}$.

If $F_j = \phi$, update $J = J \setminus \{j\}$. If $J = \phi$, go to Step 2; otherwise, go to Step 1.1.

If $F_j \neq \phi$, go to Step 1.3.

1.3 Find the next product.

Find *i'* with:

$$\min_{i \in F_j} (T(j,i) + ch(j,p,j,i,1)); \; j \in J$$

where *p* is the last product scheduled so far on machine *m* at the first stage.

Then, go to Step 1.2.

Step 2: Determine the "Final Difference" [FD(j)] of each family

This step determines the *"Final Difference" [FD(j)]* of each family, which is used for selecting the families to be scheduled at the start of the schedule. Calculations of some parameters must be made prior to the determination of *FD(j)* as detailed below.

2.1 Calculate *avg_proc/prd/mc(j)*:

$$avg\_proc/prd/mc(j) = \sum_{i=1}^{fj} \sum_{m=1}^{m(1)} PTime(j,i,1,m) / \{m(1) \cdot |F_j|\}$$

for $j = 1,2,\ldots,N$

2.2 Calculate *COT(q,j)*:

COT(q,j) is calculated by averaging the changeover time from all products of family *q* to all products of family *j*.

$$COT(q,j) = \sum_{p=1}^{fq} \sum_{i=1}^{fj} ch(q,p,j,i,1) / \{|F_j| \cdot |F_q|\}$$

$q, j \in J$ and $q \neq j$, $i \in F_j$, $p \in F_q$

2.3 Calculate *avg_COT(j)*:

$$avg\_COT(j) = \frac{\sum_{q=1}^{N} COT(q,j)}{N-1} \quad ; q \neq j, \text{ and } q, j \in J$$

2.4 Calculate *FD(j)* as follows:

$FD(j) = \{W_{1j} \times avg\_proc/prd/mc(j)\} - \{W_{2j} \times avg\_COT(j)\}$

where, $j \in J$

Step 3: Assign families to the first-stage machines

In assigning families to machines, one can either select a machine and assign a family to it or select a family and then assign it to a machine. The latter approach is used here.

In this step two tasks are performed. In the first task families are selected to be assigned to the first-stage machines. In the second, these families are assigned to machines.

3.1 Select *m** families.

The first *m** families are those with the lowest values of *FD(j).* These families will constitute the elements of set *R.* At the start of the schedule, only these *m** families are assigned to the machines in order to reduce idle times of the machines at the stage that has the smallest number of machines.

3.2 Assign the selected families to machines.

Once the families to be scheduled at the start of the schedule have been selected, the assignment of those families to machines is made. The number and speeds of the machines are considered in order to reduce machine idle times at the first stage as much as possible. In this step, these families are assigned, one at a time, to the first-stage machines. There are two cases to be considered.

3.2.1 *Case 1: m(1) = m**

In this case, the algorithm assigns the family with the minimum value of *FD(j)* on the fastest machine, the family with

the second lowest value of *FD(j)* to the second fastest machine, and so on.

3.2.2 *Case 2: m(1) > m**

Since the number of machines is greater than the number of families to be assigned to these machines, each of the *m** families may be processed on one or more than one machines, depending on their total processing times. In addition, some of these families may share a machine with other families. The assignment of the *m** families in this case is described as follows.

3.2.2.1 Calculate the total processing time and setup time (*WKL*(*j*)) for each family *j; j* $\in$ *R* on the standard machine at the first stage. This value is calculated by summing the processing times of all products and the setup times of all products when they have been arranged in the order or sequence specified in Step 1.

$$WKL(j) = \sum_{y=0}^{fj-1} ch(j, iy, iy+1, 1) + \sum_{i=1}^{fj} T(j, i)$$

$j \in R$, i_y, $i_{y+1} \in F_j$

where, i_y = the product in position *y.* If $y = 0$, that means product i_{y+1} is the first product in a sequence, and both i_y and j are equal to 0 when y = 0.

3.2.2.2 Calculate the grand total processing times (*GT*) of all families in set *R.*

$$GT = \sum_{j \in R} WKL(j)$$

3.2.2.3 Calculate the average processing time to be allocated to each machine at the first stage when using the standard machine (*avg_GT(1)*).

$$avg\_GT(1) = \frac{GT}{m\ (1)}$$

3.2.2.4 Calculate the number of machines (*num_mc(j)*), in stage 1, to be assigned to each family in set *R.*

$$num\_mc(j) = \frac{WKL(j)}{avg\_GT(1)}$$

3.2.2.5 Calculate the minimum number of machines *(Min_mach(j))*, needed to process family *j* from set *R* in the first stage:

$$Min\_mach(j) = \max\{1, \lfloor num\_mach(j) \rfloor\} ; j \in R$$

3.2.2.6 Calculate the minimum number of machines (*min_used*) needed to process all families in set *R* at the fist stage:

$$min\_used = \sum_{j \in R} Min\ \_\ mach\ (j)$$

3.2.2.7 Assign the first *m** families to the first *min_used* machines. This procedure starts by assigning family *j* with the lowest *FD(j)* value to the *Min_mach(j)* fastest machines. Then family *q* with the second lowest *FD(q)* value to the next *Min_mach(q)* fastest machines, and

so on. This procedure is then repeated until all the families in set *R* have been scheduled on the first *min_used* fastest machines. For every family *j* with the value of *num_mach(j) –Min_mach(j)* = 0, $j \in R$, update $R = R \setminus \{j\}$ and $J = J \setminus \{j\}$.

3.2.2.8 The remaining families in set *R* need to be scheduled on the *m(1) - min_used* remaining machines. These machines are the shared machines which form the elements of set *K*.

Since the families in remaining set *R* have to share machines, the limited production times (*quota_time(j)*) of these families on the machines must be determined:

$$quota\_time(j) = (num\_mach(j) - Min\_mach(j)) \; x \; avg\_GT(1) \; ; \; j \in R$$

For families not completely scheduled (i.e., those in the remaining set *R*), the assignment of these families starts with the assignment of one family to the fastest shared machine. The procedure is then repeated in a cyclic order, as presented below.

3.2.2.8.1 Find *j'* such that

$$COT(j',q) = \min_{q,j} (COT(j,q)); q, j \in R \text{ and } q \neq j$$

3.2.2.8.2 Schedule family *j'* on the fastest shared machine (e.g., machine *m': m'* $\in$ *K*).

3.2.2.8.3 Update *R* = *R*\{*j'*}, *J*=*J*\ {*j'*}, and *K* = *K* \ {*m'*}

If *R* $\neq$ ϕ and *K* $\neq$ ϕ, go back to Step 3.2.2.8.1,

if *R* $\neq$ ϕ and *K* = ϕ, reset *K* back to its original set value, and go to Step 3.2.2.8.4,

and if *R* = ϕ, go to Step 4.

3.2.2.8.4 Assign the next family, i.e., family *j'* to the next fastest shared machine (i.e. machine *m': m'*$\in$ *K*) where

$$\text{COT (q,j')} = \min_{j} COT(q,j),\ j \in R$$

and *q* is the last family scheduled on machine *m*' so far.

Update *R* = *R*\{*j*}',*J*=*J* \ {*j'*}, and *K* = *K*\ {*m'*}.

If *R* $\neq$ ϕ and *K* $\neq$ ϕ, go back to Step 3.2.2.8.4,

if *R* $\neq$ ϕ and *K* = ϕ, reset *K* back to its original set value, and go back to Step 3.2.2.8.4, and if *R* = ϕ, go to Step 4.

Step 4: Assign the remaining families to the machines at the first stage

In this step, the heuristic selects a machine and then assigns one of the remaining families to that machine. The assignment of the remaining families to the machines at the first stage starts with scheduling

one family to the fastest machine. The family selection procedure consists of finding the family with the lowest sum of the average changeover time from the last family (e.g., family *q*) scheduled on that machine (*COT(q,j))* and the average processing time per product per machine of the family at the first stage (*avg_proc/prd/mc(j)).* This procedure is then repeated in a cyclic order until all remaining families have been assigned to the machines. A description of the procedure is given below:

4.1 Determine the fastest remaining machine (e.g., machine *m*)

4.2 Find family *j'* in the remaining set *J* with:

$$\min_{j \in J} [COT(q,j) + avg\_proc/prd/mc(j)]$$

Where, *q* = the last family on machine *m* of stage 1

4.3 Schedule family j' on machine *m*.

4.4 Update J = J \ {j'} and M(1) = M(1) \ {m}.

4.5 If $M(1) \neq \phi$, and $J \neq \phi$, then go back to Step 4.1.

If $M(1) = \phi$, and $J \neq \phi$, set *M(1)* back to its original set value and go to Step 4.1.

If $J = \phi$, go to Part 2.

Part 2: Sequencing Products on Machines at the First Stage

After all families are assigned to the first-stage machines, the product scheduling is performed. There are two types of product scheduling on these machines: 1) Product scheduling for the first *m** families, and 2) product scheduling for the remaining families. The Earliest Finish Time (EFT) rule was

used to sequence products on machines at the first stage in an attempt to reduce machine idle times. A description of the EFT rule is presented below:

Earliest Finish Time (EFT) Rule

The EFT rule selects from the remaining products the one that yields the earliest finish time. The following procedure is followed to apply this rule when scheduling the products of family *j* on a set of machines at stage *s*.

1. Initialization:

   F_j = set of unscheduled products in family *j*

   MU(j) = set of machines needed to process the products of family *j,*

   $MU(j) \subset M(s)$

2. Scheduling steps:

   2.1 Product *i'* is selected to be processed on the machine with the earliest finish time such that:

$$FT(j,i',s,m') = \min_{i \in F_j, m \in MU(j)} \left\{ STime(j,i,s,m) + \frac{T(j,i,s,m)}{v_{s,m}} \right\}$$

   2.2 Assign product *i'* to machine *m'*.

   2.3 Update the finish time of product *i'* on machine *m'*

$$FTime(j,i',s,m') = \left\{ STime(j,i',s,m') + \frac{T(j,i',s,m')}{v_{s,m'}} \right\}$$

   $F_j = F_j \setminus \{i'\}$. Go back to Step 2.1 until $F_j = \phi$.

Step 5: Schedule products of the first *m** families on machines at the first stage

   5.1 For each family with no shared machine:

   The products of these families are scheduled on the corresponding machines using the EFT rule.

5.2 For other families scheduled on shared machines:

5.2.1 Start with products of the families assigned to the fastest shared machine(s). The families are selected in the order in which they were assigned to the shared machine in Steps 3 and 4 of Part 1.

5.2.2 Define *MU(j)* as the set of machines to which the selected family *j* is assigned, including the shared machine.

5.2.3 Apply the EFT rule in sequencing these products on the machines in *MU(j)*. If the machine selected with the EFT rule is the shared machine, make sure that *quota_time(j)* is not exceeded; otherwise, do not schedule the selected product on the shared machine, remove the shared machine from *MU(j)*, and proceed with the EFT rule.

5.2.4 Update $R = R \setminus \{j\}$. If $R \neq \phi$, update *j* to the following family and go to Step 5.2.2; otherwise, go to Step 6.

Step 6: Schedule products of the remaining families

Each of the remaining families is scheduled on only one machine. The Earliest Finish Time (EFT) rule is used to sequence the products of each of these families on the first-stage machines.

Part 3: Balancing the Production Times of Machines at the First Stage

Step 7: Balance the production times of machines at the first stage.

Balancing the production times of machines at the first stage is performed by moving one or more of the products of a family from the machine with the latest completion time to other machines such that the latest completion time of the first-stage machines is reduced. Balancing

is performed after the assignment of all products to machines at the first stage has been completed. The procedure used to balance the production times of the first-stage machines is presented below:

7.1 Find the machine with the latest completion time (e.g., machine *m'*)

7.2 Remove the last product scheduled on machine *m'*.

7.3 Calculate the latest completion time on each of the machines after scheduling the removed product last within its family if scheduled on the machine; otherwise, last on the machine. Select the one with the smallest updated completion time and the corresponding latest completion time.

7.4 If the latest completion time is improved, perform the product re-schedule and return to Step 7.1; otherwise, do not remove the product from machine *m'*, and go to Step 7.5.

7.5 Repeat Steps 7.1 through 7.4 with the product scheduled before the product used in the last removal attempt. If all attempts have been exhausted, proceed with Part 4.

Part 4: Scheduling All products on All other Stages (i.e., stages 2,3,4,…,S)

After all products are completely assigned to the first-stage machines, the assignment of these products on machines at the succeeding stages needs to be performed. A Look Ahead (LA) rule was developed to sequence the products on machines at stages 2 through *S,* in order to obtain low product finish times and a low makespan. Prior to the presentation of the LA rule, the notation used to explain this rule is presented and is followed by details of the rule. Figure 5.2 shows the flowchart for this rule.

Notation:

i, j = product *i* of family *j,* which just finished processing in the previous stage (stage *s*-1). It is the current product looking for a machine to be processed in stage *s*.

MU(j) = set of machines in stage *s* that are processing products of family *j, MU(j)* $\subset$ *M(s)*

m = the machine in set *MU(j)* that yields the earliest finish time for product *i* of family *j*

m' = the machine in set *M(s)* that yields the earliest finish time for product *i* of family *j.* If machine *m'* is currently processing products of family *q*$\neq$*j,* then denote the last product of family *q,* processed on this machine, as *p'*.

p = a product, if any, of family *q* that is being processed at the previous stage (stage *s*-1).

mp = the machine in stage *s*-1 on which product *p* of family *q* is processed

DST(q,p) = the delay in the start time of product *p* of family *q* when it is scheduled after product *i* on machine *m*'. The value of the DST(q,p) is the difference between the start time of product *p* when it is scheduled after product *i* and the start time when it is scheduled directly after product *p*' on machine *m*'. This value is calculated as follows:

DST(q,p) = max {0, FTime(j,i,s,m') + ch(j,i,q,p,s,) –

max{*FTime(q,p,s-1, mp), FTime(q,p',s,m')* + *ch(q,p',q,p,s)*}}.

RFT(j,i) = the reduction in the finish time of product *i* of family *j* when it is processed on machine *m'* instead of machine *m*.

= FTime(j,i,s,m) – FTime(j,i,s,m').

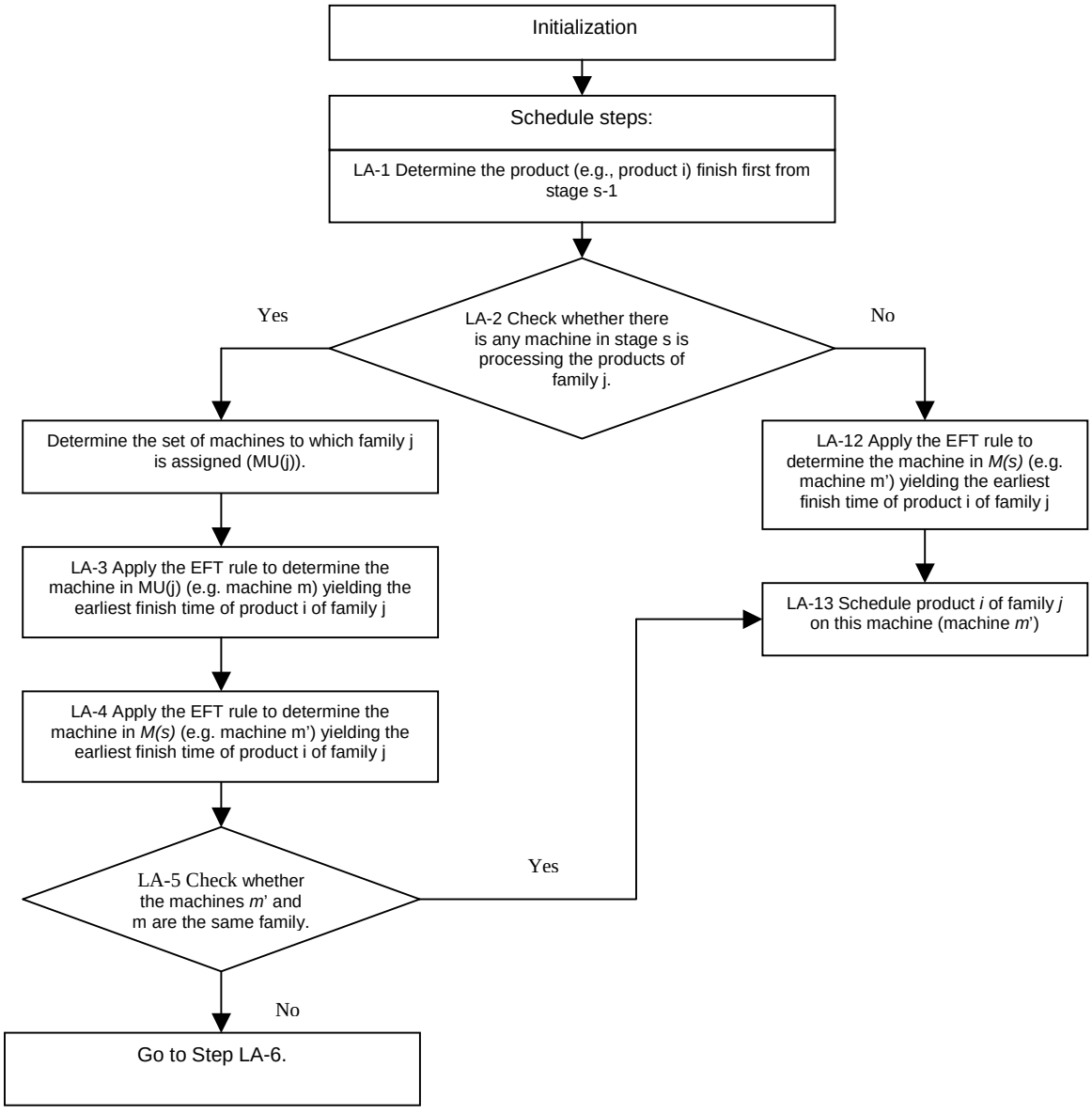

Figure 5.2: Flowchart of the Look Ahead Rule

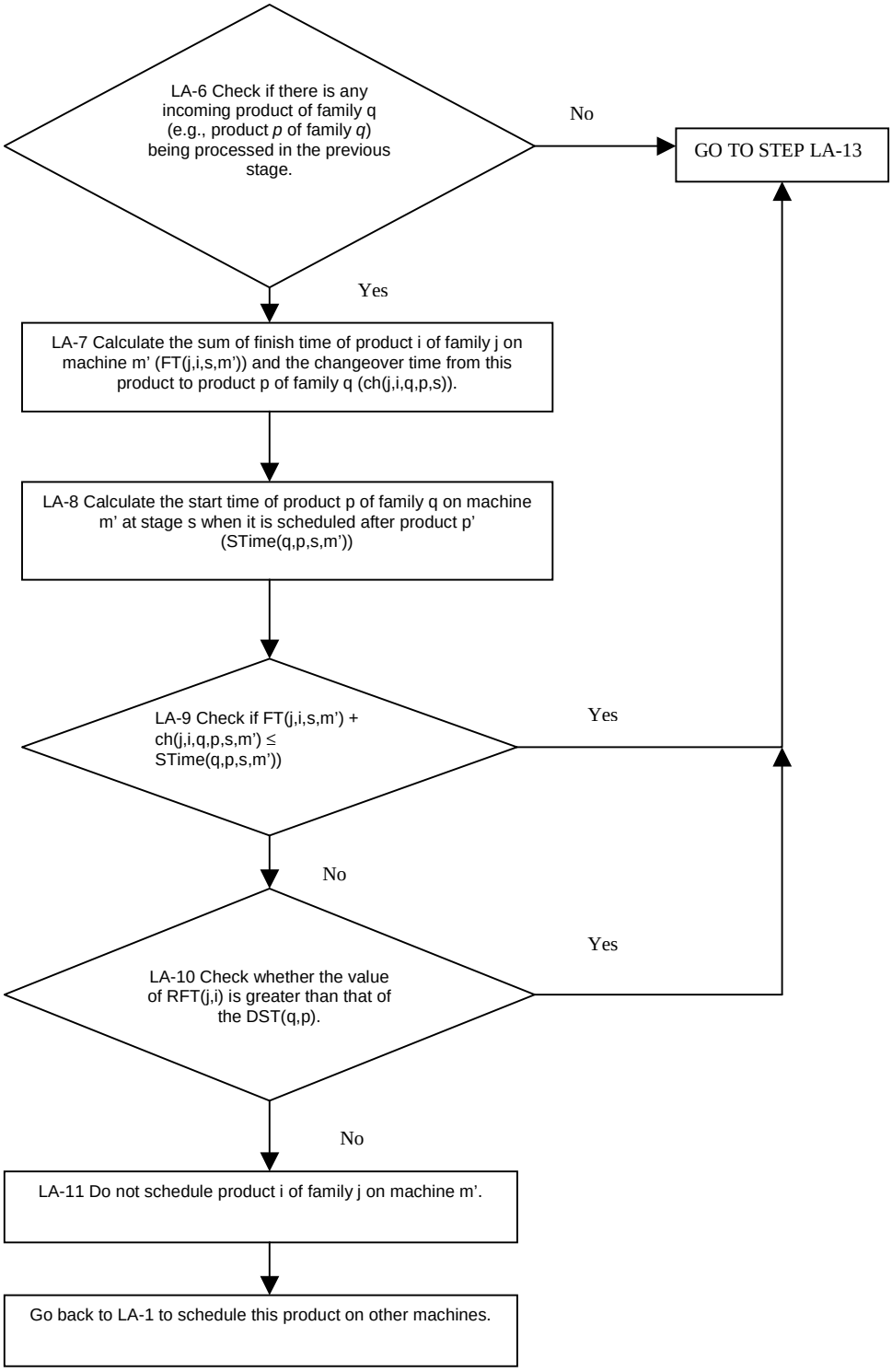

Figure 5.2: Flowchart of the Look Ahead Rule (continued)

Look Ahead (LA) Rule

The LA rule is applied when a product from a certain family (e.g., product *i* from family *j*) has finished processing in a previous stage (stage *s-1; s* >1). The algorithm starts by using the EFT rule to determine the best machine, e.g. machine *m',* for this product which yields the earliest product finish time. Th LA rule then checks if the product that precedes product *i* on machine *m*' is from the same family. If true, then product *i* is scheduled on machine *m'* as soon as it becomes available. Otherwise, the rule checks if there is an incoming product of family *q* from the previous stage (e.g., product *p* of family *q*) to be processed on machine *m'* in the near future (i.e., before time Γ where Γ is equal to the finish time of product *i* on machine *m',* plus the changeover time to product *p*). If not true, this rule schedules product *i* of family *j* on machine *m'* as soon as the machine becomes available. Otherwise, the rule schedules product *i* of family *j* on machine *m'* if either of the following conditions is true:

1. The scheduling of product *i* of family *j* on machine *m'* does not delay the start time of the incoming product of family *q.* In other words, product *i* of family *j* can be scheduled on machine *m'* if the value of DST(q,p) is equal to zero. This results in an earlier finish time of product *i* by FTime(j,i,s,m) – FTime(j,i,s,m') time units.
2. The amount of RFT(j,i) is greater than that of DST(q,p). For this condition, the machine idle time would be reduced by RFT(j,i) – DST(q,p) time units.

As described above, the LA rule tries to reduce the machine idle time. The detailed procedure for the LA rule is given below.

Initialization:

Let H = the set of products arranged in non-decreasing order of finish times from machines in stage *s* -1, *s* >1.

Scheduling steps.

LA-1 Let *i* be the next unscheduled product in set H.

LA-2 Check whether there is any machine in stage *s* processing products from the same family as product *i* (i.e., from family *j*). If true, determine the set of the machines in stage *s* processing the products of family *j (MU(j))* and go to LA-3. If no machine is processing products of this family, go to LA-12.

LA-3 Apply the EFT rule to determine machine *m*, $m \in MU(j)$, that yields the earliest finish time for product *i,* family *j.*

LA-4 Apply the EFT rule to determine machine *m'*, $m' \in M(s)$, which yields the earliest finish time of product *i,* family *j.*

LA-5 If machines *m* and *m*' are the same machine, go to LA-13; otherwise, go to LA-6.

LA-6 Check if there is any product of family *q* (e.g., product *p*) being processed in the previous stage. If yes, go to LA-7; otherwise, go to LA-13.

LA-7 Calculate the sum of the finish time of product *i,* family *j* on machine *m*' (*FTime(j,i,s,m'))* and the changeover time from this product to product *p* of family *q* (*ch(j,i,q,p,s))*.

LA-8 Calculate the start time of product *p* of family *q* on machine *m'* of stage *s* when scheduled after product *p*':

STime(q,p,s,m') = max {*FTime(q,p,mp,s-1), FTime(q,p',s,m')* + *ch(q,p',q,p,s)*}.

LA-9 Compare the time in LA-7 (i.e., FTime(j,i,s,m') + ch(j,j,q,p,s')} to that in LA-8 (i.e., STime(q,p,s,m'))

If FT(j,i,s,m') + *ch(j,j,q,p,s)*} ≤ STime(q,p,s,m'), go to LA-13; otherwise, go to LA-10.

LA-10 Check whether the value of RFT(j,i) = FTime(j,i,s,m) – FTime(j,i,s,m') is greater than that of DST(q,p). If yes, go to LA-13; otherwise, go to LA-11.

LA-11 Do not schedule product *i* of family *j* on machine *m*'. Go back to LA-1 (i.e., repeat this procedure until the product is scheduled on a machine in this stage).

LA-12 Apply the EFT rule to determine machine *m'*, *m'* ∈ *M(s)*, that yields the earliest finish time for product *i,* family *j.*

LA-13 Schedule product *i* of family *j* on machine *m*'.

The steps for Part 4 are given below.

Step 8: Schedule all products on all other stages (i.e., stage 2, 3, … , S) and calculate the makespan

8.1 Set *s* = 2.

8.2 Set H = the set of products arranged in non-decreasing order of finish times from machines in stage *s*-1.

8.3 Schedule the first product (e.g., product *i*) in set H on one of the machines of stage s using the LA rule.

8.4 Update H = H \ *{i}.* If H ≠ ϕ, go back to Step 8.3. If H = ϕ, update *s* = *s* + 1. If *s* ≤ *S*, go to Step 8.2; otherwise, calculate the makespan and go to Phase 2.

5.2 Illustration of the FFSDSTH Algorithm

To demonstrate how this algorithm works, the following problem was generated and will be used as an example. The problem data are as follows.

Number of families: $N = 4$

Number of stages: $S = 3$

Number of products: $f_j = 3$; $j = 1,2,3,4$

Number of machines: $m(1) = 3$, $m(2) = 2$, and $m(3) = 2$

Speed of machines at each stage ~ U(0.85,1.15), resulting in the speeds shown in Table 5.1.

Processing time of each product on the standard machine at each stage ~ U(10,50), resulting in the processing times shown in Table 5.2.

Setup time from idling in stage 1 for each product in terms of percentage of processing time ~ U(5%,15%), resulting in the setup times shown in Table 5.3.

Changeover time between two products at each stage (ch(q,p,j,i,s)) in terms of percentage of processing time ~U(10%,40%) and ~U(5%,15%) for major and minor setup times respectively, resulting in the times shown in Table 5.4.

Table 5.1: Speeds of Machines at Each Stage

	Speed of Machine		
	1	2	3
Stage 1	1.1	1.08	0.95
Stage 2	1	0.93	-
Stage 3	1.06	1.00	-

Table 5.2: Processing Time of Each Product at Each Stage on the Standard Machine

Stage	Family	Processing time of Products		
		Product 1	Product 2	Product 3
s =1	j = 1	47.68	18.19	26.55
	j = 2	34.72	31.58	33.43
	j = 3	21.02	27.71	32.58
	j = 4	43.13	16.06	23.36
s =2	j = 1	23.74	11.07	33.01
	j = 2	11.94	11.31	16.59
	j = 3	14.99	43.76	25.47
	j = 4	36.55	33.76	33.46
s =3	j = 1	32.47	24.8	28.59
	j = 2	48.49	34.25	22.76
	j = 3	33.32	12.87	20.25
	j = 4	48.77	27.88	17.74

Table 5.3: Setup Time from Idling for Each Product in Stage 1

Family	Setup time from Idling of Products		
	ch(0,0,j,1,1)	ch(0,0,j,2,1)	ch(0,0,j,3,1)
j = 1	5.97	6.27	4.53
j = 2	7.48	6.29	7
j = 3	5.75	7.37	5.74
j = 4	5.93	6.16	5.54

Table 5.4: Changeover Times between Products at each Stage (ch(q,p,j,i,s))

Stage s	Family j	Product i	Changeover time from product *p* of family *q* to product *i* of family *j* (ch(q,p,j,i,s))											
			Family q = 1			Family q = 2			Family q = 3			Family q = 4		
			p =1	*p* = 2	*p* = 3	*p* =1	*p* = 2	*p* = 3	*p* =1	*p* = 2	*p* = 3	*p* =1	*p* = 2	*p* = 3
1	1	1	-	4.19	4.28	11.06	8.60	6.51	6.43	7.19	7.82	10.61	10.91	8.94
		2	1.60	-	1.93	10.98	9.00	8.90	9.19	11.77	10.31	9.05	11.15	8.65
		3	4.27	4.13	-	6.26	8.06	11.49	9.93	11.73	8.01	8.09	8.20	6.48
	2	1	7.80	6.23	10.05	-	3.24	3.91	10.03	11.42	11.21	10.23	8.97	6.88
		2	8.32	10.58	9.33	2.34	-	3.66	7.90	11.77	9.71	7.37	9.96	7.98
		3	9.25	10.85	11.41	2.27	4.00	-	6.01	11.02	10.41	9.19	9.32	10.21
	3	1	8.74	9.47	6.62	8.57	9.27	6.22	-	1.97	2.02	6.15	6.94	6.02
		2	8.11	7.73	6.51	10.37	9.94	11.62	2.95	-	2.90	10.89	11.24	11.08
		3	9.44	10.27	6.55	10.83	11.19	7.79	1.57	3.52	-	6.42	11.22	11.25
	4	1	6.35	6.94	10.08	11.30	10.59	10.22	8.90	10.95	10.99	-	4.35	4.47
		2	10.64	11.63	10.19	10.78	8.63	11.82	10.32	10.77	6.24	1.65	-	3.91
		3	9.78	6.57	10.74	9.97	10.73	6.26	6.21	6.98	10.22	1.82	4.30	-
2	1	1	-	3.09	2.82	7.68	9.31	8.18	9.46	11.37	11.07	8.74	7.38	9.67
		2	2.34	-	3.17	7.28	7.77	11.12	8.51	9.21	9.82	10.95	10.45	7.02
		3	4.42	4.24	-	10.17	6.84	7.02	7.54	6.90	8.54	11.93	11.60	7.76
	2	1	10.52	11.17	7.61	-	3.83	3.89	8.74	10.64	6.23	10.63	11.63	11.75
		2	6.06	6.22	8.16	3.94	-	3.82	9.93	6.34	7.45	8.5	9.03	8.81
		3	11.98	10.14	11.57	2.40	4.18	-	9.76	8.92	8.37	10.14	10.79	10.88

Table 5.4: Changeover Times between Products at each Stage (ch(q,p,j,i,s)) (continued)

Stage s	Family j	Product i	Changeover time from product *p* of family *q* to product *i* of family *j* (ch(q,p,j,i,s))											
			Family q = 1			Family q = 2			Family q = 3			Family q = 4		
			p =1	*p* = 2	*p* = 3	*p* =1	*p* = 2	*p* = 3	*p* =1	*p* = 2	*p* = 3	*p* =1	*p* = 2	*p* = 3
2	3	1	7.72	7.16	8.13	6.83	10.59	6.60	-	2.98	2.94	8.50	7.36	9.26
		2	6.33	9.13	8.63	7.64	11.59	7.33	3.85	-	3.41	8.74	8.11	9.61
		3	9.38	8.99	11.18	11.39	11.83	6.05	2.19	3.72	-	10.06	11.85	10.46
	4	1	6.06	7.80	7.56	6.65	7.31	11.40	8.24	7.47	10.48	-	2.78	3.64
		2	10.21	7.30	10.16	7.77	9.52	8.01	11.94	9.07	8.53	4.36	-	1.59
		3	11.66	10.94	9.82	8.07	10.57	8.57	9.20	11.04	10.50	4.36	3.92	-
3	1	1	-	3.27	2.91	10.82	8.55	9.68	9.74	6.89	9.50	10.27	6.64	11.08
		2	3.14	-	1.76	11.00	6.83	7.14	11.08	10.84	9.09	6.02	6.53	9.11
		3	3.52	4.14	-	10.12	7.05	11.74	8.52	8.91	10.81	6.98	6.50	11.94
	2	1	10.42	7.19	11.54	-	2.33	3.18	8.34	7.27	7.37	11.98	11.65	10.44
		2	10.74	10.76	8.35	1.65	-	2.58	6.83	8.96	8.48	6.12	9.77	8.64
		3	10.8	10.31	7.99	3.93	3.47	-	6.52	11.73	6.95	10.44	6.22	10.09
	3	1	10.04	10.93	10.84	10.04	10.54	9.06	-	3.27	3.88	11.99	6.00	8.50
		2	9.44	8.38	6.11	8.76	11.74	7.54	3.73	-	2.67	11.05	10.06	11.90
		3	11.66	11.83	6.11	8.45	8.41	7.13	2.26	1.76	-	6.34	9.52	9.84
	4	1	6.82	8.46	9.62	9.43	10.58	7.13	9.38	6.37	8.69	-	4.10	3.69
		2	10.38	9.65	11.18	8.22	10.36	8.77	6.22	11.26	8.83	2.75	-	3.44
		3	8.96	9.85	10.48	8.81	9.82	7.76	11.02	6.93	8.32	3.13	3.87	-

Part 1: Assigning Families to Machines at the First Stage

Step 1: Make the initial arrangement of products in each family

1.1 Find the first product.

j = 1: ch(0,0,j,1,1) + T(j,1) = 5.97 + 47.68 = 53.65

ch(0,0,j,2,1) + T(j,2) = 6.27 + 18.19 = 24.46

ch(0,0,j,3,1) + T(j,3) = 4.53 + 26.55 = 31.08

Then, the first product of the sequence in this family is product 2.

1.2 Update $F_1 = F_1 \backslash \{2\} = \{1,3\}$

1.3 Find the next product.

ch(1,2,1,1,1) + T(1,1) = 4.19 + 47.68 = 51.87

ch(1,2,1,3,1) + T(1,3) = 4.13 + 26.55 = 30.68

The second product of the sequence in this family is product 3.

Since there is only one product left, the last product in the products sequence of family 1 is product 1. This procedure is repeated with families 2, 3 and 4, resulting in the following sequences:

Products sequence for family 1 is 2--> 3--> 1.

Products sequence for family 2 is 2--> 3--> 1.

Products sequence for family 3 is 1--> 2--> 3.

Products sequence for family 4 is 2--> 3--> 1.

Step 2: Determine the “Final Diffrence” [FD(j)] of each family.

2.1 Calculate *avg_proc/prd/mc(j); j = { 1,2,3,4}*

avg_proc/prd/mc(1) = {(47.68 + 18.19 + 26.55) x

(1/1.1+1/1.08 +1/0.95)} /(3x3)

= 29.65

avg_proc/prd/mc(2) = {(34.72 + 31.58 + 33.43) x
(1/1.1 + 1/1.08 +1/0.95)} /(3x3)
= 32.00

avg_proc/prd/mc(3) = {(21.02 + 27.71 + 32.58) x
(1/1.1 + 1/1.08 +1/0.95)} /(3x3)
= 26.09

avg_proctime/prd/mc(4) = {(43.13 + 16.06 + 23.36) x
(1/1.1 + 1/1.08 +1/0.95)} /(3x3)
= 26.48

2.2 Calculate *COT(q,j)*

COT(2,1) = {11.06 +8.6 +6.51+10.98+9+8.9+6.26+8.06+11.49} /(3x3)
= 8.98

Using the same procedures, the following values are obtained.

COT(3,1) = 9.15, *COT(4,1)* = 9.12,

COT(1,2) = 9.31, *COT(3,2)* = 9.94,

COT(4,2) = 8.90, *COT(1,3)* = 8.16,

COT(2,3) = 9.53, *COT(4,3)* = 9.02,

COT(1,4) = 9.21, C*OT(2,4)* = 10.03,

COT(3,4) = 9.06

2.3 Calculate *avg_COT(j)*

avg_COT(1) = (8.98 + 9.15+9.12) / 3 = 9.08

avg_COT(2) = 9.38

avg_COT(3) = 8.90

avg_COT(4) = 9.43

2.4 Calculate *FD(j)*

The minimum value of an average processing time per product per machine at the first stage is *avg_proc/prd/mc(3)* = 26.09, and the minimum value of the average changeover time at the first stage is *avg_COT(3)* = 8.90. Then, the FD(j) values are obtained as follows.

FD(1) = $(29.65)^2/26.09 – (9.08)^2/8.9 = 24.43$

FD(2) = $(32.00)^2/26.09 – (9.38)^2/8.9 = 29.36$

FD(3) = $(26.09)^2/26.09 – (8.90)^2/8.9 = 17.19$

FD(4) = $(26.48)^2/26.09 – (9.43)^2/8.9 = 16.88$

Step 3: Assign some families to the first-stage machines.

3.1 *m(s*)* = 2, *R* = {4,3}

3.2 Since *m(1) > m(s*)*, then case 2 is applied.

3.2.2.1 *WKL(4)* = (6.16 + 4.3+ 4.47) + (43.13+ 16.06+23.36)
= 97.48 time units

WKL(3) = (5.75 + 2.95+ 3.52) + (21.02+ 27.71+ 32.58)
= 93.53 time units

3.2.2.2 *GT* = 97.48 + 93.53 = 191.01 time units

3.2.2.3 *avg_GT(1,m)* = 191.01/3 = 63.67 time units

3.2.2.4 *num_mc(4)* = 97.48/63.67 = 1.53 machines

num_mc(3) = 93.53/63.67 = 1.47 machines

3.2.2.5 *min_mach(4)* = 1 machine

min_mach(3) =1 machine

3.2.2.6 *min_used* = 1+ 1 = 2 machines

3.2.2.7 Assign family 4 to machine 1, and assign family 3 to machine 2.

3.2.2.8 R = {4,3}

K= {3}

quota_time(4) = (1.53 – 1) x 63.67 = 33.75 time units

quota_time(3) = (1.47 – 1) x 63.67 = 29.92 time units

3.3.2.8.1 – 3.3.2.8.4 COT(3,4) = 9.06

COT(4,3) = 9.02

Since the minimum value is 9.02, schedule family 4 on the shared machine first (i.e., machine 3) and then family 3. Go to Step 4.

Step 4: Assign the remaining families to the machines at the fist stage

4.1 Set M(1) = {1,2}. The fastest machine in this set is 1.

4.2 The candidate families are families 1 and 2 (set J = {1,2}).

Calculate:

COT(4,1) + avg_proctime/prd/mc(1) = 9.12 + 29.65

= 38.77

COT(4,2) + avg_proctime/prd/mc(2) = 9.38 + 32.00

= 41.38

4.3 Since the minimum value is 38.77, then schedule family 1 on machine 1.

4.4 Set J = {2} and M(1) = {2}.

4.1 The fastest remaining machine is 2.

4.2 Since family 2 is the last family to be scheduled, it is assigned to machine 2.

Figure 5.3 shows the assignment of families to the machines at the first stage.

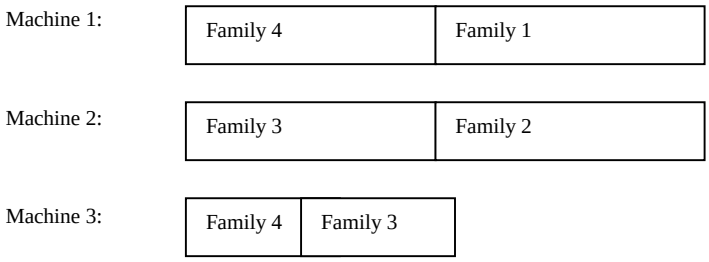

Figure 5.3: The Assignment of all Families to the First-Stage Machines

Part 2: Sequencing products on machines at the first stage

Step 5: Schedule products of the first m^* families (i.e., $m^* = 2$)

Case 5.2 is applied.

5.2.1 Since family 4 is the first family scheduled on the shared machine (i.e., machine 3), its products are sequenced first.

5.2.2 $MU(4) = \{1,3\}$

5.2.3 Apply the EFT rule to schedule the products of family 4.

Scheduling of the first product:

$FTime(4,1,1,1)$ = ch(0,0,4,1,1) + PTime(4,1,1,1)

= 5.93+43.13/1.1 = 45.14

$FTime(4,2,1,1)$ = ch(0,0,4,2,1) + PTime (4,2,1,1)

= 6.16 + 16.06/1.1 = 20.76

$FTime(4,3,1,1)$ = ch(0,0,4,3,1) + PTime (4,3,1,1)

= 5.54 + 23.36/1.1 = 26.78

$FTime(4,1,1,3)$ = ch(0,0,4,1,1) + PTime (4,1,1,3)

= 5.93+43.13/0.95 = 51.33

FTime(4,2,1,3) = ch(0,0,4,2,1) + PTime (4,2,1,3)

= 6.16 + 16.06/0.95 = 23.06

FTime(4,3,1,3) = ch(0,0,4,3,1) + PTime (4,3,1,3)

= 5.54 + 23.36/0.95 = 30.13

Since *FTime(4,2,1,1)* is the minimum value, schedule product 2 of family 4 on machine 1.

Scheduling the next products of family 4.

FTime(4,1,1,1) = FTime(4,2,1,1) + ch(4,2,4,1,1) + PTime (4,1,1,1)

= 20.76 + 4.35 + 43.13/1.1 = 64.32

FTime(4,3,1,1) = FTime(4,2,1,1) + ch(4,2,,4,3,1) + PTime (4,3,1,1)

= 20.76 +4.30 + 23.36/1.1 = 46.30

FTime(4,1,1,3) = ch(0,0,4,1,1) + PTime (4,1,1,3)

= 5.93+43.13/0.95 = 51.33

FTime(4,3,1,3) = ch(0,0,4,3,1) + PTime (4,3,1,3)

= 5.54 + 23.36/0.95 = 30.13

From the above calculations, schedule product 3 of this family on machine 3. Since machine 3 is the shared machine, then go back to Step 5.2.4, check whether the limited processing time of family 4 (quota_time(4)) on the shared machine is not exceeded, as detailed below.

quota_time(4) = (1.53 - 1) x (63.67) = 33.75 time units.

FTime(4,3,1,3) = 30.13 time units

Since the value of quota_time(4) is greater than that of FTime(4,3,1,3), product 3 of family 4 is scheduled on machine 3,

and this machine can still be considered to process the remaining products of family 4.

Scheduling of the last product of family 4.

FTime(4,1,1,1) = *FTime(4,2,1,1)* + ch(4,2,4,1,1) + PTime(4,1,1,1)

= 20.76 + 4.35 + (43.13/1.1)

= 64.32

FTime(4,1,1,3) = *FTime(4,3,1,3)* + ch(4,3,4,1,1) + PTime(4,1,1,3)

= 30.13 + 4.47 + (43.13/0.95)

= 80.00

Then, schedule product 1 of family on machine 1.

5.2.4 Update $R = R \setminus \{4\} = \{3\}$, and go back to Step 5.2.2 in order to schedule the products of family 3, as presented below.

Scheduling of the products of family 3 to the machines at the first stage.

5.2.2 *MU*(3) = {2,3}

5.2 3 Apply the EFT rule to schedule the products of this family.

Scheduling of the first product of family 3.

FTime(3,1,1,2) = ch(0,0,3,1,1) + PTime(3,1,1,2)

= 5.75 + 21.02/1.08 = 25.21

FTime(3,2,1,2) = ch(0,0,3,2,1) + PTime(3,2,1,2)

= 7.37 + 27.71/1.08 = 33.03

FTime(3,3,1,2) = ch(0,0,3,3,1) + PTime(3,3,1,2)

= 5.74 + 32.58/1.08 = 35.91

FTime(3,1,1,3) = *FTime(4,3,1,3)*+ch(4,3,3,1,1) + T(3,1,1,3)

= 30.13 +6.02 + 21.02/0.95 = 58.28

FTime(3,2,1,3) = *FTime(4,3,1,3)*+ch(4,3,3,2,1) + T(3,2,1,3)

= 30.13+11.08 + 27.71/0.95 = 70.38

FTime(3,3,1,3) = *FTime(4,3,1,3)*+ch(4,3,3,3,1) + T(3,3,1,3)

= 30.13+11.25 + 32.58/0.95 = 75.67

From the above calculations, schedule product 1 of family 3 on machine 2. Scheduling the remaining products of family 3 is continued as follows.

FTime(3,2,1,2) = *FTime(3,1,1,2)*+ ch(3,1,3,2,1) + PTime(3,2,1,2)

= 25.21+2.95 + 27.71/1.08 = 53.82

FTime(3,3,1,2) = *FTime(3,1,1,2)*+ ch(3,1,3,3,1) + PTime(3,3,1,2)

= 25.21+1.57 + 32.58/1.08 = 56.95

FTime(3,2,1,3) = *FTime(4,3,1,3)*+ch(4,3,3,2,1) + PTime(3,2,1,3)

= 30.13+11.08 + 27.71/0.95 = 70.38

FTime(3,3,1,3) = *FTime(4,3,1,3)*+ch(4,3,3,3,1) + PTime(3,3,1,3)

= 30.13+11.25 + 32.58/0.95 = 75.67

So, schedule product 2 of family 3 on machine 2. Finally, to schedule the last product of this family:

FTime(3,3,1,2) = *FTime(3,2,1,2)*+ch(3,2,3,3,1) + PTime(3,3,1,2)

= 53.82+3.52 + 32.58/1.08 = 87.51

FTime(3,3,1,3) = *FTime(4,3,1,3)*+ch(4,3,3,3,1) + PTime(3,3,1,3)

= 30.13+11.25 + 32.58/0.95 = 75.67

Hence, schedule product 3 of family 3 on machine 2.

5.2.5 Update $R = R \setminus \{3\} = \phi$. Since $R = \phi$, go to Step 6.

Step 6: Sequence products of the remaining families to machines at the first stage

The EFT rule is also applied to sequence the products of the families scheduled next on each machine.

Scheduling the products of family 1.

1. Find the first product of this family to be scheduled using the EFT rule.

   FTime(1,1,1,1) = *FTime(4,1,1,1)*+ ch(4,1,1,1,1) + PTime(1,1,1,1)

   = 64.32 + 10.61 + 47.50/1.10 = 118.11

   FTime(1,2,1,1) = *FTime(4,1,1,1)*+ ch(4,1,1,2,1) + PTime(1,2,1,1)

   = 64.32 + 9.05 + 18.19/1.10 = 89.91

   FTime(1,3,1,1) = *FTime(4,1,1,1)*+ ch(4,1,1,3,1) + PTime(1,3,1,1)

   = 64.32 + 8.09 + 26.55/1.10 = 96.55

   Hence, schedule product 2 of family 1 on machine 1.

2. Find the next product of the family to be scheduled.

   FTime(1,1,1,1) = *FTime(1,2,1,1)*+ ch(1,2,1,1,1) + PTime(1,1,1,1)

   = 89.91 + 4.19 + 47.50/1.10

   = 137.28

   FTime(1,3,1,1) = *FTime(1,2,1,1)*+ ch(1,2,1,3,1) + PTime(1,3,1,1)

   = 89.91 + 4.13 + 26.55/1.10

   = 118.18

   Hence, schedule product 3 of family 1 on machine 1. Product 1 of family 1 is then scheduled as the last product. The finish time of product 1 is determined as follows.

   FTime(1,1,1,1) = *FTime(1,3,1,1)*+ ch(1,3,1,1,1) + PTime(1,1,1,1)

   = 118.18 + 4.28 + 47.50/1.10 = 165.64

Scheduling the products of family 2

3. Find the first product of this family to be scheduled using the EFT rule.

FTime(2,1,1,2) = *FTime(3,2,1,2)*+ ch(3,2,2,1,1) + PTime(2,1,1,2)

= 53.82 + 11.42 + 34.72/1.08

= 97.39

FTime(2,2,1,2) = *FTime(3,2,1,2)*+ ch(3,2,2,2,1) + PTime(2,2,1,2)

= 53.82 + 11.77 + 31.58/1.08

= 94.83

FTime(2,3,1,2) = *FTime(3,2,1,2)*+ ch(3,2,2,3,1) + PTime(2,3,1,2)

= 53.82 + 11.02 + 33.43/1.08

= 95.79

Hence, schedule product 2 of family 2 on machine 2.

4. Find the next product of family 2 to be scheduled.

FTime(2,1,1,2) = *FTime(2,2,1,2)*+ ch(2,2,2,1,1) + PTime(2,1,1,2)

= 94.83 + 3.24 + 34.72/1.08

= 130.22

FTime(2,3,1,2) = *FTime(2,2,1,2)*+ ch(2,2,2,3,1) + PTime(2,3,1,2)

= 94.83 + 4 + 33.43/1.08

= 129.78

Hence, schedule product 3 of family 2 on machine 2. Product 2 of family 1 is then scheduled as the last product. The finish time of product 1 of family 2 is calculated as follows.

FTime(2,1,1,2) = *FTime(2,3,1,2)*+ ch(2,3,2,1,1) + PTime(2,1,1,2)

= 129.78 + 3.91 + 34.72/1.08 = 165.84

The sequences of products on the machines at the first stage obtained so far are presented in Figure 5.4.

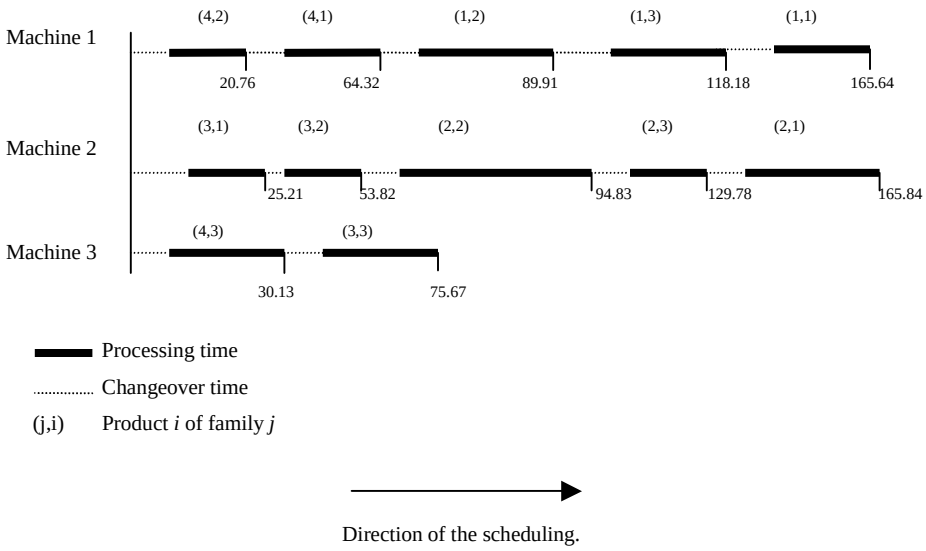

Figure 5.4: Sequences of Products on the Machines at Stage 1

Part 3: Balancing the Production Times of Machines at the First Stage

Step 7: Balance the production times of machines at the first stage

7.1 The machine with the largest finish time is 2.

7.2 Remove product 1 of family 2 from machine 2 and move it to other machines (i.e., machines 1 and 3).

7.3 Calculate the latest completion time on machines 1 and 3 after scheduling product 1 of family 2.

FTime(2,1,1,1) = *FTime(1,1,1,1)*+ ch(1,1,2,1,1) + PTime(2,1,1,1)

= 165.64 + 7.8 + 34.72/1.10 = 205.00

FTime(2,1,1,3) = *FTime(3,3,1,3)*+ ch(3,3,2,1,1) + PTime(2,1,1,3)

= 75.67 + 11.21 + 34.72/0.95 = 123.43

The machine yielding the shortest finish time of this product is machine 3.

Update the finish time of all machines at the first stage.

Machine 1: latest completion time = 165.64 time units

Machine 2: latest completion time = 129.78 time units

Machine 3: latest completion time = 123.43 time units.

7.4 The new latest finish time of the first-stage machines is equal to 165.64 time units for machine 1. Go back to Step 7.1. Using the same procedure, it was found that product 1 of family 1 could not be moved since it results in a higher latest completion time. Go to Step 7.5.

7.5 Remove product 3 of family 1 from machine 1. The calculations of the latest completion times on machines 2 and 3 after scheduling product 3 of family 1 are as follows:

$FTime(1,3,1,2) = FTime(2,3,1,2) + ch(2,3,1,3,1) + PTime(1,3,1,2)$

$= 129.78 + 11.49 + 26.55/1.08 = 165.85$

$FTime(1,3,1,3) = FTime(2,1,1,3,) + ch(2,1,1,3,1) + PTime(2,1,1,3)$

$= 123.43 + 6.26 + 26.55/0.95 = 157.64$

The machine yielding the earliest finish time of this product is machine 3. Hence product 3 of family 1 is rescheduled on machine 3.

Update the finish time of all machines at the first stage.

Machine 1: latest finish time = $89.91 + ch(1,2,1,1,1) + PTime(1,1,1,1)$

$= 89.91 + 4.19 + 47.68/1.1$

$= 137.44$ time units

Machine 2: latest finish time = 129.78 time units

Machine 3: latest finish time = 157.64 time units

This process is continued with all the products scheduled on machine 3, but none can be allocated to other machines. The final sequence of the products on the machines at the first stage is presented in Figure 5.5.

Part 4: Scheduling All Products on All other Stages

Step 8: Sequence all products on machines at stage *s : s* > 1, and calculate the makespan.

The sequences of products on machines at the first stage were obtained at the last step as shown in Figure 5.5.

8.1 Set s = 2.

8.2 Set H = {(4,2), (3,1), (4,3), (3,2), (4,1), (3,3), (1,2), (2,2), (2,1), (2,3), (1,1),(1,3)}

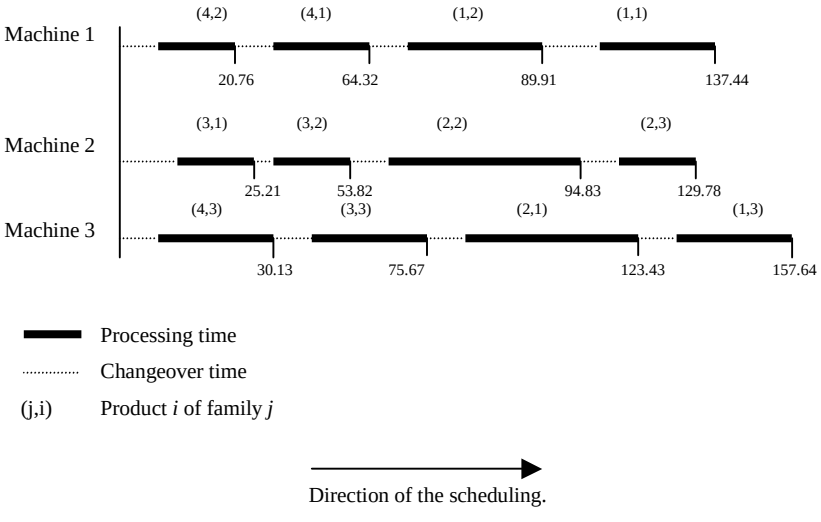

Figure 5.5: Final Sequences of Products on the Machines at Stage 1

8.3 Scheduling of the first product on one of the machines of the second stage using the LA rule.

Schedule steps:

LA-1 The first unscheduled product in set H is product 2 of family 4.

LA-2 Since no machine is processing the products of family 4, go to LA-12 to schedule this product to the machine yielding the lowest finish time, as detailed below.

LA-12 FTime(4,2,2,1) = STime(4,2,2,1) + PTime(4,2,2,1)

= 20.76 + 33.76/1.00 = 54.52 time units

FTime(4,2,2,2) = STime(4,2,2,2) + PTime(4,2,2,2)

= 20.76 + 33.76/0.93 = 57.06 time units

Schedule this product to machine 1.

8.4 Update H = H \ {(4,2} = {(3,1), (4,3), (3,2), (4,1), (3,3), (1,2), (2,2), (2,1), (2,3), (1,1), (1,3)}. Then go back to Step 8.3.

8.3 Scheduling of the first product in set H on one of the machines of stage 2 using the LA rule:

LA-1 The first unscheduled product in set H is product 1 of family 3.

LA-2 Since no machine is processing the products of family 3, go to LA-12, as follows.

LA-12 Schedule this product to the machine yielding the lowest finish time.

FTime(3,1,2,1)= STime(3,1,2,1) + PTime(3,1,2,1)

=FTime(4,2,2,1)+ch(4,2,3,1,1)+PTime(3,1,2,1)

= 54.52 + 7.36 + 14.99/1.00 = 76.87 time units

FTime(3,1,2,2) = STime(3,1,2,2) + PTime(3,1,2,2)

= 25.21 + 14.99/0.93 = 41.33 time units

LA-13 Schedule product 3, family 1 to machine 2.

8.4 Update H = H \ {(3,1} = {(4,3), (3,2), (4,1), (3,3), (1,2), (2,2), (2,1), (2,3), (1,1), (1,3)}. Then go back to Step 8.3.

8.3 Scheduling of the first product in set H on one of the machines of stage 2 using the LA rule:

LA-1 The first unscheduled product in set H is product 3 of family 4.

LA-2 The machine processing the products of this family is machine 1. Hence, MU(4) = {1}.

LA-3 Determine machine *m, m* $\in$ MU(4), which yields the earliest finish time.

FTime(4,3,2,1) = STime(4,3,2,1) + PTime(4,3,2,1)

= max{FTime(4,2,2,1)+ch(4,2,4,3,1),

FTime(4,3,1,3)} + PTime(3,1,2,1)

= max{54.52 + 3.92, 30.13} + 33.46/1.00

= 91.90 time units

Hence, *m* = 1.

LA-4 Determine the machine *m', m'* $\in$ M(2), which yields the earliest finish time.

FTime(4,3,2,1) = 91.90 time units (as determined in the last step)

FTime(4,3,2,2) = STime(4,3,2,2) + PTime(4,3,2,2)

= max{FTime(3,1,2,2) + ch(3,1,4,3,1),

FT(4,3,1,3)} + PTime(4,3,1,3)

= max{41.33 + 9.2, 30.13} + 33.46/0.93

= 86.51 time units

Hence, m' = 2.

LA-5 Since $m \neq m'$, go to LA-6 to check whether there is any incoming product of family 3 in the previous stage.

LA-6 Product 2 of family 3 (i.e., product 2) is scheduled to finish at time 53.82 in stage 1, so go to LA-7.

LA-7 Calculate FTime(4,3,2,2) + ch(4,3,3,2,2) = 86.51 + 9.61 = 96.12.

LA-8 Calculate STime(3,2,2,2) = max {FTime(3,2,1,2), FTime(3,1,2,2)+ch(3,1,3,2,2)}

= max {53.82, 41.33+3.85}

= 53.82

LA-9 Since, STime(3,2,2,2) < FTime(4,3,2,2) + ch(3,1,4,3,2), go to LA-10.

LA-10 Check whether the amount of reduced finish time of product 3 of family 4 (RFT(4,3)) is greater than DST(3,2).

RFT(4,3) = FTime(4,3,2,2) - FTime(4,3,2,1)

= 91.90 – 86.51

= 5.39 time units

DST(3,2) = FT(4,3,2,2) + ch(4,3,3,2,2) – max {FTime(3,1,2,2) + ch(3,1,3,2,2), FTime(3,2,1,2)}

= 86.51+9.61 – max{53.82, 41.33 + 3.85}

= 42.30 time units.

Since the value of RFT(4,3) is less than that of DST(3,2), go to LA-11.

LA-11 Do not schedule product 3 of family 4 on machine 2. Go back to LA-1 and apply the EFT rule to schedule this product on other machine(s). From the previous calculations in LA-4, it was found that this product can be scheduled on machine 1.

8.4 Update H = H \ {(4,3} = {(3,2), (4,1), (3,3), (1,2), (2,2), (2,1), (2,3), (1,1), (1,3)}. Then go back to Step 8.3.

8.3 Scheduling of the first product in set H on one of the machines of stage 2 using the LA rule:

LA-1 The first unscheduled product in set H is product 2 of family 3.

LA-2 The machine processing the products of family 3 is machine 2. Hence, MU(3) = {2}.

LA-3 Determine the machine *m, m* $\in$ MU(3), which yields the earliest finish time.

FTime(3,2,2,2) = STime(3,2,2,2) + PTime(3,2,2,2)

= max{FTime(3,1,2,2) + ch(3,1,3,2,2), FTime(3,2,1,2)} + PTime(3,1,2,1)

= max{41.33 + 3.85, 53.82} + 43.76/0.93

= 100.87 time units

Hence, *m* = 2.

LA-4 Determine the machine *m', m'* $\in$ M(2), which yields the earliest finish time.

FTime(3,2,2,2) = 100.87 time units (as determined in the last step)

FTime(3,2,2,1) = STime(3,2,2,1) + PTime(3,2,2,1)

= max{FTime(4,3,2,1)+ch(4,3,3,2,2),

FTime(3,2,2,1)} + PTime(4,3,1,3)

= max{91.9 + 9.61, 53.82} + 43.76/1.0

= 145.27 time units

Hence, *m*' = 2.

LA-5 Since *m'* = *m* = 2, go to LA-13.

LA-13 Schedule product 2 of family 3 on machine 2.

8.4 Update H = H \ {(3,2} = {(4,1), (3,3), (1,2), (2,2), (2,1), (2,3), (1,1), (1,3)}. Then go back to Step 8.3.

8.3 Scheduling of the first product in set H on one of the machines of stage 2 using the LA rule:

LA-1 The first unscheduled product in set H is product 1 of family 4.

LA-2 The machine processing the products of family 4 is machine 1.

Hence, MU(1) = {1}.

LA-3 Determine machine *m, m* ∈ MU(1), which yields the earliest finish time.

FTime(4,1,2,1) = STime(4,1,2,1) + PTime(4,1,2,1)

= max{91.90 + 3.64, 64.32} + 36.55/1.00

= 132.09 time units

Hence, *m* = 1.

LA-4 Determine the machine *m, m* ∈ M(2), yielding the earliest finish time.

FTime(4,1,2,1) = 132.09 time units (as determined in the last step)

Hence, m = 1.

FTime(4,1,2,2) = STime(4,1,2,2) + PTime(4,1,2,2)

= max{100.87 + 7.47, 64.32} + 36.55/0.93

= 147.64 time units

Hence, m' = 1.

LA-5 Since m' = m = 1, go to LA-13.

LA-13 Schedule product 1 of family 4 on machine 1.

8.4 Update H = H \ {(4,1} = {(3,3), (1,2), (2,2), (2,1), (2,3), (1,1), (1,3)}. Then go back to Step 8.3 to schedule the first product in set H on one of the machines of stage 2. The process is continued until all products in set H are scheduled on the machines in this stage. Figure 5.6 shows the product sequences obtained on the machines of stage 2.

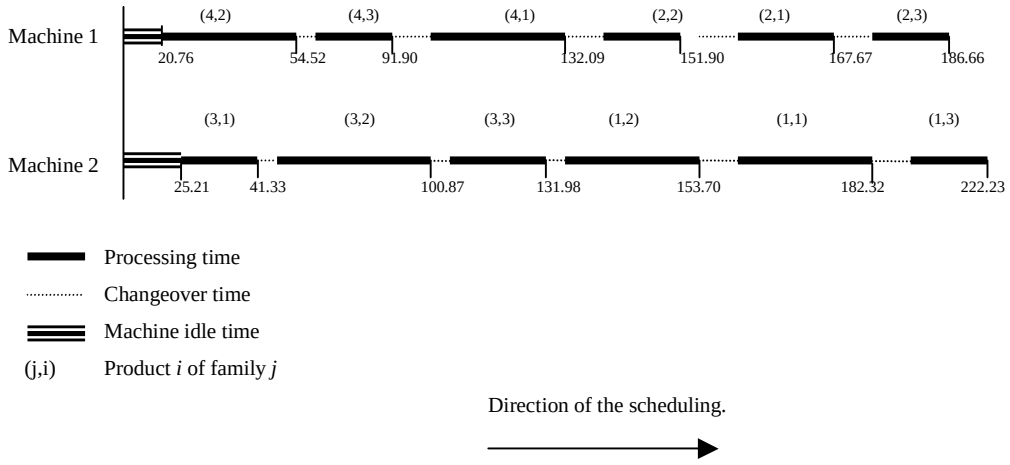

Figure 5.6: Product Sequences on Machines at Stage 2

8.5 Update s = s+1 = 3. Since s ≤ S, go to Step 8.2. The procedure is repeated to schedule all products on the machines of the third stage. Figure 5.7 shows the results of the products sequence obtained for this stage.

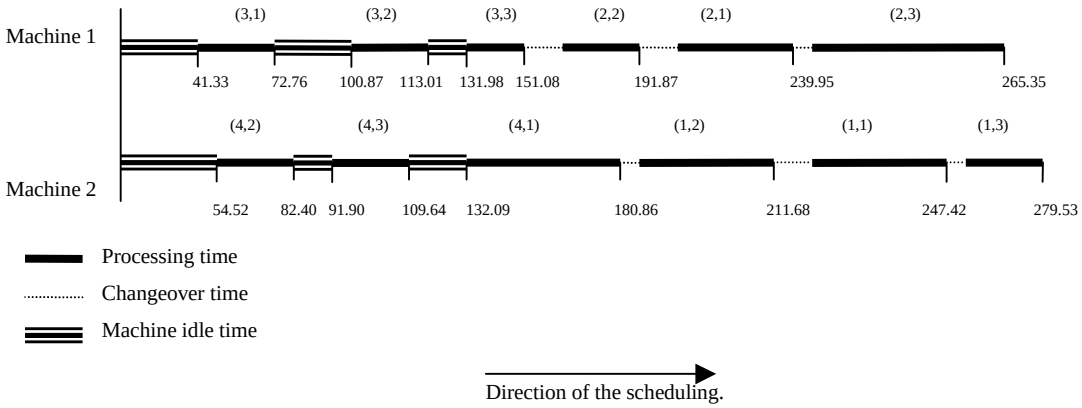

Figure 5.7: Sequences of Products on Machines at the Last Stage

From Figure 5.7, the makespan of this solution is 279.53 time units. The product sequences on each machine of each stage are presented as follows.

Stage 1: Machine 1: (4,2) -> (4,1) -> (1,2) ->(1,1)

Machine 2: (3,1) ->(3,2)->(2,2)->(2,3)

Machine 3: (4,3)->(3,3)->(2,1)->(1,3)

Stage 2: Machine 1: (4,2) -> (4,3) ->(4,1)->(2,2)->(2,1)->(2,3)

Machine 2: (3,1) ->(3,2)->(3,3)->(1,2)->(1,1)->(1,3)

Stage 3: Machine 1: (3,1)->(3,2) -> (3,3) ->(2,2)->(2,1)->(2,3)

Machine 2: (4,2) -> (4,3)->(4,1)->(1,2)->(1,1)->(1,3)

5.3 Phase 2: Improving the Initial Solution Using the TSH Algorithm

The initial solution obtained from Phase 1 (using the FFSDSTH algorithm) may not be close to the optimal solution. A different heuristic is required to generate better schedules. The final solution of the first phase can be considered as an initial solution that will be improved in this phase. From the flow process presented in Figure 5.1, the heuristic of the second phase has three main steps: 1) moving families between (or within) machines at the first stage, 2) moving products between (or within) machines at the first stage, and 3) finding the best sequence resulting in the minimum makespan. Prior to the presentation of the TSH algorithm, the background of the TS as implemented in this problem is introduced in the following five sections. The implementation of the TS heuristic with the $FFs(Q_{m1},Q_{m2},\ldots,Q_{ms})/S_{ipm}/C_{max}$ problem is introduced in Section 5.3.1. The tabu list is discussed in Section 5.3.2 and is followed by a discussion of the neighborhood size in Section 5.3.3, the tabu restriction in Section 5.3.4, and the admissible moves in Section 5.3.5.

5.3.1 Implementing the TS Heuristic with the $FFs(Q_{m1},Q_{m2},\ldots,Q_{ms})/S_{ipm}/C_{max}$ Problem

In the tabu search, a decision is made from the set of admissible candidates. The candidate decisions are evaluated and the best one is selected. A candidate is admissible either if it is not tabu or if its tabu status can be overridden by the aspiration criterion. As suggested by Laguna et al. (1993) and Barnes & Laguna (1993), there are four key elements to be considered in the TS:

- To identify the attributes (i.e., the criteria used to define or characterize a move) of a move that will be used to generate the tabu classification. Attributes of moves, e.g., indices of jobs (or jobs

numbers), positions of jobs, and weights of jobs, are identified and recorded in the tabu list in order to prevent move reversals.

- To identify the actual tabu restriction based on the attributes.
- To identify a good data structure to keep track of moves that have a tabu status, and to free those moves from their tabu condition when their short-term memory has expired.
- To identify an aspiration condition in an effort to allow the tabu status of a move to be overridden if it yields a better solution.

Two popular types of moves found in the literature for the flowshop problem are: (1) exchanging jobs (i.e., swap move) and (2) removing the job placed at the x^{th} position and then putting it at the y^{th} position (i.e., insertion move). Taillard's (1990) experiments showed that the insertion move is the most efficient in terms of quality and computation time. Hence, only the insertion move will be considered in this research.

Insertion moves allow a single job to move from one machine to another. Let *P* be the set of all jobs, $P = \{1,2,\ldots,np\}$ and $np_{s,m}$ denote the number of jobs scheduled on machine *m* of stage *s, m* $\in$ *M(s) and s* $\in$ ψ. At each stage *s*, the jobs in set *P* are partitioned into *m(s)* groups. This means that there are *m(s) job processing orders* (or schedules) at stage *s*. The processing order of jobs on machine *m* of stage *s* can be expressed by a permutation $\pi_{s,m}$:

$$\pi_{s,m} = (\pi_{s,m}(1), \pi_{s,m}(2), \pi_{s,m}(3), \ldots, \pi_{s,m}(np_{s,m}))$$

where $\pi_{s,m}(k)$ denotes the job of set *P* which is in position *k* in $\pi_{s,m}$. Hence, the processing order of jobs at stage *s* can be completely presented by the set of

m(s) permutations $\pi_s = \{\pi_{s,1}, \pi_{s,2}, \ldots, \pi_{s,m(s)}\}$. The collection of the job processing orders (i.e., schedules) is defined by *s*-tuple $\pi = (\pi_1, \pi_2, \ldots, \pi_S)$.

Let *s* denote a stage, m_1 and m_2 two machines in this stage, and *x, y* two positions of jobs on machine m_1 and m_2, respectively. For a processing order π, the move (s,m_1,x,m_2,y) is defined as the insertion move in which the job at position *x* is removed from machine m_1 and placed on machine m_2 at position *y*. If the insertion-type move is performed between two machines ($m_1 \neq m_2$) in stage *s,* the deletion of job *i* from position *x* in permutation $\pi_{s,m1}$ and its insertion in position *y* in permutation $\pi_{s,m2}$ implies the following events:

1. jobs $\pi_{s,m1}(x+1), \ldots, \pi_{s,m1}(np_{s,m1})$ are moved to the left by a single position in the new permutation $\pi'_{s,m1}$, and
2. job *i* is located at position y and jobs $\pi_{s,m2}(y), \pi_{s,m2}(y+1), \ldots, \pi_{s,m1}(np_{s,m2})$ are moved to the right by a single position in the new permutation $\pi'_{s,m2}$.

Conversely, if the insertion-type move is performed within the same machine ($m_1 = m_2$) in stage *s*, the deletion of job *i* from position *x* and its insertion in position *y* in permutation $\pi_{s,m1}$ implies the following events:

1. If x < y, jobs $\pi_{s,m1}(x+1), \ldots, \pi_{s,m1}(y)$ are moved to the left by a single position, and job *i* is located at position *y* in the new permutation $\pi'_{s,m1}$, or
2. If x > y, job *i* is located at position *y* and jobs $\pi_{s,m1}(y), \pi_{s,m1}(y+1), \ldots, \pi_{s,m1}(x-1)$ are moved to the right by a single position in the new permutation $\pi'_{s,m1}$.

5.3.2 Tabu List

The tabu list stores attributes of the performed moves. These moves are defined by a pair (or two pairs) of adjacent jobs in a production stage, as detailed below. The selection of the pair(s) depends on the insertion move performed. In this research, the tabu status corresponding to the insertion move is defined as a triple element (*s, i, p*) representing the pair of jobs *i* and *p* from stage *s*. This representation was also used in the study of Nowicki and Smutnicki (1998). Let $T = (T_1, T_2, \ldots, T_{maxtl})$ be a tabu list of a fixed length maxtl, where $T_{tl} = (s, i, p)$ is a triple element and tl = 1, 2, …, maxtl. The tabu list is initially empty. Every time an insertion move is performed in a processing order π, this move is added to the tabu list.

Details of the definition of the stored attributes of a move performed in a processing order π are presented below. Figure 5.8 shows an illustration of the moves. In this figure, thick arcs link the pair of jobs at stage *s* that will be added to the tabu list after the move is performed.

1. Moves are performed within a machine (i.e., $m_1 = m_2 = m$)

   In this case, only one triple element is added to the tabu list. Two cases are considered here:

   Case 1.1: $x < y$

   The triple element added to the tabu list is composed of the stage number, the index of the moved job, and the index of the job to the right of the moved job (prior to the move). This triple element is represented as $(s, \pi_{s,m}(x), \pi_{s,m}(x+1))$.

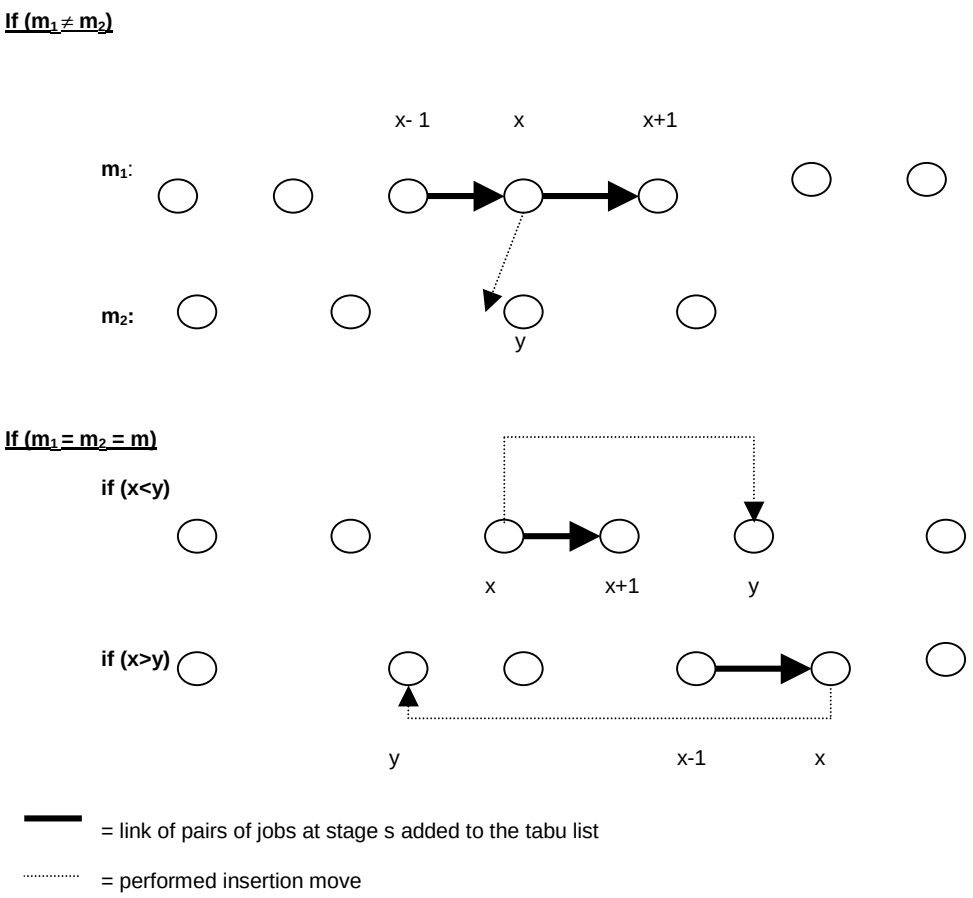

Figure 5.8: Tabu List of a Move (s,m_1,x,m_2,y)

Case 1.2: x > y

The triple element added to the tabu list consists of the stage number, the index of job to the left of the moved job (prior to the move), and the index of the moved job. The triple element is represented as (s, $\pi_{s,m}(x-1)$, $\pi_{s,m}(x)$).

2. Moves are performed between two different machines ($m_1 \neq m_2$). In this case, one or two triple elements may be added to

the tabu list depending on the move that has been performed, as detailed below.

Case 2.1: The Job to be moved is not the first or the last job in $\pi_{s,m1}$ (i.e., $1 < x < np_{s,m1}$).

Two triple elements are added to the tabu list. These are: (1) the triple element that comprises the stage number, the index of the moved job, and the index of job to the right of the moved job (prior to the move) (i.e., (s, $\pi_{s,m1}(x)$, $\pi_{s,m1}(x+1)$)) and (2) the triple element that consists of the stage number, the index of the job to the left of the moved job (prior to the move), and the index of the moved job (i.e., (s, $\pi_{s,m1}(x-1)$, $\pi_{s,m1}(x)$)).

Case 2.2 The job to be moved is the first job in $\pi_{s,m1}$ (i.e., $x = 1$).

Only one triple element is added to the tabu list which consists of the stage number, the index of the moved job, and the index of job to the right of the moved job (prior to the move) (i.e., (s, $\pi_{s,m1}(x)$, $\pi_{s,m1}(x+1)$)).

Case 2.3: The job to be moved is the last job in $\pi_{s,m1}$ (i.e., $x = np_{s,m1}$).

The triple element added to the tabu list is composed of the stage number, the index of job to the left of the moved job (prior to the move), and the index of the moved job (i.e., (s, $\pi_{s,m1}(x-1)$, $\pi_{s,m1}(x)$)).

The attributes of the performed moves in a tabu list are applied along with the neighborhood size and the tabu restriction, as explained in the subsequent sections (Sections 5.3.3 and 5.3.4., respectively) to prevent move reversals in the future moves.

5.3.3 Neighborhood Size

The neighborhood generation is one of the important elements of TS. The neighborhood generation usually has a very significant effect on the efficiency of the search. In the case of $FFs(Q_{m1},Q_{m2},\ldots,Q_{ms})/S_{ipm}/C_{max}$ sequencing problems, for instance, when an insertion move is performed within the same machine (e.g., machine *m* in stage *s*), the size of the neighborhood (i.e., number of possible moves) can be shown to be equal to $(np_{s,m} - 1)^2$. If too few neighborhoods are produced, some good solutions may be overlooked. Conversely, if all neighborhood solutions are produced, the search may produce better solutions but will be time consuming. The evaluation of the entire neighborhood for large size problems may not be practical. A procedure to curtail the length of the search (i.e., by reducing the size of the neighborhood) is determined based on the use of the *move distance.*

Consider the case of problems where an insertion move is performed within the same machine. Instead of examining all possible moves of job $\pi_{s,m}(x)$ to be inserted in position *y*, the search is restricted to those positions within a certain distance *d* from the job's position. More precisely, job $\pi_{s,m}(x)$ can be moved (i.e., inserted in position *y*) if the difference between *y* and *x* is less than *d* (i.e., $|y - x| < d$), where *d* is the maximum moving distance allowed and may be determined after experiencing with different problem settings.

In general, defining a good size of *d* depends on the structure of the problem. Based on studies by Laguna et al. (1993) and Barnes and Laguna (1993), the value of *d* can be obtained as follows:

- For $np_{s,m} \leq 30$

  $d = \lfloor np_{s,m}/2 \rfloor - 1$

  where $\lfloor h \rfloor$ = the largest integer less than or equal to h

- For $np_{s,m} > 30$

  $d = (\lfloor np_{s,m}/2 \rfloor / 2) \times c/4$

  where *c* is determined experimentally (Laguna et al., 1993 and Brandao & Mercer, 1997). The value of *c* is usually a number between 1 and 4 (Laguna et al., (1993)).

The move distance concept was used in many studies such as in those of Laguna et al. (1993), Barnes and Laguna (1993), Amin-Naseri (1993), Brandao and Mercer (1997), and Nowicki and Smutnicki (1998).

5.3.4 Tabu Restriction

In order to prevent a move reversal, a tabu restriction is used to determine if the future move is admissible. There are many ways to generate the tabu restriction. One effective way is to apply a move distance. Consider the case when the job is moved within the same machine. After a job $\pi_{s,m}(x)$ is removed from position *x* and inserted in position *y* on the same machine *m* of stage *s* where $y > x$, job $\pi_{s,m}(x)$ cannot be placed in the future (as long as this move is in the tabu list) any earlier than position *y.* This means that the job that was initially at position *x* cannot move to the left in the subsequent schedules until the attributes of this job are removed from the tabu list (Laguna et al., 1993).

In this research, the move distance is also used to generate the tabu restriction. The move is considered to be admissible if no triple element resulting

from performing a move *(s, m_1, x, m_2, y)* exists in the tabu list. The tabu restrictions of a move *(s,m_1,x,m_2,y)* of each case are explained as follows.

1. Jobs are moved within a machine ($m_1 = m_2 = m$)

   There are two cases considered when jobs are moved within a machine, as detailed below. Also, Figure 5.9 shows the tabu restriction of the move *(s, m, x, m, y).*

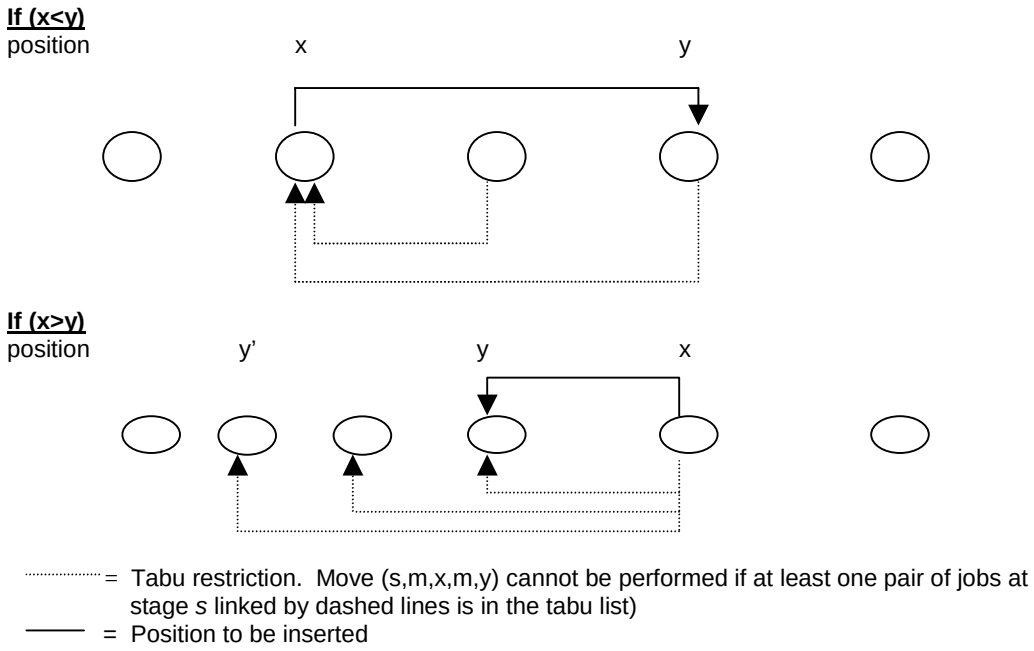

Figure 5.9: Tabu Restriction when Jobs are Moved within a Machine

Case 1.1: x < y, where *y - x < d*

The tabu restrictions consist of all the triple elements resulting from performing the move *(s,m,x,m,y),* which comprise the stage number, the index of job at position *k* (prior to the move) where $x<k\leq y$,

and the index of the moved job. These triple elements are represented as (s, $\pi_{s,m}(k)$, $\pi_{s,m}(x)$).

Case 1.2: $x > y$, where $x - y < d$

Let y' be the end position of the move distance. This means that $x - y' = (d - 1)$. The tabu restrictions consist of all the triple elements resulting from performing the move *(s, m, x, m, y)*, which comprise the stage number, the index of the moved job, and the index of job at position k (prior to the move) where $y' \leq k < x$ (i.e., (s, $\pi_{s,m}(x)$, $\pi_{s,m}(k)$)).

2. Jobs are moved between machines ($m_1 \neq m_2$)

When jobs are moved between two machines, the move distance starts from position (y-y') and ends at position (y + y") on machine m_2 (i.e., (y" + y) – (y' + y) = (d - *1)*). Figure 5.10 shows the tabu restriction when insertion is performed in different machines. Details of the triple element generation for each case are presented as follows.

Case 2.1: $y = 1$

The tabu restrictions consist of all the triple elements resulting from performing the move *(s, m, x, m, 1)*, which comprise the stage number, the job index at position x, and the job index at position $y + z$ (prior to the insertion of job $\pi_{s,m1}(x)$), where $0 \leq z < d$ (i.e., (s, $\pi_{s,m1}(x)$, $\pi_{s,m2}(1 + z)$)).

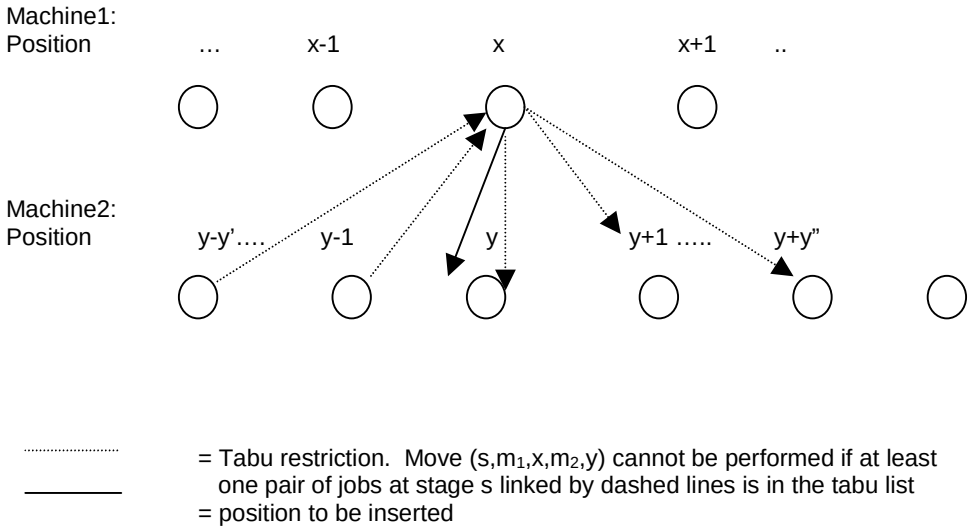

Figure 5.10: Tabu Restriction when Jobs are Moved between Machines

Case 2.2: 1 < y ≤ ⌈d/2⌉, where ⌈u⌉ is the least integer greater than or equal to u.

The tabu restrictions consist of all the triple elements resulting from performing the move *(s, m_1, x, m_2, y),* which for this case are:

(1) the triple elements that consist of the stage number, the job index at position *w* (prior to the insertion of job $\pi_{s,m1}(x)$), where $1 \le w < y$, and the job index at position x (i.e., (s, $\pi_{s,m2}(w)$, $\pi_{s,m1}(x)$)), and

(2) the triple elements that consist of the stage number, the job index at position x, and the job index at position $y + z$ (prior to the insertion of job $\pi_{s,m1}(x)$), where $0 \le z \le (d - y)$ (i.e., (s,$\pi_{s,m1}(x)$, $\pi_{s,m2}(y+z)$)).

Case 2.3: $\lceil d/2 \rceil < y < \lceil np_{s,m2} - d/2 \rceil$

The tabu restrictions consist of all the triple elements resulting from performing the move *(s, m_1, x, m_2, y)*, which for this case are:

(1) the triple elements consisting of the stage number, the job index at position *y - w* (prior to the insertion of job $\pi_{s,m1}(x)$) where $1 \le w < \lceil d/2 \rceil$, and the job index at position *x* (i.e., (s, $\pi_{s,m2}(y - w)$, $\pi_{s,m1}(x)$)), and

(2) the triple elements comprising the stage number, the job index at position x, and the job index at position *y + z* (prior to the insertion of job $\pi_{s,m1}(x)$) where $0 \le z \le \lceil d/2 \rceil$ (i.e., (s, $\pi_{s,m1}$(x), $\pi_{s,m2}(y+z)$)).

Case 2.4: $\lceil np_{s,m2} - d/2 \rceil \le y \le np_{s,m2}$

The tabu restrictions consist of all the triple elements resulting from performing the move *(s,m_1,x,m_2,y)*, which for this case are:

(1) the triple elements consisting of the stage number, the job index at position x, and the job index at position *y + w* (prior to the insertion of job $\pi_{s,m1}(x)$) where $y + w \le np_{s,m2.}$. These triple elements are represented as (s, $\pi_{s,m1}$(x), $\pi_{s,m2}(y+w)$).

(2) the triple elements comprising the stage number, the job index at position *y - z* (prior to the insertion of job $\pi_{s,m1}(x)$), where $1 \le z \le (d-1) - (np_{s,m2} - y)$, and the job index at position *x*. These triple elements are represented as (s, $\pi_{s,m2}(y-z)$, $\pi_{s,m1}(x)$).

Case 2.5: $y = np_{s,m2} + 1$

The tabu restrictions consist of all the triple elements resulting from performing the move *(s,m_1,x,m_2,y),* which for this case comprise the stage number, the job index at position *y - z* (prior to the insertion

of job $\pi_{s,m1}(x)$) where 0 < z < d, and the job index at position *x*. These triple elements are represented as $(s,\pi_{s,m2}(y\text{-}z), \pi_{s,m1}(x))$.

5.3.5 Admissible Moves

The move to be performed at a given iteration may be found by examining the value of the objective function for all candidate moves and selecting the best one. As discussed in Sections 5.3.3 and 5.3.4, the move is considered to be admissible if the following two conditions are satisfied.

1. If the move is within the same machine, the difference between the initial position of the job to be moved and its new position is less than *d* (i.e., $|y - x| < d$), where *d* is the maximum moving distance allowed.
2. No triple element of a tabu restriction exists in the tabu list.

The following example shows how to determine whether a move is admissible.

Example: Consider moving a job between two machines (m_1 and m_2) in stage s. Assume that the tabu list T is initially empty. The value of *m(s)* is equal to 2, and the job processing orders on the two machines are presented below.

$\pi_{s,m1}$ = (3, 2, 1, 4,9,10,15,16,17,18,19,24,25), and

$\pi_{s,m2}$ = (5,7,6,8,11,12,13,14,20,21,22,23).

Consider the move (s,1,2,2,2). The value of *d* can be obtained using the formula presented in Section 5.3.3. Hence, d = (12/2)–1 = 5. The tabu restrictions resulting from the move (s,1,2,2,2) consist of the following triple elements: (s,5,2), (s,2,7), (s,2,6), (s,2,8), and (s,2,11).

Since none of these triple elements is in the tabu list, the move (s,1,2,2,2) is admissible. Performing this move yields the following new sequences:

$\pi'_{s,m1}$ = (3,1,4,9,10,15,16,17,18,19,24,25), and

$\pi'_{s,m2}$ = (5,2,7,6,8,11,12,13,14,20,21,22,23).

The triple elements added to the tabu list after performing the move (s,1,2,2,2) are: 1) (s,3,2), and 2) (s,2,1).

Consider the move (s,2,2,1,2). Using the formula presented in Section 5.3.3, the value of *d* is equal to 5. The tabu restrictions resulting from the move (s,2,2,1,2) consist of the following triple elements: (s,3,2), (s,2,1), (s,2,4), (s,2,9), and (s,2,10). The move (s,2,2,1,2) cannot be performed because the triple elements (s,3,2) and (s,2,1) are in the tabu list.

Details of the TSH heuristic are given below.

Part 5: Moving Families between Machines (and within a Machine) at the First Stage

In this part, the families scheduled on machines at the first stage are moved between machines (or within a machine) in an effort to minimize the makespan. This process is not performed for the other stages as it takes a large amount of computation time, and yields very little improvement. The best solution obtained from the previous Phase will be used as the initial solution. For each iteration, all the admissible moves within the neighborhood in the current schedule are evaluated and the best move is selected. The tabu list, neighborhood size, and tabu restrictions are applied in the process of moving

families between machines at the first stage. The details of these three components are described below, and are followed by the notation used in this part and the detailed procedure of the TSH algorithm.

Tabu List

Let *N* be the total number of families. The size of the tabu list is determined as follows:

1. *m(1)* =1.

   Based on the studies of Laguna et al. (1993), the size of the tabu list when jobs are moved within a machine is determined as described below.

   1.1 $N \leq 12$

   $|T| = \lfloor N / 2 \rfloor$

   where, $|T|$ = size of the tabu list

   1.2 $N > 12$

   $|T| = 7$

2. *m(1)* > 1

   2.1 If $2 \leq N \leq 10$, $1 \leq |T| \leq 3$.

   2.2 If $11 \leq N \leq 20$, $3 \leq |T| \leq 5$.

   2.3 If $21 \leq N \leq 50$, $5 \leq |T| \leq 10$.

   2.4 If $N > 51$, $10 \leq |T| \leq 15$.

Neighborhood Size and Tabu Restriction

1. For $m_1 = m_2 = m$

   Let $nf_{s,m}$ be the number of families schedule on machine *m* in stage *s*.

   The value of *d* is determined as follows:

   - If $nf_{s,m} = 2$, $d = 1$.
   - If $3 \leq nf_{s,m} \leq 5$, $d = 2$.

- If $6 \leq nf_{s,m} \leq 9$, d =3.
- If $nf_{s,m} > 9$, the value of *d* is calculated using the formula presented in Section 5.3.3. If $nf_{s,m} > 30$, the value of *c* is equal to 2.

2. For $m_1 \neq m_2$

- If $nf_{s,m2} = 1$, or 2, d = 1.
- If $nf_{s,m2} = 3$, d = 2.
- If $4 \leq nf_{s,m2} \leq 9$, d = 3.
- If $nf_{s,m2} \geq 10$, the value of *d* is determined using the formula presented in Section 5.3.3. If $nf_{s,m2} > 30$, the value of *c* is equal to 2.

Notation

iter_fam	=	current iteration number for the process of moving families between machines at the first stage
iter_max_fam	=	maximum number of iterations allowed to be performed in the family insertion move procedure
best_value_fam	=	the minimum makespan found so far
best_seq_fam	=	the best schedule found so far
tor_iter_fam	=	maximum number of iterations allowed between two successive improvements
best_iter_fam	=	iteration where the best solution was found so far
size_tabu_list_fam	=	size of tabu list
move_value_fam	=	the minimum makespan obtained from the evaluation of all admissible moves in the iteration

move_seq_fam = the schedule that yields the minimum makespan in the iteration

Figure 5.11 shows the flow of the TS search implementation when moving families between or within machines at the first stage. Details of this part are described below.

Step 9: Initialize all parameters used in the process of moving families between the machines at the first stage.

Set iter_fam = 0

best_value_fam = makespan obtained in Phase 1 (Part 4)

best_iter_fam = 0

iter_max_fam =100

tor_iter_fam = 30

size_tabu_list_fam = 3 for 12 families (50 products)

= 4 for 18 families (80 products).

The values of parameters iter_max_fam, tor_iter_fam and size_tabu_list_fam are a-priori fixed constants that were determined experimentally. In this research, only two data sets (sets of 50 and 80 products, as detailed in Chapter 7) were tested with the TSH algorithm. Computational experience showed that a value of 100 of the maximum number of iterations (iter_max_fam) is a good value in terms of computational time and solution quality. Likewise, a value of 30 for the maximum number of iterations without improving the best solution (tor_iter_fam) was found to be good. Also values of 3 and 4 are adequate for the size of the tabu list (size_tabu_list_fam) when the numbers of families are 12 and 18, respectively.

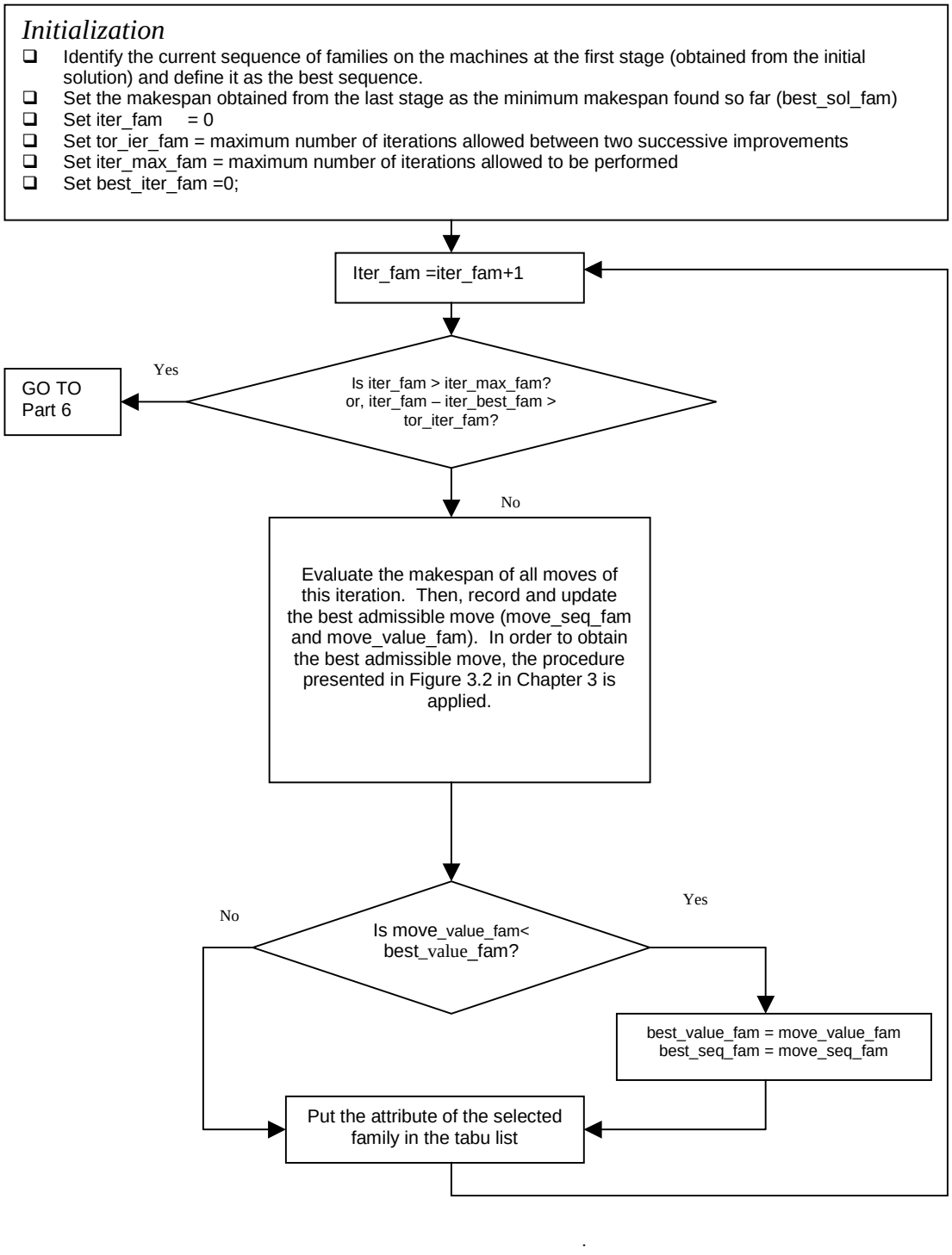

Figure 5.11: Flow Process of Moving Family between and within Machines at the First Stage

Step 10: Update the number of current iterations.

Increment the number of iterations (iter_fam) by 1.

Step 11: Check if the search should be stopped.

In this step, two stopping criteria are used:

11.1 Stop the search if the number of the current iterations (iter_fam) is greater than max_iter_fam, or

11.2 Stop the search if the number of successive iterations without improvement is greater than tor_iter_fam.

If the search is not stopped, go to Step 12; otherwise, go to Part 6 to proceed with the movement of products.

Step 12: Move families between (or within) machines.

Families that were divided between machines are treated as individual sub-families. Sequences of products within families (or sub-families) are not changed in this step.

12.1 For each admissible move, perform the following:

- determine the tentative schedule of families on machines in stage 1 after performing the move for the entire family (or sub-family).
- tentatively re-schedule all products on machines in stages 2 through *S* using the procedure detailed in Step 8 and find the corresponding makespan.

12.2 After all admissible moves have been performed, select the move that yields the minimum makespan. Denote the minimum makespan as move_value_fam and the corresponding schedule as move_seq_fam.

12.3 Check whether move_value_fam is less than the best_value_fam. If true, perform the following updates and go to Step 12.4

best_value_fam = move_value_fam,

best_seq_fam = move_seq_fam.

Otherwise, go to Step 12.4

12.4 Put the attribute of this move in the tabu list and go back to Step 10.

Part 6: Moving Products between (and within) Machines at the First Stage

In this part, the products are moved between (and within) machines in an effort to minimize the makespan. As in Part 5, the process of moving products between (and within) machines is performed only in the first stage. The best solution obtained in the previous part is used as the initial solution. The notation used in the implementation of the TS is described below and is followed by the procedure. Basically, the rules used to define the tabu list and to determine the tabu list size, neighborhood size, and tabu restriction are the same as in Part 5.

Notation

iter_prod = current iteration number for the process of moving products between machines at the first stage

iter_max_prod = maximum number of iterations allowed to perform in the process of products insertion procedure

best_value_prod = the minimum makespan found so far

best_seq_prod = the best schedule found so far

tor_iter_ prod = maximum number of iterations allowed between two successive improvements

best_iter_prod = iteration where the best solution has been found so far

size_tabu_list_ prod = size of tabu list

move_value_ prod = the minimum makespan obtained from the evaluation of all admissible moves in the iteration

move_ seq_prod = the schedule that yields the minimum makespan in the iteration

Details of this part are described as follows.

Step 13: Initialize all parameters used in the process of moving product between machines at the first stage.

Set iter_ prod = 0,
best_sol_ prod = makespan obtained in Part 5
best_iter_ prod = 0,
iter_max_ prod =100,
tor_iter_ prod = 30,
size_tabu_list_prod = 7 for 50 products
= 12 for 80 products.

The values of parameters iter_max_prod, tor_iter_prod and size_tabu_list_prod are a-priori fixed constants that were determined experimentally. Computational experience showed that a value of 100 for the maximum number of iterations (iter_max_prod) is a good value in terms of computational time and solution quality. Likewise, a value of 30 for the maximum number of iterations without improving the best

solution (tor_iter_prod) was found to be good. Also, values of 7 and 12 are adequate for the size of the tabu list (size_tabu_list_prod) when the numbers of products are 50 and 80, respectively.

Step 14: Update the number of current iteration.

Increment the number of (iter_prod) by 1.

Step 15: Check if the search should be stopped.

The two stopping criteria used in Step 10 are also used in this step, as detailed below.

1. Stop the search if the maximum number of current iterations (iter_prod) is greater than max_iter_prod, or
2. Stop the search if the number of successive iterations without improvement is greater than tor_iter_prod.

If the search is not stopped, go to Step 16. Otherwise, go to Step 17.

Step 16: Move products between (or within) machines.

16.1 For each admissible move, perform the following:

- determine the tentative schedule of products on machines in stage 1 after performing a product move.
- tentatively re-schedule all products on machines in stages 2 through *S* using the procedure detailed in Step 8 and find the corresponding makespan.

16.2 After all admissible moves have been performed, select the move that yields the minimum makespan. Denote the minimum makespan as move_value_prod and the corresponding schedule as move_seq_prod.

16.3 Check if move_value_prod is less than best_value_prod. If true, perform the following updates and go to Step 16.4

best_value_prod = move_value_prod,

best_seq_prod = move_seq_prod.

Otherwise, go to Step 16.4

16.4 Put the attribute of this move in the tabu list and go back to Step 14.

Step 17: Determine the best makespan at the last stage and the best sequence found so far.

Applying the TSH algorithm to the solution obtained for the illustrated problem in Section 5.2, the makespan was improved to 247.75 time units. The product sequences obtained on the machines of each stage are presented below.

Stage 1: Machine1: (2,2)-> (2,1) -> (4,3) -> (4,2) -> (1,1)

Machine 2: (3,1) -> (1,2) -> (2,3)

Machine 3: (3,3) -> (4,1) -> (1,3) -> (3,2)

Stage 2: Machine 1: (3,1)-> (3,3) -> (2,1) -> (2,3) -> (4,3) -> (1,3) -> (1,1)

Machine 2: (2,2)-> (1,2) -> (4,1) -> (4,2) -> (3,2)

Stage 3: Machine 1: (3,1)-> (1,2)-> (2,1) -> (4,3) -> (4,2) -> (1,1)

Machine 2: (2,2)-> (3,3) -> (2,3) -> (4,1) -> (1,3) -> (3,2)

where (j,i) means product i of family j.

CHAPTER 6

LOWER BOUNDS

6.1 Introduction

Normally, the quality of heuristic solutions is assessed by comparing their results to: (1) optimal solutions, (2) lower bounds, and/or (3) reference objective values obtained by the best known approximation algorithms. The flexible flowshop problem with sequence dependent setup is known to be NP-hard, and hence finding an optimal solution for average or large-size problems will be computationally intractable. Since the $FFs(Qm_1, Qm_2,\ldots, Qm_s)/s_{ipm}/C_{max}$ is also relatively new, and no approximation algorithms can be found for it in the literature, the only alternative left is to develop lower bounds for the problem and use them to assess the quality of the TS heuristic solutions.

Lower bounds can be obtained using a combinatorial approach as detailed below. Other lower bounds can be obtained by relaxing the integrality constraints in the integer programming formulation. Using the latter approach, several problems with relaxed formulations were solved using the MPL/CPLEX software, but the results obtained were not good enough, as the lower bounds obtained were less than fifty percent of those obtained with the combinatorial approach. Hence, the relaxed linear programming formulation was not considered any further.

6.2 Lower Bound Determination

Problem parameters and notation used in the development of the lower bound are defined below. The notation used in Chapters 4 and 5 is kept as much as possible and supplemented with some additional variables.

Notation

i, p = product indices

j, q = family indices

N = number of families

J = set of all families

= {1,2,..,N)

(j, i) = product i of family j

F_j = set of products in family j; $j \in J$

= {1, 2,…,f_j}

f_j = number of products in family j

ψ = set of stages in a production line

= {1,2,..,S}

s = stage index

np = total number of products

NP = set of products from all families

$= \bigcup_{j=1}^{N} F_j$; | NP | = np

$m(s)$ = number of machines in stage s

$M(s)$ = set of machines at stage s

= {1,2,…, $m(s)$}

$v_{s,m}$ = speed of machine m at stage s

$\lceil x \rceil$ = the least integer value greater than or equal to x.

$SI(i)$ = the setup time from idling for product i in stage 1

$P(i,s)$ = the processing time of product i on the fastest machine in stage s

$T(i,s)$ = processing time of product i on a standard machine (i.e., speed = 1) in stage s

$CT(i)$ = the cumulative processing time of product i on the fastest machines from stage 1 through stage $S-1$

$$= \sum_{s=1}^{S-1} P(i,s)$$

MN(i,s) = the minimum minor setup time of product *i* at stage *s.* *MN(i,s)* is the lowest setup time for product *i* at stage *s* from any other product that belongs to the same family. Let $i \in F_j$, the value of *MN(i,s)* is obtained as follows.

$$MN(i,s) = \min_{\substack{i \neq p, \\ p \in F_j}} \text{ch(j,p,j,i,s)}$$

MJ(i,s) = the minimum major setup time of product *i* at stage *s.* MJ(i,s) is the lowest setup time for product *i* at stage *s* from any product that belongs to a different family. Let $i \in F_j$, then:

$$MJ(i,s) = \min_{\substack{q \neq j, \\ p \in F_q}} \text{ch(q,p,j,i,s)}$$

ICT(i) = the sum of the setup time from idling at the first stage and the cumulative processing times of product *i* on the fastest machines from stage 1 through stage *S-1.*

$= SI(i) + CT(i)$

λ = the minimum value between *m(S)* and *m(1)*

$= \min \{m(S), m(1)\}$

xtra(s) = the difference between the number of machines in the last stage and that in stage *s*. If negative, a value of zero is used.

$= \max \{0, m(S) - m(s)\}$

E = set of λ products with lowest values of CT(*i*)

A = set of λ products with lowest values of ICT(*i*)

B = set of *np – N* products yielding the lowest values of *MN(i,S)*

C = set of *N – m(S)* products yielding the lowest values of *MJ(i,S)*

G = set of *m(1)* products yielding the lowest values of SI(*i*)

K = NP - A

Z = $B \cap C$

D = $NP - (B \cup C)$

LB^F = the lower bound on the makespan obtained by the forward method

LB^B = the lower bound on the makespan obtained by the backward method

BLB = the best lower bound

= max $\{LB^F, LB^B\}$

Based on the flow or routing of products, two methods were developed in this research to calculate a lower bound on the makespan: 1) the forward method and 2) the backward method. The best lower bound (BLB) is obtained by taking the maximum value of the LB^F and LB^B.

To calculate the lower bound on the makespan for the $FFS(Qm_1,Qm_2,\ldots,Qm_s)/s_{ipm}/C_{max}$ sequencing problem, the key idea is to consider a flexible flowshop structure with all machines in each stage as fast as the fastest machine. The makespan can be determined by considering the sum of two quantities: (1) the last-stage machine total waiting and idle times and (2) the total setup and production times on the last-stage machines. These two quantities can be divided into five components, as presented below.

- total waiting time at the last stage (*total_wait*)
- total processing time of all products at the last stage (*total_proc*)
- total major setup time at the last stage (*total_major*)
- total minor setup time at the last stage (*total_minor*)
- adjustments to setup times at the last stage (*adjust_setup*)

A detailed description of these components and how they are used to calculate LB^F and LB^B is presented in sections 6.3.1 and 6.3.2, respectively. The optimal makespan cannot be less than the sum of the above five components divided by the number of machines in the last stage. Hence, using the forward method:

$$LB^F = \frac{1}{m(S)}[total\_wait + total\_proc + total\_major + total\_minor + adjust\_setup]$$

Similarly, for the backward method:

$$LB^B = \frac{1}{m(1)}[total\_wait + total\_proc + total\_major + total\_minor + adjust\_setup]$$

6.2.1 Forward Method

1. Total waiting time at the last stage (total_wait)

The *total_wait* is the minimum amount of time that the machines at the last stage have to wait until their first products are processed. This means that the first *m(S)* products have to complete their processing on stage 1 through stage *S*-1. Two cases are considered in calculating the *total_wait.*

Case 1: m(S) $\leq$ *m*(1)

The *total_wait* is determined by summing the first λ, λ = *m(S),* smallest values of ICT(*i*).

Hence:

$$total\_wait = \sum_{i \in A} ICT(i)$$

Case 2: m(S) > m(1)

In this case, the machines in stage *S* are divided into two groups. The first group contains *m(1)* machines, and the second contains

m(S) – m(1) machines (i.e. *xtra(1)*). The total waiting time for the machines in the first group (*waiting_time_g1*) is calculated as the sum of the first λ smallest values of ICT(i): $\sum_{i \in A} ICT(i)$. For the second group, the ratio (R) between *xtra(1)* and *m(1)* is determined and will be used to calculate the machine waiting times (*waiting_time_g2*). The value of R is determined as $\left\lceil \frac{m(S) - m(1)}{m(1)} \right\rceil$. Two cases are considered in calculating the machine waiting times in this group: (1) R = 1, and (2) R > 1. Details for each of these cases are described below.

2.1 R = 1

The following procedure is followed:

Let

$\Omega(i)$ = SI(i) + P(i,1); $i \in NP$

$\beta(i)$ = min {min{MN(p,1)}, MN(i,1)} + CT(i)

where, $p \in A$ and $i \in K$

2.1.1 Let x be the machine number in the second group, x = 1,2,…, *xtra(1)*. Set x = 1.

2.1.2 Determine the machine waiting time on machine x using the following steps.

2.1.2.1 Sort all values of $\Omega(i)$ in non-decreasing order. Let $\Omega_{[1]}$, $\Omega_{[2]}$, $\Omega_{[3]}$,…, $\Omega_{[np]}$ be the values resulting from the order. Then, find the product with the first lowest value of $\Omega(i)$ (e.g., product k):

$$\Omega(k) = \Omega_{[1]} = \min_{i \in NP} \Omega(i)$$

2.1.2.2 Sort all values of $\beta(i)$ in non-decreasing order. Let $\beta_{[1]}, \beta_{[2]}, \beta_{[3]},\ldots, \beta_{[k]}$ be the values resulting from the order. Then, find the product with the first lowest value of $\beta(i)$ (e.g., product g):

$$\beta(g) = \beta_{[1]} = \min_{i \in K} \beta(i)$$

2.1.2.3 Check if $k = g$. If not true, calculate *waiting_time(x)* and update set *NP* as follows.

$$waiting\_time(x) = \Omega(k) + \beta(g)$$

$$NP = NP \setminus \{k\},$$ delete $\beta(g)$

and go to step 2.1.3; otherwise, go to step 2.1.2.4.

2.1.2.4 Find the product with the second lowest value of $\Omega(i)$ (e.g., product k'):

$$\Omega(k') = \Omega_{[2]} = \min_{i \in NP \setminus \{k\}} \Omega(i)$$

2.1.2.5 Find the product with the second lowest value of $\beta(i)$ (e.g., product g'):

$$\beta(g') = \beta_{[2]} = \min_{i \in NP \setminus \{g\}} \beta(i)$$

2.1.2.6 Calculate the minimum waiting time on machine x (*waiting_time(x)*) as follows:

$$waiting\_time(x) = \min \{\Omega(k) + \beta(g'), \Omega(k') + \beta(g)\}$$

2.1.2.7 If $\Omega(k) + \beta(g') < \Omega(k') + \beta(g)$, update $K = K - \{k\}$ and delete $\beta(g')$.

Otherwise, update $K = K - \{k'\}$ and delete $\beta(g)$.

2.1.3 Update $x = x + 1$. If x is greater than $m(S) - m(1)$, go to step 2.1.4; otherwise, go back to step 2.1.2.

2.1.4 Calculate *total_wait* as follows:

$$total\_wait = \sum_{i \in A} ICT(i) + \sum_{x=1}^{m(S)-m(1)} waiting\_time(x)$$

2.2 $R > 1$

For this case, the machines in the second group are divided into smaller subgroups of $m(1)$ machines (the last subgroup may have a smaller number). The minimum waiting tine of the machines in the first subgroup (i.e., machine number $m(1)+1$, $m(1)+2$, …, $2m(1)$) is determined using the procedure detailed in case 2.1 (i.e., $R = 1$). To calculate the minimum waiting time for the machines of the remaining subgroups, the same procedure is repeated with the following modifications.

(1) Function $\Omega(i)$ is replaced with function $\alpha(i, w_1, w_2, \ldots, w_r)$ which is defined as follows.

$$\alpha(i, w_1, w_2, \ldots, w_r) = SI(i) + P(i,1) + \sum_{\sigma=1}^{r} \{MN(w_\sigma,1) + P(w_\sigma,1)\}$$

where, $i, w_\sigma \in$ NP, $\sigma = 1,2,\ldots,r$, $i \neq w_1 \neq w_2, \ldots, \neq w_r$

To calculate the waiting time on each subgroup of machines in the last stage, function $\alpha(i,w_1,w_2,\ldots,w_r)$ must be regenerated for each r until the value of r reaches R-1. For instance, when r =1, the quantity $\alpha(i, w_1)$ is used to calculate the waiting time for the second subgroup of machines (i.e., machines $2 \cdot m(1)+1$, $2 \cdot m(1)+2$,…, $3 \cdot m(1)$). Likewise, when $r = R - 1$, the quantity $\alpha(i, w_1, w_2, \ldots, w_r)$ is used to calculate the

waiting time for the R^{th} subgroup of machines (i.e., machines $(R-1)\cdot m(1)+1,\ldots, m(S)$).

In step 2.1.2.1, all values of $\alpha(i, w_1, w_2, \ldots, w_r)$ obtained from all combinations of i and w_σ are sorted in non-decreasing order and let $\alpha_{[1]}$, $\alpha_{[2]}$, $\alpha_{[3]}, \ldots, \alpha_{[np]}$ be the values resulting from the order.

(2) In step 2.1.2.3 of Case 2.1, product g is checked to find if it is a member of set ϖ, where ϖ is set of products $(i, w_1, w_2, \ldots, w_r)$ that yielded $\alpha_{[1]}$.

(3) Steps 2.1.2.4 through 2.1.2.6 are modified to find the combination of $\alpha(\varpi)$ and $\beta(g)$ such that g is not a member of ϖ, which yield the minimum value of the sum of $\alpha(\varpi)$ and $\beta(g)$. Step 2.1.2.7 is then modified to update $K = K - \varpi$ and delete $\beta(g)$.

The value *total_wait* when $R > 1$ is calculated as follows:

total_wait = waiting_time_g1 + waiting_time_g2

$$= \sum_{i \in A} ICT(i) + \sum_{x=1}^{m(S)-m(1)} waiting\ \_time(x)$$

2. Total processing time of all products at the last stage (total_proc)

A lower bound of the total processing times on the machines at the last stage is calculated as the sum of the processing times of all products when processed on machines with the average speed in that stage. The value of *total_proc* is hence calculated as follows:

$$total\_proc = \frac{\sum_{i \in NP} T(i,S) \cdot m(S)}{\sum_{m \in M(S)} v_{S,m}}$$

A better (higher) lower bound may be calculated for *total_proc* by allowing preemption and applying the "Shortest Remaining Processing Time on Fastest Machine [SRPT-FM] rule; but this may take some effort and the improvement can be very little, especially when the ratio of the number of products to the number of machines is high.

3. Total major setup time at the last stage (total_major)

In minimizing major changeovers, the number of machines assigned to each family should be as few as possible. Major setups can be minimized by scheduling each family on only one machine. Thus, the minimum number of major setups for the entire production schedule on the last-stage machines is equal to *N - m(S)* setups. The value of *total_major* is hence determined as the sum of the *N – m(S)* smallest major changeovers.

$$total\_major = \sum_{i \in C} MJ(i,S)$$

4. Total minor setup times at the last stage (total_minor)

With each family assigned to only one machine, a total of *np - N* minor setups would be required. The *total_minor* is hence determined by summing the first *np – N* smallest minimum minor changeovers, as shown below.

$$total\_minor = \sum_{i \in B} MN(i,S)$$

5. Adjustments to setup times at the last stage (adjust_setup)

The lower bound on the total setup times at the last stage can be improved if some of the products in set B are also members of set C (i.e., $B \cap C = Z \neq \phi$). In

this case, some members of set D must replace members of either set C (major setup times) or set B (minor setup times), whichever yields a smaller difference.

Let $z \in Z$.

If a member $d \in D$ replaces z in set C, then the difference is calculated as follows:

mj_diff(d,z) = MJ(d, S) – MJ(z, S)

The minimum value mj_diff(d*,z*) is realized by selecting $\min_{d \in D}$ (MJ(d,*S*)) and $\max_{z \in Z}$ (MJ(z,*S*)). Denote $\max_{z \in Z}$ (MJ(z, S)) as MJMax.

Similarly, if d replaces z in set B, then the minimum difference mn_diff(d',z') = $\min_{d \in D}$ (MN(d,*S*) – $\max_{z \in Z}$ (MN(z,*S*)). Denote $\max_{z \in Z}$ (MN(z,S)) as MNMax. The minimum value between mj_diff(d*,z*) and mn_diff(d',z') is then added to *adjust_setup* (which has an initial value of zero). Product z* (or z') is then deleted from set Z and product d* (or d') is deleted from set D. However, the values of MJMax and MNMax should not be updated. This process is repeated until set Z is void.

The overall lower bound is then calculated as follows:

$$LB^F = \frac{1}{m(S)} \left[total\_wait + total\_proc + total\_major + total\_minor + adjust\_setup \right]$$

6.2.2 Backward Method

Consider a schedule where products are processed from stage *S* to stage 1 (i.e., reverse order of machines), then its antithetical schedule (mirror image) yields the same makespan for the original problem when no setup times are considered. With setup times, the lower bound for the backward schedule would still remain a

lower bound for the original problem, when calculated as in the forward method with the following two adjustments:

1. Setup times from idling for the first *m(S)* products in stage *S* must not be considered when calculating *total_wait* (i.e., assume SI(*i*) = 0 for all products, where SI(*i*) in this case is the setup time for product *i* from idling at stage *S*).
2. The sum of the *m(1)* minimum setup times from idling in stage 1 (*sum_setup_idle*) should be added to *total_wait*.

The backward lower bound will then be calculated as follows:

$$LB^B = \frac{1}{m(1)} [total\_wait + total\_proc + total\_major + total\_minor + adjust\_setup]$$

The best lower bound (BLB) is then determined as max $\{LB^F, LB^B\}$.

6.3 Illustration of the Lower Bound Calculations

The problem presented in Chapter 5 is used here to demonstrate the calculation of the lower bound.

Number of families: J = 4

Number of stages: S = 3

Number of products: f_j = 3, *j* = 1, 2, 3, 4

Number of machines: *m(1)* =3, *m(2)* = 2, and *m(3)*= 2

Processing times of each product on the fastest machine at each stage (P(i,s)) and changeover times of each product in terms of setup times from idling (SI(i)), major (MJ(i,s)) and minor (MN(i,s)) setup times in each stage are shown in Table 6.1.

Table 6.1: Processing Times on the Fastest Machine at each Stage and Changeover Times of Each Product on Each Stage

Description	Family											
	1			2			3			4		
	Product			Product			Product			Product		
	1	2	3	1	2	3	1	2	3	1	2	3
Processing Time (P(i,s))												
s=1	43.35[1]	16.54	24.14	31.56	28.71	30.39	19.11	25.19	29.62	39.21	14.60	21.24
s=2	23.74	11.07	33.01	11.94	11.31	16.59	14.99	43.76	25.47	36.55	33.76	33.46
s=3	30.63	23.40	26.97	45.75	32.31	21.47	31.43	12.14	19.10	46.01	26.30	16.74
Setup time From idle (SI(i))	5.97	6.27	4.53	7.48	6.29	7.00	5.75	7.37	5.74	5.93	6.16	5.54
Minor Setup time (MN(i,s))												
s=1	4.19[2]	1.6	4.13	3.24	2.34	2.27	1.97	2.9	1.57	4.35	1.65	1.82
s=2	2.82	2.34	4.24	3.83	3.82	2.4	2.94	3.41	2.19	2.78	1.59	3.92
s=3	2.91	1.76	3.52	2.33	1.65	3.47	3.27	2.67	1.76	3.69	2.75	3.13
Major setup time (MJ(i,s))[3]												
s=1	6.43[3]	8.65	6.26	6.23	7.37	6.01	6.02	6.51	6.42	6.35	6.24	6.21
s=2	7.38	7.02	6.84	6.23	6.06	8.37	6.60	6.33	6.05	6.06	7.30	8.07
s=3	6.64	6.02	6.50	7.19	6.12	6.22	6.00	6.11	6.11	6.37	6.22	6.93

Note:

[1] (47.68/1.1) = 43.35
[2] MN(1,1) = min {4.19, 4.28}
[3] MJ(1,1) = min {11.06, 8.6, 6.51, 6.43, 7.19, 7.82, 10.61, 10.91, 8.94}

6.3.1 Lower bound Calculations Based on Forward Method:

Calculations of the total waiting time at the last stage (*total_wait*)

In this problem, the value of *m*(3) is less than *m*(1), hence λ = *m(3)* = 2. The *total_wait* is determined as:

$$total\_wait = \sum_{i \in A} ICT(i)$$

From the data obtained in Table 6.1, the summations of idle time and processing time of each product from stages *s* = 1 through *S-1* are presented in Table 6.2.

Table 6.2: The Summations of Setup Time from Idling of the First Stage and Cumulative Processing Times of Each Product on the Fastest Machine from Stages 1 through *S-1*

Family j (1)	Product i (2)	SI(*i*) (time units) (3)	CT(*i*) (time units) (4)	SI(*i*) + CT(*i*) (time units) (3)+(4)
1	1	5.97	67.09	73.06
	2	6.27	27.61	33.88
	3	4.53	57.15	61.68
2	1	7.48	43.50	50.98
	2	6.29	40.02	46.31
	3	7.00	46.98	53.98
3	1	5.75	34.10	39.85
	2	7.37	68.95	76.32
	3	5.74	55.09	60.83
4	1	5.93	75.76	81.69
	2	6.16	48.36	54.52
	3	5.54	54.70	60.24

From Table 6.2, it is obvious that the lowest two values of the sum of SI(*i*) and CT(*i*) are 33.88 and 39.85 time units. These values belong to product 2 of family 1 and product 1 of family 3, respectively. Hence, A = {(1,2), (3,1)}, and

total_wait = (33.88 + 39.85)

= 73.73 time units

Calculations of the total processing time of all products at the last stage (*total_proc*)

$$total\_proc = [\sum_{i \in NP} T(i,3) \cdot m(3)] / \sum_{m \in M(3)} v_{s,m}$$

From Tables 5.1 and 5.2 in chapter 5, the values of $\sum_{m \in M(S)} v_{s,m}$ and that of $\sum_{i \in NP} T(i,3)$ are equal to 2.06 and 352.19, respectively. Hence, the total processing time of all products from all families at the last stage is presented as follows:

total_proc = [2 x 352.74] / 2.06

= 341.93 time units

Calculations of the total major setup time at the last stage (*total_major*)

$$total\_major = \sum_{i \in C} MJ(i,3)$$

From Table 6.1, the lowest two major setup times at the last stage are 6.00 and 6.02 time units belonged to product 1 of family 3 and product 2 of family 1, respectively. Hence, C = {(3,1), (1,2)} and

total_major = 6.00 + 6.02

= 12.02 time units

Calculations of the total minor setup time at the last stage (*total_minor*)

$$total\_minor = \sum_{i \in B} MN(i,3)$$

From Table 6.1, the lowest eight minor setup times at the last stage are presented below:

1.65 time units from product 2 of family 2,

1.76 time units from product 2 of family 1,

1.76 time units from product 3 of family 3,

2.33 time units from product 1 of family 2,

2.67 time units from product 2 of family 3,

2.75 time units from product 2 of family 4,

2.91 time units from product 1 of family 1, and

3.13 time units from product 3 of family 4.

Hence, B = {(2,2), (1,2), (3,3), (2,1), (3,2), (4,2), (1,1), (4,3)}

total_minor = 1.65+1.76+1.76 + 2.33 + 2.67 + 2.75 + 2.91 + 3.13

= 18.96 time units

Calculations of the adjustments to setup time at the last stage (*adjust_setup*)

From the previous calculations of the major and minor setup times, the products in the different sets are presented below:

Products in Set B: {(2,2), (1,2), (3,3), (2,1), (3,2), (4,2), (1,1), (4,3)}

Products in set C: {(3,1), (1,2)}

Products in set Z = B $\cap$ C: {(1,2)}

Products in set D = NP - (B $\cup$ C): {(2,3), (4,1), (1,3)}

The adjustments to the setup times for this problem are calculated as follows:

MJMax = $\max_{z \in Z}$ MJ(z,3)

= 6.02

mj_diff(d*,z*) = $\min_{d \in D}$ (MJ(d,3) – MJMax

= min {6.5, 6.22, 6.37} – 6.02

= 0.20 time units

d* = (4,1) and z* = (1,2)

Similarly, MNMax = $\max_{z \in Z}$ MN(z,3)

= 1.76

mn_diff(d', z') = $\min_{d \in D}$ MN(d,3) – MNMax

= min {3.52, 3.47, 3.69} – 1.76

= 1.71 time units

d' = (4,1) and z' = (1,2)

Hence, *adjust_setup* = min {0.20, 1.71}

= 0.20 time units

And Z = ϕ.

After all five components have been determined, the lower bound, using the forward method, is calculated as follows:

LB^F = ½ (73.73 + 341.93 + 12.02 + 18.96 + 0.20)

= 223.42 time units

Hence, LB^F = 224 time units

6.3.2 Lower bound Calculations Based on Backward Method

Calculations of the total waiting time at the first stage (*total_wait*)

In this example, *m*(1) > *m*(3), hence *total_wait* in this case is:

total_wait = waiting_time_g1 + waiting_time_g2 + sum_setup_idle

$$= \sum_{i \in E} CT(i) + \sum_{x=1}^{m(1)-m(3)} waiting\_time(x) + \sum_{i \in G} SI(i)$$

From the data obtained in Table 6.1, the summations of the processing times of each product from stages *s* = 3 to 2 are presented in Table 6.3. From this table, it is obvious that the lowest two values of the sum of the total processing times from stages 3 to stage 2 are 34.47 and 38.06 time units. These values belong to product 2 of family 1 and product 3 of family 2, respectively. Hence, E = {(1,2), (2,3)}, and

waiting_time_g1 = 34.47 + 38.06 = 72.53 time units.

Table 6.3: The Values of CT(i) and β(i) Used to Calculate the Backward Lower Bound

Family j (1)	Product i (2)	min {1.76, MN(i,1)} (time units) (3)	P(i,2) (time units) (4)	P(i,3) (time units) (5)	CT(i) (time units) (4) +(5)	β(i) (time units) (3) + (4) + (5)
1	1	1.76	23.74	30.63	54.37	56.13
	2	1.76	11.07	23.40	34.47	36.23
	3	1.76	33.01	26.97	59.98	61.74
2	1	1.76	11.94	45.75	57.69	59.45
	2	1.65	11.31	32.31	43.62	45.27
	3	1.76	16.59	21.47	38.06	39.82
3	1	1.76	14.99	31.43	46.42	48.18
	2	1.76	43.76	12.14	55.90	57.66
	3	1.76	25.47	19.10	44.57	46.33
4	1	1.76	36.55	46.01	82.56	84.32
	2	1.76	33.76	26.30	60.06	61.82
	3	1.76	33.46	16.74	50.20	51.96

The waiting time on the second group machines is determined as follows.

$R = \lceil (3\text{-}2)/2 \rceil = 1$, hence case 2.1 is applied.

$\Omega(i) = SI(i) + P(i,1);\ i \in NP$

$\beta(i) = \min \{\min\{MN(p,1)\}, MN(i,1)\} + CT(i)$

where, $p \in A$ and $i \in K$

2.1.1 Set x = 1.

2.1.2 Calculation steps:

2.1.2.1 Since the setup time from idling in the last stage is not considered, the value of $\Omega(i)$ is $P(i,1)$. Hence, the

lowest value of $\Omega(i)$ is $\Omega((3,2))$ which is equal to 12.14.

2.1.2.2 To determine the value of $\beta(i)$, the minimum value of MN(p,1) is 1.76 time units. The values of $\beta(i)$ are shown in Table 6.3. From this table, the lowest value of $\beta(i)$ (i.e., $\beta(g)$) is $\beta((1,2))$ which is equal to 36.23 time units.

2.1.2.3 Since (3,2) $\neq$ (1,2), then the waiting time on the second group machine is determined below.

waiting_time(1) = 12.14 + 36.23

= 48.37 time units

Then, go to step 2.1.3.

2.1.3 Update x = x+1 = 2. Since x is greater than *m(1) – m(S)*, go to 2.1.4.

2.1.4 The sum of the lowest three setup times from idling at stage 1 ($\sum_{i \in G} SI(i)$) is equal to 15.81 (i.e., 4.53 + 5.54 + 5.74 = 15.81) time units. The value of *total_wait* is calculated as follows:

total_wait = 72.53 + 48.23 + 15.81

= 136.57 time units

Calculations of the total processing time of all products at the first stage (*total_proc*)

$$total\_proc = \sum_{i \in NP} T(i,1) \cdot m(1) / [\sum_{m \in M(1)} v_{s,m}]$$

From Tables 5.1 and 5.2 in chapter 5, the values of $\sum_{m \in m(1)} v_{1,m}$ and $\sum_{i \in NP} T(i,1)$ are 3.13 and 356.01, respectively. Hence, the total processing time of all products from all families at the last stage is calculated as follows:

total_proc = (356.01 x 3)/ 3.13

= 341.22 time units

Calculations of the total major setup time at the first stage (*total_major*)

$$total\_major = \sum_{i \in C} MJ(i,1)$$

where, C has 4 – 3 = 1 family.

From Table 6.1, the lowest major setup time at the first stage is 6.01 time units belonged to product 3 of family 2. Hence, C = {(2,3)}, and

total_major = 6.01 time units

Calculations of the total minor setup time at the first stage (*total_minor*)

$$total\_minor = \sum_{i \in B} MN(i,1)$$

where, B has 12 – 4 = 8 products.

From Table 6.1, the lowest eight minor setup times at the first stage are presented below:

1.57 time units from product 3 of family 3,

1.60 time units from product 2 of family 1,

1.65 time units from product 2 of family 4,

1.82 time units from product 3 of family 4,

1.97 time units from product 1 of family 3,

2.27 time units from product 3 of family 2,

2.34 time units from product 2 of family 2, and

2.90 time units from product 2 of family 3.

Hence, B = {(3,3), (1,2), (4,2), (4,3), (3,1), (2,3), (2,2), (3,2)}

total_minor = 1.57 + 1.60 + 1.65 + 1.82 + 1.97 + 2.27 + 2.34 + 2.90

= 16.12 time units

Calculations of the total adjustments to setup times at the first stage (*adjust_setup*)

From the previous calculations of the major and minor setup times, the products in the different sets are presented below:

Products in Set B: {(3,3), (1,2), (4,2), (4,3), (3,1), (2,3), (2,2), (3,2)}

Products in set C: {(2,3)}

Products in Set D: {(1,1), (1,3), (2,1), (4,1)}

Products in set Z: {(2,3)}

The adjustments to the setup times for this problem are calculated as follows:

MJMax = $\max_{z \in Z}$ MJ(z,1)

= 6.01

mj_diff(d*,z*) = $\min_{d \in D}$ (MJ(d,1) – MJMax

= 6.23 – 6.01

= 0.22 time units

d* = (2,1) and z* = (2,3)

Similarly, MNMax = $\max_{z \in Z}$ MN(z,1)

= 2.27

mn_diff(d', z') = $\min_{d \in D}$ MN(d,1) – MNMax

= 3.24 – 2.27

= 0.97 time units

d' = (2,1) and z' = (2,3)

Hence, *adjust_setup* = min {0.22, 0.97}

= 0.22 time units

And Z = ϕ.

LB^B = 1/3 [136.57 + 341.22 + 6.01 + 16.12 + 0.22]

= 166.71 time units

Hence, LB^B = 167 time units.

The best lower bound for this problem (BLB) = max { LB^F, LB^B}

= max {224, 167}

= 224 time units

CHAPTER 7

COMPUTATIONAL EXPERIMENTS

7.1 Introduction

The flexible flowshop with sequence dependent setup time is known to be NP-hard. Obtaining an optimal solution using mathematical formulation would require large computational effort; hence, optimal solutions will not be investigated further. This chapter will focus on computational experience with the heuristic algorithms (FFSDSTH and TSH). Two quantities are investigated: (1) the performance of the heuristic algorithms, obtained by comparing their solutions to the lower bound and (2) the relative improvement of the solutions obtained by the FFSDSTH algorithm with respect to those of the TSH algorithm.

Two sets of problems, with six types of data characteristics in each set, were generated to evaluate the above two quantities:

Set 1: 50 products (12 families)

Set 2: 80 products (18 families)

Six types (A, B, C, D, E, and F) of data characteristics were generated for each set, and 10 test problems were generated for each data type. The parameters for each data type, processing times of products on a standard machine (speed = 1) at each stage (PTime(j,i,s,m)), machine speed deviations ($v_{s,m}$), changeover times between products at each stage (ch(j,i,q,p,s)), and setup times from idling of products at the first stage (ch(0,0,j,i,s)), were randomly selected from different uniform distributions as shown in Table 7.1

Table 7.1: Values of Parameters Used with the Different Data Types

Parameter	Type					
	A	B	C	D	E	F
Total number of machines and stages	9 machines, 3 stages (3,3,3)	20 machines, 5 stages (4,4,4,4,4)	11 machines, 3 stages (4,2,5)	9 machines, 3 stages (3,3,3)	20 machines, 5 stages (4,4,4,4,4)	11 machines, 3 stages (4,2,5)
PTime(j,i,s,m)	U[10,50]	U[10,50]	U[10,50]	U[10,50]	U[10,50]	U[10,50]
$v_{s,m}$	U[0.85, 1.15]	U[0.85, 1.15]	U[0.85, 1.15]	U[0.75, 1.25]	[0.75, 1.25]	U[0.75, 1.25]
ch(j,i,q,p,s)	U[20%, 40%] of PTime(j,i,s,m)	U[20%, 40%] of PTime(j,i,s,m)	U[20%, 40%] of PTime(j,i,s,m)	U[20%, 40%] of PTime(j,i,s,m)	U[20%, 40%] of PTime(j,i,s,m)	U[20%, 40%] of PTime(j,i,s,m)
ch(j,i,j,p,s)	U[5%, 15%] of PTime(j,i,s,m)	U[5%, 15%] of PTime(j,i,s,m)	U[5%, 15%] of PTime(j,i,s,m)	U[5%, 15%] of PTime(j,i,s,m)	U[5%, 15%] of PTime(j,i,s,m)	U[5%, 15%] of PTime(j,i,s,m)
ch(0,0,j,i,s)	U[15%, 25%] of PTime(j,i,s,m)	U[15%, 25%] of PTime(j,i,s,m)	U[15%, 25%] of PTime(j,i,s,m)	U[15%, 25%] of PTime(j,i,s,m)	U[15%, 25%] of PTime(j,i,s,m)	U[15%, 25%] of PTime(j,i,s,m)

Changeover times between products at each stage (ch(j,i,q,p,s) and setup times from idling at the first stage (ch(0,0,j,i,s)) are identical on all machines at the same stage. Types A, B, and C generate problems with small deviations in the speed of machines. Conversely, types D, E, and F generate problems with large deviations in the speeds. Characteristics of the data types can be summarized as follows:

A: A small number of stages, small deviations in machine speeds, and small, identical number of machines in each stage.

B: A large number of stages, small deviations in machine speeds, and large, identical number of machines in each stage.

C: A small number of stages, small deviations in machine speeds, and small, non-identical number of machines in each stage.

D: A small number of stages, large deviations in machine speeds, and small, identical number of machines in each stage.

E: A large number of stages, large deviations in machine speeds, and large, identical number of machines in each stage.

F: A small number of stages, large deviations in machine speeds, and small, non-identical number of machines in each stage.

In section 7.2, the computational results obtained with the heuristics are presented and compared to the lower bounds for the large size problems. Section 7.3 presents the relative improvement of the solutions obtained by the FFSDSTH algorithm with the application of the TSH algorithm.

7.2 Comparison of the Results of Heuristic Algorithms with the Lower Bounds

The heuristic algorithms were coded in C++ and run on a 300 MHz PC, with 96 MegaBytes of RAM, for testing and evaluation. In this section, the heuristic algorithms are evaluated using two performance measures: (1) solution quality, and (2) computational speed. The quality of a solution generated by the heuristics is measured in terms of their performance (HP), as presented below.

$$HP = (sol_{LB}/sol_{heu}) \times 100$$

where,

HP = the heuristic performance (%)

sol_{LB} = the lower bound of the solution

sol_{heu} = the solution obtained from the heuristic algorithms

The computational speed of the algorithms is measured by the amount of CPU time required to execute the algorithms. The CPU time includes compiling, linking, and

execution times, and is reported in seconds and seconds per iteration for the FFSDSTH and TSH algorithms, respectively.

For each combination of problem set and data type, ten different test problems were generated. The solution of each test problem using the heuristic algorithm and its lower bound were obtained for all combinations of sets and data types. The results of these computations are presented in Tables 7.2-7.13. Table 7.14 shows the averages obtained for these results.

Table 7.2: Computational Results for Set 1 Type A: Heuristic Algorithms vs. Lower Bound

Problem Number	CPU Time			Heuristic Performance (%)	
	FFSDSTH (seconds)	TSH		FFSDSTH	TSH
		seconds/iteration	Number of Iterations (iterations)		
1	1.2	10.90	50	81.463	88.485
2	1.3	10.60	48	74.322	79.616
3	1.3	10.60	45	85.698	91.364
4	1.2	10.40	69	79.576	85.127
5	1.4	10.90	91	74.400	80.286
6	1.5	10.80	38	85.392	90.700
7	1.3	10.70	43	85.243	92.651
8	1.4	10.60	44	82.684	90.368
9	1.4	10.60	89	80.403	86.788
10	1.2	10.50	66	82.049	90.265

Table 7.3: Computational Results for Set 1 Type B:
Heuristic Algorithms vs. Lower Bound

Problem Number	CPU Time			Heuristic Performance (%)	
	FFSDSTH (seconds)	TSH		FFSDSTH	TSH
		seconds/iteration	Number of Iterations (iterations)		
1	1.4	29.20	63	79.611	89.156
2	1.3	28.70	48	80.244	87.251
3	1.4	28.80	47	80.756	86.361
4	1.5	28.70	60	79.442	84.660
5	1.4	29.30	58	79.245	84.955
6	1.5	28.90	58	81.353	86.972
7	1.3	29.50	80	76.119	82.724
8	1.4	29.70	49	74.618	80.249
9	1.6	28.50	42	82.102	88.748
10	1.5	29.90	65	79.576	86.824

Table 7.4: Computational Results for Set 1 Type C:
Heuristic Algorithms vs. Lower Bound

Problem Number	CPU Time			Heuristic Performance (%)	
	FFSDSTH (seconds)	TSH		FFSDSTH	TSH
		seconds/iteration	Number of Iterations (iterations)		
1	1.8	11.50	52	86.812	91.170
2	1.6	11.20	49	80.613	85.890
3	1.7	11.30	65	83.836	88.659
4	1.9	11.40	65	81.935	88.358
5	1.8	10.90	49	80.302	86.184
6	1.9	11.10	84	80.852	86.725
7	1.8	11.40	38	80.916	88.698
8	1.6	11.60	68	84.384	90.345
9	1.6	11.50	55	85.291	90.926
10	1.7	11.20	33	81.817	87.446

Table 7.5: Computational Results for Set 1 Type D:
Heuristic Algorithms vs. Lower Bound

Problem Number	CPU Time			Heuristic Performance (%)	
	FFSDSTH (seconds)	TSH		FFSDSTH	TSH
		seconds/iteration	Number of Iterations (iterations)		
1	1.4	10.80	51	81.463	88.485
2	1.5	10.60	63	74.322	79.616
3	1.3	10.70	39	85.698	91.364
4	1.5	10.80	52	79.576	85.127
5	1.2	10.80	67	74.400	80.286
6	1.5	10.60	80	85.392	90.700
7	1.4	11.10	34	85.243	92.651
8	1.2	11.00	36	82.684	90.368
9	1.3	10.70	52	80.403	86.788
10	1.6	10.70	64	82.049	90.265

Table 7.6: Computational Results for Set 1 Type E:
Heuristic Algorithms vs. Lower Bound

Problem Number	CPU Time			Heuristic Performance (%)	
	FFSDSTH (seconds)	TSH		FFSDSTH	TSH
		seconds/iteration	Number of Iterations (iterations)		
1	1.9	30.20	47	72.721	81.117
2	1.8	29.80	47	71.241	77.730
3	1.8	29.90	79	77.481	83.921
4	2.0	30.10	70	73.650	78.533
5	2.1	30.00	77	76.324	81.758
6	1.8	29.70	40	73.338	82.765
7	1.7	29.20	39	70.361	79.638
8	1.6	30.40	64	74.031	82.585
9	1.6	30.50	46	70.099	79.522
10	2.1	29.50	36	73.438	79.352

Table 7.7: Computational Results for Set 1 Type F:
Heuristic Algorithms vs. Lower Bound

Problem Number	CPU Time			Heuristic Performance (%)	
	FFSDSTH (seconds)	TSH		FFSDSTH	TSH
		seconds/iteration	Number of Iterations (iterations)		
1	1.4	12.00	53	74.972	81.550
2	1.5	12.00	55	79.406	87.135
3	1.3	11.70	32	78.046	83.117
4	1.5	11.80	78	76.516	83.759
5	1.5	12.00	64	75.047	83.983
6	1.7	12.00	79	79.438	87.860
7	1.4	11.80	38	73.959	83.066
8	1.6	11.60	48	80.097	84.536
9	1.7	11.50	50	73.708	81.224
10	1.3	11.60	52	77.924	87.810

Table 7.8: Computational Results for Set 2 Type A:
Heuristic Algorithms vs. Lower Bound

Problem Number	CPU Time			Heuristic Performance (%)	
	FFSDSTH (seconds)	TSH		FFSDSTH	TSH
		seconds/iteration	Number of Iterations (iterations)		
1	1.9	42.00	35	82.545	86.052
2	2.0	42.30	73	89.203	93.335
3	2.2	42.30	68	79.945	83.556
4	2.0	41.80	34	82.587	85.287
5	2.1	41.70	42	81.262	83.822
6	1.8	41.90	68	85.502	89.785
7	1.7	42.00	92	83.549	87.084
8	2.0	42.00	80	84.135	87.464
9	1.9	42.50	65	80.573	84.459
10	1.8	42.10	69	83.306	86.599

Table 7.9: Computational Results for Set 2 Type B:
Heuristic Algorithms vs. Lower Bound

Problem Number	CPU Time			Heuristic Performance (%)	
	FFSDSTH (seconds)	TSH		FFSDSTH	TSH
		seconds/iteration	Number of Iterations (iterations)		
1	2.4	121.00	40	77.985	82.079
2	2.3	122.40	47	76.963	81.000
3	2.7	119.80	80	76.169	81.784
4	2.1	119.40	32	78.279	83.382
5	2.2	119.80	37	73.902	79.459
6	2.4	121.40	54	76.708	81.632
7	2.5	119.10	39	71.015	77.274
8	2.6	122.00	40	75.044	78.824
9	2.7	119.00	39	77.164	81.636
10	2.3	120.00	80	74.787	79.803

Table 7.10: Computational Results for Set 2 Type C:
Heuristic Algorithms vs. Lower Bound

Problem Number	CPU Time			Heuristic Performance (%)	
	FFSDSTH (seconds)	TSH		FFSDSTH	TSH
		seconds/iteration	Number of Iterations (iterations)		
1	1.8	54.30	54	79.432	84.766
2	2.0	55.00	59	79.921	85.334
3	2.1	55.10	66	79.689	83.490
4	2.0	55.20	94	90.751	95.583
5	2.1	55.00	80	78.983	82.120
6	2.2	54.30	33	79.306	84.550
7	2.3	56.10	61	78.511	84.095
8	1.9	49.70	42	80.116	85.185
9	2.1	55.00	34	80.083	84.636
10	1.8	55.00	57	79.041	83.844

Table 7.11: Computational Results for Set 2 Type D:
Heuristic Algorithms vs. Lower Bound

Problem Number	CPU Time			Heuristic Performance (%)	
	FFSDSTH (seconds)	TSH		FFSDSTH	TSH
		seconds/iteration	Number of Iterations (iterations)		
1	1.9	44.00	62	76.373	81.358
2	1.8	44.00	38	75.985	80.374
3	2.1	44.50	80	84.538	87.975
4	2.0	44.20	96	75.046	79.265
5	2.2	43.90	52	79.512	84.750
6	2.1	43.80	42	80.065	87.427
7	2.1	44.00	34	72.910	77.382
8	1.9	44.10	41	75.276	80.168
9	1.9	44.50	98	80.829	87.815
10	2.0	44.30	92	79.116	83.524

Table 7.12: Computational Results for Set 2 Type E:
Heuristic Algorithms vs. Lower Bound

Problem Number	CPU Time			Heuristic Performance (%)	
	FFSDSTH (seconds)	TSH		FFSDSTH	TSH
		seconds/iteration	Number of Iterations (iterations)		
1	2.3	123.00	39	69.419	74.763
2	2.2	123.70	40	73.694	81.635
3	2.4	123.30	54	73.802	80.123
4	2.1	123.90	33	72.369	79.595
5	2.3	122.80	80	71.746	77.625
6	2.1	122.50	42	73.947	78.457
7	2.2	123.00	63	72.906	79.426
8	2.3	123.70	65	68.858	74.331
9	2.2	122.00	54	64.291	70.117
10	2.1	122.70	53	69.923	75.175

Table 7.13: Computational Results for Set 2 Type F:
Heuristic Algorithms vs. Lower Bound

Problem Number	CPU Time			Heuristic Performance (%)	
	FFSDSTH (seconds)	TSH		FFSDSTH	TSH
		seconds/iteration	Number of Iterations (iterations)		
1	2.5	57.00	38	72.073	77.789
2	2.5	57.40	45	72.914	79.415
3	2.6	57.10	76	74.506	79.341
4	2.8	56.90	68	73.253	80.775
5	2.9	57.00	80	75.267	79.050
6	2.5	56.80	40	74.360	81.395
7	2.5	56.40	47	70.350	77.351
8	2.6	57.00	70	70.000	76.455
9	2.8	57.00	57	75.491	79.847
10	2.5	57.30	47	74.442	80.141

Table 7.14: Averages of Computational Results for Sets 1 and 2 for all Data Types:
Heuristic Algorithms vs. Lower Bound

Set	Type	CPU time			Heuristic Performance (%)	
		FFSDSTH (seconds)	TSH		FFSDSTH	TSH
			seconds/iteration	Number of iterations (iterations)		
1	A	1.3	10.66	59	86.309	90.876
	B	1.7	29.12	57	79.307	88.790
	C	1.4	11.31	56	82.676	88.440
	D	1.4	10.78	54	81.123	87.565
	E	1.5	29.93	55	73.268	80.692
	F	1.8	11.80	55	76.911	84.404
2	A	1.9	42.06	63	83.261	86.744
	B	2.4	120.39	49	75.802	80.687
	C	2.0	54.47	58	80.583	85.360
	D	2.0	44.13	64	77.965	83.004
	E	2.6	123.06	53	71.096	77.125
	F	2.2	56.99	57	73.266	79.156

Based on these results, the average performance for set 1 ranges between 73.3-86.3% for the FFSDSTH algorithm and 80.7-90.9% for the TSH algorithm. For set 2, the average performance is lower than that of set 1, and ranges between 71.1-83.3% for the FFSDSTH algorithm and 77.1-86.7% for the TSH algorithm.

The computational times for the FFSDSTH are extremely small-- less than 3 seconds. These times do not significantly increase with the size of the problem. This means that the FFSDSTH algorithm is very efficient, and more importantly it is not sensitive to the problem size. In contrast, computational times for the TSH algorithm seem to be high-- between 10 and 30 seconds per iteration for data set 1 and between 42 and 124 seconds per iteration for data set 2. These times increase significantly with the size of the problem in terms of numbers of products (families), stages, and machines.

A Factorial Design was used to evaluate the performance of the heuristic algorithms (HP). The design has three factors: deviations in machine speeds, number of products, and number of machines and stages. The analysis was performed using SAS Software V8 for Windows and the results are presented in Appendix C. The statistical results show a significant effect for each of the three factors on the heuristic performance. Tukey's test was performed to compare between the three means obtained with different number of machines and stages. Results of the test (see Appendix C, Section C.3) indicate that the three means are different from each other.

The statistical results obtained from ANOVA and Tukey's test show that the heuristic performance declines with the increase of: (1) number of products, (2) number of machines and stages, and (3) deviation in machine speeds. This decline is due mainly to the decrement in the value of the lower bound rather than the performance of the heuristics. The lower bound value may be affected by the following factors:

(1) the difference between the actual processing times and the smallest processing times of products used to calculate the first component of lower bound. The difference in processing times gets larger when the difference in the speeds between the fastest and the slowest machines increases.

(2) the difference between actual processing times and the processing times on the average speed machine of products used to calculate the second component of the lower bound, and

(3) the difference between actual setup times (both major and minor setup times) and the smallest setup times of the products, used to calculate components 3,4, and 5 of the lower bound.

If the differences were small, the lower bound would be relatively high resulting in higher algorithm performance, and vice versa. Larger deviations in machine speeds, a number of products (families), and of machines and stages would most probably cause larger differences in processing times and setup times.

7.3 Comparison between the FFSDSTH Algorithm and the TSH Algorithm

In this section, the relative improvement of the solutions obtained from the FFSDSTH algorithm after applying the TSH is evaluated and presented below.

Let RI = $\{(sol_{FFSDSTH/} - sol_{TSH}) / sol_{FFSDSTH}\} \times 100$

where,

RI = the relative improvement (%) between $sol_{FFSDSTH}$ and sol_{TSH}

$sol_{FFSDSTH}$ = the solution obtained from the FFSDSTH algorithm

sol_{TSH} = the solution obtained from the TSH algorithm

Two sets of relatively large size problems are used in this section. These sets are identical to those described in Section 7.2. For each combination of problem set and data type, 10 different test problems were generated. The solutions of each test

problem using the FFSDSTH and TSH algorithms were obtained for all combinations of sets and data types. The results obtained are presented in Tables 7.15 and 7.16. Table 7.17 shows the averages obtained for these results.

Table 7.15: Relative Improvement Results for the Different Data Types in Set 1:

Problem Number	Relative Improvement (%)					
	Type					
	A	B	C	D	E	F
1	4.220	10.706	4.780	7.936	10.351	8.066
2	7.706	8.031	6.143	6.650	8.348	8.870
3	5.649	6.490	5.440	6.201	7.675	6.102
4	3.992	6.164	7.269	6.521	6.218	8.647
5	5.945	6.721	6.825	7.331	6.647	10.639
6	3.573	6.461	6.771	5.853	11.390	9.586
7	2.948	7.985	8.774	7.995	11.649	10.963
8	5.601	7.017	6.598	8.503	10.358	5.250
9	7.511	7.489	6.198	7.357	11.850	9.253
10	3.059	8.348	6.437	9.102	7.453	11.258

Table 7.16: Relative Improvement Results for the Different Data Types in Set 2:

Problem Number	Relative Improvement (%)					
	Type					
	A	B	C	D	E	F
1	4.075	4.987	6.293	6.127	7.148	7.348
2	4.427	4.983	6.344	5.460	9.728	8.186
3	4.322	6.865	4.552	3.907	7.889	6.093
4	3.166	6.120	5.055	5.322	9.078	9.313
5	3.054	6.994	3.820	6.180	7.573	4.786
6	4.771	6.032	6.202	8.420	5.749	8.643
7	4.059	8.100	6.640	5.779	8.208	9.051
8	3.806	4.795	5.951	6.102	7.363	8.442
9	4.601	5.478	5.380	7.955	8.309	5.455
10	3.802	6.286	5.729	5.278	6.987	7.111

Table 7.17: Averages of Relative Improvement Results for Sets 1 and 2

Set	Relative Improvement (%)					
	Type					
	A	B	C	D	E	F
1	5.02	7.54	6.52	7.35	9.20	8.86
2	4.01	6.06	5.60	6.05	7.80	7.44

As shown in Tables 7.15 and 7.16, the TSH algorithm provides better makespan values than the FFSDSTH algorithm by 2.95-11.85% in the individual test runs. A Factorial Design was used to evaluate the relative improvement (RI) of the solutions obtained by the FFSDTSH algorithm with the application of the TSH algorithm. The design has three factors: deviations in machine speeds, number of products, and number of machines and stages. The analysis was performed using SAS Software V8 for Windows and the results are presented in Appendix C. The statistical results show a significant effect for each of the three factors on the RI. Tukey's test was performed to compare between the three means. Results of the test (see Appendix C, Section C.4) show no difference in the relative improvement (RI) obtained with the (4,2,5) and the (4,4,4,4,4) configurations, and a smaller RI for the (3,3,3) configuration. This can be expected as the quality obtained when applying the FFSDSTH algorithm to problems with larger number of stages and machines (e.g.,(4,4,4,4,4) configuration) or different number of machines per stage (e.g., (4,2,5)) may suffer, thus leaving more room for the TSH to improve the solutions. Results obtained in the ANOVA tables and Tukey's test

show that the relative improvement increases with the increase of the number of machines and stages and the deviations in machine speeds. In contrast, the relative improvement declines as the size of number of products (or families) increases.

CHAPTER 8

CONCLUSIONS AND RECOMMENDATIONS

8.1 Introduction

A comprehensive research was undertaken to minimize the makespan for the "flexible flowshop with sequence dependent setup times" problem. An exact algorithm was first developed and used to solve small problems. Two heuristic algorithms (FFSDSTH and TSH) were then developed to solve larger and more practical problems. In order to evaluate the performance of the heuristic algorithms, two lower bounds were developed for the solution of the problem. In this chapter, a summary of the research performed and the conclusions obtained are presented and followed by its contributions and recommendations for future research.

8.2 Summary of the Research

In Chapter 2, the flexible flowshop with sequence-dependent setup time problem ($FFs(Qm_1,Qm_2,\ldots,Qm_S)/S_{ipm}/ C_{max}$) was introduced in details. The problem investigated in this research consists of one production line with *S* stages. Each stage has one or more non-identical parallel machines (uniform). Machine setup times are required to change over from one product to another. The objective of this research was to minimize the makespan. A review of the relevant literature was presented in Chapter 3 for flexible flowshop scheduling with no setup time consideration, and flowshop scheduling with sequence dependent setup times (SDST). No work was found in the literature for the flexible flowshop scheduling with SDST. A brief review and description of the "Tabu Search" was also given in the same chapter.

In Chapter 4, a 0-1 mixed integer programming model was developed. Since the optimal solution can be obtained for only small size problems, two heuristic algorithms (FFSDSTH and TSH) were developed in Chapter 5. The first algorithm (FFSDSTH) was

developed to obtain a good initial solution. This algorithm starts by assigning families to machines at the first stage, and then proceeds by sequencing the products on the machines. Once all products have been scheduled on the first-stage machines, the algorithm tries to move individual products between machines in an effort to reduce the latest completion time of all products in the first stage. After completing the schedule for the first-stage machines, the assignments of products to machines at the succeeding stages are performed. A Look Ahead (LA) rule was developed to sequence the products on machines at stages 2 through *S*.

The solution obtained from the first phase algorithm (FFSDSTH) is improved in the second phase using the TSH algorithm. The TSH algorithm has 3 main steps: (1) moving families between machines (and within a machine) at the first stage, (2) moving products between machines (and within a machine) at the first stage, and (3) finding a good sequence that results in a low makespan. The processes of moving families and products are not performed for other stages as their computations take large amount of times and they yield very little improvement.

In Chapter 6, two methods were presented for obtaining a lower bound for the flexible flowshop with sequence dependent setup times problems: (1) forward method and (2) backward method. Machine waiting time, idle time, and the total setup and processing times on machines at the last stage were used to obtain the lower bounds.

In Chapter 7, the computational experience obtained with the application of the heuristic procedures was presented. Two data sets with six problem configurations for each set were generated, and ten test problems were generated for each configuration. The performances of the heuristics were presented and evaluated using two measures: (1) solution quality and (2) computational speed. The quality of heuristic solutions was evaluated using lower bounds. The results showed a performance for the FFSDSTH algorithm between 76.9-86.3% for data set 1 and 71.1-83.3% for data set 2. The

performance for the TSH algorithm ranged between 80.7-90.9% for data set 1 and 79.2-86.7% for data set 2. The performance of the algorithms declined with the increase of: (1) deviation in machine speeds (2) number of products, and (3) number of machines and stages.

The computational times were very small for the FFSDSTH algorithm, indicating that this algorithm is very efficient and not sensitive to problem size. Conversely, the computational times of the TSH algorithm increased significantly with problem size--number of products, stages, and machines. For the relative improvement realized when applying the TSH algorithm to the results obtained with the FFSDSTH algorithm, the results indicated an improvement between 2.95 and 11.85%. This improvement increased as the deviations in machine speeds, number of stages, and machines increased. On the other hand, it decreased as the number of products (families) increased.

8.3 Contribution of the Research

According to the literature review, the flexible flowshop with sequence-dependent setup time problem has never been studied. This is true for both cases with identical and uniform processing. The exact algorithm as well as the heuristic algorithm and the lower bound methods developed for the FFSDSTH can also be applied to both identical and uniform parallel processing problems with or without dependent setup times. Computational experience showed that both heuristic algorithms are effective in solving the problem.

8.4 Recommendations for Future Research

The following recommendations are made for future research:

- Additional research may be performed for flexible flowshop with sequence-dependent setup time problems that have several production lines and unrelated machines.
- The calculation of the lower bounds may be further enhanced. In this research, the performance of the lower bound developed declined as deviations in speeds, number of products, number of stages, and number of machines increased. Further research needs to be performed to develop better ways to calculate more accurate lower bounds rather than taking the smallest setup times or the smallest processing times. In this research, the lower bounds were determined by summing two quantities: machine waiting time and total of setup and processing times at the last stage. These lower bounds may be improved by determining these two quantities on every stage rather than just the last stage.
- Improvements may be made to the TSH algorithm. The Tabu search was utilized in this research without using intensification or diversification strategies. These strategies, which are used to guide the search in a more intelligent way, need to be further studied.
- Other search methods (e.g., Neural Network or Genetic Algorithm) may be applied to solve this problem. Their performances may be compared to that of the Tabu Search algorithm.

Bibliography

Allahverdi, A., Gupta, J. N. D., & Aldowaisan, T. (1999). A Review of Scheduling. Research Involving Setup Considerations, *OMEGA, The International Journal of Management Science*, 27: 219-239.

Amin-Narseri, Mohammad Reza (1993). *Pre-Emptive Job Scheduling on a Single Processor with Difference Release Dates to Minimize Total Weighted Flow Time* Doctoral Dissertation, West Virginia University, West Virginia.

Arthanary, T. S. & Ramaswamy, K. G. (1971). An Extension of Two Machine Sequencing Problem. *Opsearch*, 8, 10-22.

Baker, K. R. (1974). *Introduction to Sequencing and Scheduling*., John Wiley & Sons, New York.

Barns, J. W. & Laguna, M. (1993). A Tabu Search Experience in Production Scheduling, *Annals of Operations Research*, 41, 141-156.

Barns, J. W. & Laguna, M. (1993). Solving the Multiple-Machine Weighted Flow Time Problem Using Tabu search. *IIE Transactions*, 25(2), 121-128.

Brah, S. A. & Hunsucker, J. L. (1991). Branch and Bound Algorithm for the Flow Shop with Multiple Processors. *European Journal of Operational Research*, 51, 88-99.

Brandao, J. & Mercer, A. (1997). A Tabu Search Algorithm for the Multi-Trip Vehicle Routing and Scheduling Problem. *European Journal of Operational Research*, 100, 180-191.

Campbell, H. G., Dudek, R. A. & Smith, M. L. (1970). A Heuristic Algorithm for the *n* job *m* Machine Sequencing Problem, *Management Science*, 16, B630-B637.

Chen, C. H. (1997). *Scheduling in a Flowline Manufacturing Cells System Considering Intercellular Pats: The Tabu Search Approach*. Doctoral Dissertation, University of Texas at Arlington, Texas.

Cheng, T. C. E. & Sin, C. C. S. (1990). A State-of-the-Art Review of Parallel-Machine Scheduling Research. *European Journal of Operational Research*, 47, 271-292.

Corwin, B.D. & Esogbue, A.O. (1974). Two Machine Flowshop Scheduling Problems with Sequence Dependent Setup Times: a Dynamic Programming Approach. *Naval Research Logistics Quarterly*, 21(3), 515-524.

Cutright, K. W. (1990). *Scheduling Production Lines with Changeover Costs and Dependent Parallel Processors*. Doctoral Dissertation, West Virginia University, West Virginia.

Day. J. E. & Hottenstien, M. P. (1970). Review of Sequencing Research. *Naval Research Logistics Quarterly*, 17(1),11-39.

De Werra, D. & Hertz, A. (1989). Tabu Search Techniques: A Tutorial and an Application to Neural Networks, *OR Spektrum*. 11, 131-141.

Ding, Fong-Yeun & Kittichartphayak, D. (1994). Heuristics for Scheduling Flexible Flow Lines. *Computers Industrial Engineering,* 26(1), 27-34*.*

Dudek, R, A., Panwalkar, S. S., & Smith, M. L. (1992). The Lessons of Flowshop Scheduling Research, *Operations Research*, 40(1), 7-13.

Egbelu, P. J. (1991). Concurrent Specification of Unit Load Sizes and Autonomated Guided Vehicle Fleet Size in Manufacturing System. *International Journal of Production Economics*, 29, 49-64.

Franca, P. M., Gendreau, M., Laporte, G. & Muller, F. M. (1996). A Tabu Search Heuristic for the Multiprocessor Scheduling Problem with Sequence Dependent Setup Times, *International Journal of Production Economics*, 43, 79-89.

Glover, F. (1989). Tabu Search part I, *ORSA Journal on Computing*, 1(3), 190-206.

Glover, F. (1990). Tabu Search part II, *ORSA Journal on Computing*, 2(1), 4-32.

Goldratt , E. M. (1990). *What's This Thing Called Theory of Constraints*?, North-River Press, Milford, Connecticut.

Guinet, A., Solomon, M.M., Kedia, P.K. & Dussauchoy, A. (1996). A Computational Study of Heuristics for Two-Stage Flexible Flowshops*, International of Production Research*, 34(5), 1399-1415.

Gupta, J. N. D. (1977). Optimal Flowshop Schedules with no Intermediate Storage, *Naval Research Logistics Quarterly*, 12, 235-242.

Gupta, S. K. (1982). N Jobs and m Machines Job-Shop Problems with Sequence Dependent Set-up Times. *International Journal of Production Research*, 20(5), 643-656.

Gupta, J. N. D. (1986). Flowshop Schedules with Sequence Dependent Setup Times. *Journal of the Operational Research Society of Japan*, 29(3), 206-219.

Gupta, J. N. D. (1988). Two-Stage, Hybrid Flowshop Scheduling Problem. *Journal of the Operation Research Society*, 39, 359-364.

Gupta, J. N. D. & Tunc, E. A. (1991). Schedules for a Two-Stage Hybrid Flowshop with Parallel Machine at the Second Stage. *International Journal of Production Research*, 29(7), 1489-1502.

Gupta, J. N. D. & Tunc, E. A. (1994). Scheduling a Two-Stage Hybrid Flowshop with Separable Setup and Removal Time. *European Journal of Operational Research*, 77, 415-428.

Haouari, M. & M'Hallah R. (1997). Heuristic Algorithms for the Two-Stage Hybrid Flowshop Problem. *Operations Research Letters*, 21, 43-53.

Hertz, A. & De Werra, D. (1990). The Search Metaheuristic: How We Used It. *Annals of Mathematics and Artificial Intelligence*, 1, 111-121.

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(2), 194-202.

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

Laguna, M. & Barns, J. W. & Glover, F. (1993). Intelligent Scheduling with Tabu Search: An Application to Jobs with Linear Delay Penalties and Sequence-Dependent Setup Costs and Times. *Journal of Applied Intelligence*, 3, 159-172.

Lee, C. Y. & Vairaktarakis, G. (1994). Minimizing makespan in Hybrid flowshops. *Operations Research Letters*, 16, 149-158.

Lee, Y-. H. & Pinedo, M., (1997). Theory and Methodology: Scheduling Jobs on Parallel Machines with Sequence-Dependent Setup Times, *European Journal of Operational Research*, 100, 464-474.

Li, S. (1997). Theory and Methodology: A Hybrid Two-Stage Flowshop with Part Family, Batch Production, Major and Minor Setup Times, *European Journal of Operation Research*, 102, 142-156.

Moursli, O. (1995). Branch and Bound Lower Bounds for the Hybrid Flowshop. *Intelligent Manufacturing Systems*, 4th IFAC Workshop, 31-36.

Nawaz, M., Emory, E. E. Jr., Ham, I. (1983). A Heuristic Algorithm for the *m*-Machine, *n*-Job Flowshop Sequencing Problem, *OMEGA, The International Journal of Management Science*, 11(1), 91-95.

Norman, B. A. (1999). Scheduling Flowshops with Finite Buffers and Sequence-Dependent Setup Times. *Computers & Industrial Engineering*, 36, 163-177.

Nowicki, E. and Smutnicki, C. (1996). A Fast Tabu Search Algorithm for the Permutation Flow-shop Problem. *European Journal of Operational Research*, 91, 160-175.

Nowicki, E. & Smutnicki, C. (1998). The Flowshop with Parallel Machines: A Tabu Search Approach, *European Journal of Operational Research*, 106, 226-253.

Osman, I. H.& Potts, C. N. (1989). Simulated Annealing for Permutation Flowshop Scheduling, *OMEGA, The International Journal of Management Science*, 17(6): 551-557.

Pinedo, M., (1995). Theory, Algorithm, and Systems., Prentice-Hall, Englewood Cliffs, New Jersey.

Portmann, M-. C., Vignier, A., Dardilhac, D. & Dezalay, D. (1998). Branch and Bound Crossed with GA to Solve Hybrid Flowshops, *European Journal of Operational Research*, 107, 384-400.

Riane, F., & Artiba, A. (1997). A Hybrid Heuristic for Scheduling a Hybrid Flowshop Maximum Completion Time Problem. International Conference on Neural Network Information Processing and Intelligence Information Systems, New Zealand, 1021-1024.

Riane, F., Artiba, A. & Elmaghraby, S. E. (1998). A Hybrid Three-Stage Flowshop Problem: Efficient Heuristics to Minimize Makespan, *European Journal of Operational Research*, 109, 321-329.

Rajendran, C. & Chaudhuri, D. (1992). Scheduling in *n*-job, *m*-machine Flowshop with Parallel Processors to Minimize makespan, *International Journal of Production Economics*, 27, 137-143.

Randhawa, S. U. & Smith, T.A. (1995). An Experiment Investigation of Scheduling Non-Identical, Parallel Processors with Sequence-Dependent Setup Times and Due Date, *International Journal of Production Research*, 33(1), 59-69.

Randhawa, S. U. & Kuo, C. H. (1997). Evaluation Scheduling Heuristic for Non-Identical Parallel Processors, *International Journal of Production Research*, 35(4), 969-981.

Rios-Mercado, R. Z. (1997*). Optimization of the Flowshop Scheduling Problem with Setup Times*. Doctoral Dissertation, University of Texas at Austin, Texas.

Rios-Mercado, R. Z. & Bard, J.F. (1998). Heuristic for the Flowline Problem with Setup Costs, *European Journal of Operational Research*, 110(1), 76-98.

Rios-Mercado, R. Z. & Bard, J.F. (1999). A Branch-and-Bound Algorithm for Permutation Flow Shops with Sequence-Dependent Setup Times*. IIE Transactions*, 31, 721-731.

Rios-Mercado, R. Z. & Bard, J.F. (1999). An Enhanced TSP-Based Heuristic for Makespan Minimization in a Flow Shop with Setup Times*. Journal of Heuristics*, 5, 53-70.

Sarin, S. & Lefoka, M. (1993). Scheduling Heuristics for the n-Job m-Machine Flow Shop. *OMEGA, The International Journal of Management Science*, 21(2): 229-234.

Simons Jr.,J. V., (1992). Heuristics in Flowshop Scheduling with Sequence Dependent Setup Times. *OMEGA, The International Journal of Management Science*, 20(2): 215-225.

Skorin-Kapov, J. & Vakharia, A., J. (1993). Scheduling a Flow-Line Manufacturing Cell: A Tabu Search Approach. *International Journal of Production Research*, 31(7), 1721-1734.

Soewadi, H. (1998). *Sequencing Jobs on Two-and Three-Stage Hybrid Flowshop to Minimize Makespan*, Doctoral Dissertation, North Carolina State University, North Carolina.

Srikar, B. N. & Ghosh, S. (1986). A MILP Model for the *n*-job, *m*-stage Flowshop with Sequence Dependent Set-up Times. *International Journal of Production Research*, 24(6), 1459-1474.

Stafford, E. F. & Tseng, F. T. (1990). On Srikar-Ghosh MILP Model for the *NxM* SDST Flowshop Problem. *International Journal of Production Research*, 28(10), 1817-1830.

Szwarc, W. & Gupta, J. N. D. (1987). A Flow-Shop with Sequence-Dependent Additive Setup Times. *Naval Research Logistics Quarterly*, 34(5), 619-627.

Tillard, E. (1990). Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem. *European Journal of Operational Research*, 47(1), 65-74.

Umbel, M. M. & Srikanth, M. L. (1992). *Synchronous Manufacturing: Principle for World Class Excellence*, South-western publishing Co, Cincinnati, OH.

Verma, S. & Dessouky, M. (1999). Multistage Hybrid Flowshop Scheduling with Identical Jobs and Uniform Parallel Machines. *Journal of Scheduling*., 2, 135-150.

Vignier A., Dardilhac, D., Dezalay, D., & Proust C. (1996). A Branch and Bound Approach to Minimize the Total Completion Time in a k-Stage Hybrid Flowshop, IEEE Conference on Engineering Technologies and Factory Automation, November, Kauai, Hawaii, 18-21.

Vollman, T. E., Berry, W. L. & Whybark, D. C. (1992). Manufacturing Planning and Control Systems., Richard D. Irwin, INC, Boston, Massachusetts.

Wilbrecht, J. K. & Prescott, W. B. (1969). The Influence of Setup Time on Job Shop Performance, *Management Science*, 16, B274-B280.

Wortman, D. B. (1992). Managing Capacity: Getting the most From Your Company Assets, *Industrial Engineering*, 24, 47-49.

APPENDICIES

APPENDIX A

Sequencing Notation Used in This Research

Normally, a notation of scheduling problems has the form which consists of three parameters, $\alpha/\beta/\gamma$. The first parameter (α) describes a machine environment and contains a single entry. The second parameter (β) is a field providing the details of processing characteristics and constraints. The β field may contain no entry, a single entry, or multiple entries. The last parameter (γ) contains the objective to be minimized and usually contains a single entry. Additionally, the number of jobs and machines are denoted by *n* and *m*, respectively. Both *m* and *n* are assumed to be finite. In this research, subscripts *i* and *p* refer to jobs, whereas subscript *k* refer to machines.

There are two sections presented in this appendix. The first section describes data associated with jobs, and the second section presents descriptions of possible entries of the fields in the triple form ($\alpha/\beta/\gamma$) that are used in this research. The notation described in this appendix is adapted from Pinedo (1995).

A.1 Fundamental Data Associated with Jobs

The following pieces of data are associated with job *i*.

- *Processing time* (*t(i,k)*). The *t(i,k)* represents the processing time of *j*ob *i* on machine *k*. The subscript *i* is dropped if the processing time of job *i* does not depend on the machine or if job *i* is only to be processed on one given machine. In this research, both products and families are considered. Products are grouped within a family. The *t(j,i,k,s)*denotes the processing time of product *i* of family *j* on machine *k* on stage *s*.

- *Due date (d_i).* The due date d_i of job *i* represents the committed shipping or completion date (the date of the job that is promised to the customers).
- *Weight (w_i).* The weight w_i of job *i* is a priority factor denoting the importance of job *i* relative to the other jobs in the system.

A.2 Problem Description

In this section, the possible entries for each of the fields in a triplet $\alpha/\beta/\gamma$ of a scheduling problem are presented.

Field α. This field describes the machine environment and contains a single entry. The following examples are possible machine environments contained in the α field.

- *Flowshop (Fm)*. There are m machines in series. Each job has to be processed on each one of the machines. All jobs have the same routing; that is, they have to be processed first on machine 1, then on machine 2, and so on and so forth. After completion on one machine, a job joins the queue at the next machine. Normally, all queues are assumed to operate under the first-in-first-out (FIFO) discipline; that is, a job cannot "pass" another while waiting in a queue. If the FIFO discipline is in effect, the flowshop is referred to as a permutation flowshop, and the β field includes the entry prmu. Often, when a general *m*-machine case is considered, the m identifier may be dropped such that $F//C_{max}$, for instance, refers to the *m*-machine flowshop with the objective of minimizing makespan.
- *Flexible flowshop (FFs)*. A flexible flowshop is a generalization of the flowshop and the parallel machine environments. A flexible flowshop consists of *S* production stages in series with a number of machines in parallel at each stage. Each job is

processed first at stage 1, then at stage 2, and so on. Normally, job *i* requires only 1 machine at each stage and any machine can process any job.

- *Identical machines in parallel (Pm).* There are *m* identical machines in parallel. Job *i* requires a single operation and may be processed on any one of the *m* machines or on any one belonging to a given subset. If job *i* is not allowed to be processed on just any one, but rather only on any one belonging to a given subset, that is, M_i, then the entry M_i appears in the β field. In this environment, if the unit processing time of job *i* on machine *k* is denoted by *t(i,k)*, then *t(1,k)= t(2,k)= … = t(i,k) = t(i,m)* for *i* = 1,2,…,*n.*
- *Machines in parallel with different speeds (Qm).* There are *m* machines in parallel with different speeds. The speed of machine *k* is denoted by v_k. If job *i* is assumed to process only on machine *k*, the time *t(i)* job *i* spends on machine *k* is equal to $t(i)/v_k$. This environment is also called uniform machines. If all machines have the same speed, that means $v_k = 1$ for all *k* and *t(i,k) = t(k)*, then this environment is identical to the identical machines in parallel (Pm).
- *Unrelated machines in parallel (Rm).* This environment is a generalization of the machines in parallel with different speed (Qm) environment. There are m different machines in parallel. Machine *k* can process job *i* at speed v_{ki}. The time *t(i,k)* job *i* spends on machine *k* is equal to $t(i)_i/v_{ki}$. If the speeds of the machines are independent of the jobs, that means $v_{ki} = v_k$ for all *i* and *k*, then the environment is identical to the machines in parallel with different speed (Qm) environment.

Field β. This field provides details of processing characteristics and constraints and may contain no entries, a single entry, or multiple entries. Possible entries are described as follows:

- *Sequence dependent setup times* (*s(i,p)*). The *s(i,p)* represent the setup time between jobs *i* and *p*. *s(i,p)* denotes the setup time for job *p* if job *p* is first in the sequence and *s(i,0)* denotes the clean-up time after job *i* if job *i* is the last in the sequence. However, *s(0,p)* and *s(i,0)* may be zero. If the setup time between job *i* and *p* depends on the machine, then the subscript *m* is included, that is, *s(i,p,m)*. If no *s(i,p)* appears in the β field, all setup times are assumed to be zero or sequence independent, in which case they can simply be added to the processing times.

  In this research, both end products and families are considered. This means there are many end products within each family and both major and minor setup times are considered. If the previous product belongs to the same family, setup time is minor. On the other hand, if the product is of a different family, a major setup time is needed. The *s(j,i,j,p)* denotes the minor setup time between product *i* and product *p* from the same family *j*. The *s(j,i,q,p)* denotes the major setup time between product *i* family *j* and product *p* family *q.* If the setup time between two products depends on the machine of any stage *s*, then the subscripts *m* and *s* are included. For instance, *s(j,i,q,p,s,m)* denotes the major setup time between product *i* family *j* and product *p* family *q* on machine *m* stage *s.*

- *Permutation (prmu*). A constraint that may appear in the flowshop is that the queues in form of each machine operate according to the FIFO discipline. This means that the order (or permutation) in which the jobs go through the first machine is maintained throughout the system.

- *No-wait (nwt*). The no-wait requirement is another phenomenon which may occur in flowshops. Jobs are not allowed to wait between two successive machines. This means that the starting time of a job at the first machine has to be delayed to ensure that the job can go through the flowshop without having to wait for any machine. An example of such an operation is a steel-rolling mill in which a slab of steel is not

allowed to wait because it would cool off. In other words, under no-wait the machines also operate under the FIFO discipline.

Field γ. This field contains the objective to be minimized and usually contains a single entry. In order to minimize the objective, it is always a function of the completion times of the jobs which depend on the schedule. The completion time of job *i* on machine *m* is represented by C_{im}. The time of job *i* exits the system (i.e. its completion time on the last machine on which it requires processing) is denoted by C_i. The objective may also be a function of the due dates. The lateness of job *i* is defined as

$$L_i = C_i - d_i \qquad \text{(A.1)}$$

Which is positive when job *i* is completed late and negative when it is completed early. The tardiness of job *i* is defined as

$$T_i = \max(C_i - d_i, 0) = \max(L_i, 0). \qquad \text{(A.2)}$$

The difference between tardiness and lateness lies in the fact that tardiness is never negative. The unit penalty of job *i* is defined as

$$U_j = \begin{cases} 1, \text{If} C_i > d_i \\ 0, \text{otherwise} \end{cases} \qquad \text{(A.3)}$$

- *Makespan* (C_{max}). The makespan, defined as $\max_i \{C_i\}$: i=1,2,3,..,n, is equivalent to the completion time of the last job to leave the system. A minimum makespan usually indicates a high utilization of the machine(s).
- *Total weighted completion time ($\sum w_i C_i$).* The minimization of $\sum w_i C_i$ is equivalent to the minimization of the in-process inventory cost for the shop.

- *Total weighted tardiness ($\sum w_i T_i$)*. The total weighted tardiness may be used as a measure for meeting due dates.

APPENDIX B

Listing of the 0-1 Mixed Integer Programming Model for the Problem Illustrated in Chapter 4

```
MINIMIZE
  Z:  E
SUBJECT TO
  FT1111 - 16.61818181818 x1111  >=  3.89
  FT1211 - 30.73636363636 x1211  >=  1.52
  FT2111 - 44.5 x2111  >=  2.26
  FT2211 - 22.81818181818 x2211  >=  3.99
  - FT1111 + FT1121 - 27.45217391304 x1121  >=  0
  - FT1121 + FT1131 - 23.68 x1131  >=  0
  - FT1121 + FT1132 - 24.16326530612 x1132  >=  0
  - FT1211 + FT1221 - 24.55652173913 x1221  >=  0
  - FT1221 + FT1231 - 44.87 x1231  >=  0
  - FT1221 + FT1232 - 45.78571428571 x1232  >=  0
  - FT2111 + FT2121 - 22.68695652173 x2121  >=  0
  - FT2121 + FT2131 - 19.09 x2131  >=  0
  - FT2121 + FT2132 - 19.47959183673 x2132  >=  0
  - FT2211 + FT2221 - 15.12173913043 x2221  >=  0
  - FT2221 + FT2231 - 49.26 x2231  >=  0
  - FT2221 + FT2232 - 50.26530612244 x2232  >=  0
  - E + FT1131  <=  0
  - E + FT1132  <=  0
  - E + FT1231  <=  0
  - E + FT1232  <=  0
  - E + FT2131  <=  0
  - E + FT2132  <=  0
  - E + FT2231  <=  0
  - E + FT2232  <=  0
  FT1111 - FT1211 - 5000 w121111 - 16.61818181818 x1111      >=  - 4995.76
  FT1121 - FT1221 - 5000 w121121 - 27.45217391304 x1121      >=  - 4997.04
  FT1131 - FT1231 - 5000 w121131 - 23.68 x1131  >=      - 4996.78
  FT1132 - FT1232 - 5000 w121132 - 24.16326530612 x1132      >=  - 4996.78
  FT1111 - FT2111 - 5000 w211111 - 16.61818181818 x1111      >=  - 4994.12
  FT1121 - FT2121 - 5000 w211121 - 27.45217391304 x1121      >=  - 4993.55
  FT1131 - FT2131 - 5000 w211131 - 23.68 x1131  >=      - 4993.36
  FT1132 - FT2132 - 5000 w211132 - 24.16326530612 x1132      >=  - 4993.36
  FT1111 - FT2211 - 5000 w221111 - 16.61818181818 x1111      >=  - 4995.1
  FT1121 - FT2221 - 5000 w221121 - 27.45217391304 x1121      >=  - 4989.71
  FT1131 - FT2231 - 5000 w221131 - 23.68 x1131  >=      - 4989.37
  FT1132 - FT2232 - 5000 w221132 - 24.16326530612 x1132      >=  - 4989.37
  - FT1111 + FT1211 - 5000 w111211 - 30.73636363636 x1211      >=  - 4997.51
  - FT1121 + FT1221 - 5000 w111221 - 24.55652173913 x1221      >=  - 4997.8
  - FT1131 + FT1231 - 5000 w111231 - 44.87 x1231  >=      - 4996.87
  - FT1132 + FT1232 - 5000 w111232 - 45.78571428571 x1232      >=  - 4996.87
  FT1211 - FT2111 - 5000 w211211 - 30.73636363636 x1211      >=  - 4993.46
  FT1221 - FT2121 - 5000 w211221 - 24.55652173913 x1221      >=  - 4988.75
  FT1231 - FT2131 - 5000 w211231 - 44.87 x1231  >=      - 4989.83
  FT1232 - FT2132 - 5000 w211232 - 45.78571428571 x1232      >=  - 4989.83
  FT1211 - FT2211 - 5000 w221211 - 30.73636363636 x1211      >=  - 4995.48
  FT1221 - FT2221 - 5000 w221221 - 24.55652173913 x1221      >=  - 4992.13
  FT1231 - FT2231 - 5000 w221231 - 44.87 x1231  >=      - 4993.85
  FT1232 - FT2232 - 5000 w221232 - 45.78571428571 x1232      >=  - 4993.85
  - FT1111 + FT2111 - 5000 w112111 - 44.5 x2111  >=      - 4994.67
  - FT1121 + FT2121 - 5000 w112121 - 22.68695652173 x2121      >=  - 4990.61
  - FT1131 + FT2131 - 5000 w112131 - 19.09 x2131  >=      - 4990.21
  - FT1132 + FT2132 - 5000 w112132 - 19.47959183673 x2132      >=  - 4990.21
  - FT1211 + FT2111 - 5000 w122111 - 44.5 x2111  >=      - 4993.78
  - FT1221 + FT2121 - 5000 w122121 - 22.68695652173 x2121      >=  - 4989.76
  - FT1231 + FT2131 - 5000 w122131 - 19.09 x2131  >=      - 4990.05
  - FT1232 + FT2132 - 5000 w122132 - 19.47959183673 x2132      >=  - 4990.05
  FT2111 - FT2211 - 5000 w222111 - 44.5 x2111  >=  - 4995.8
  FT2121 - FT2221 - 5000 w222121 - 22.68695652173 x2121      >=  - 4997.59
```

```
  FT2131 - FT2231 - 5000 w222131 - 19.09 x2131  >=      - 4996.54
  FT2132 - FT2232 - 5000 w222132 - 19.47959183673 x2132      >=  - 4996.54
  - FT1111 + FT2211 - 5000 w112211 - 22.81818181818 x2211      >=  - 4994.09
  - FT1121 + FT2221 - 5000 w112221 - 15.12173913043 x2221      >=  - 4989.42
  - FT1131 + FT2231 - 5000 w112231 - 49.26 x2231  >=      - 4993.37
  - FT1132 + FT2232 - 5000 w112232 - 50.26530612244 x2232      >=  - 4993.37
  - FT1211 + FT2211 - 5000 w122211 - 22.81818181818 x2211      >=  - 4994.49
  - FT1221 + FT2221 - 5000 w122221 - 15.12173913043 x2221      >=  - 4993.02
  - FT1231 + FT2231 - 5000 w122231 - 49.26 x2231  >=      - 4988.5
  - FT1232 + FT2232 - 5000 w122232 - 50.26530612244 x2232      >=  - 4988.5
  - FT2111 + FT2211 - 5000 w212211 - 22.81818181818 x2211      >=  - 4996.52
  - FT2121 + FT2221 - 5000 w212221 - 15.12173913043 x2221      >=  - 4996.93
  - FT2131 + FT2231 - 5000 w212231 - 49.26 x2231  >=      - 4998.35
  - FT2132 + FT2232 - 5000 w212232 - 50.26530612244 x2232      >=  - 4998.35
  x1111  =  1
  x1121  =  1
  x1131 + x1132  =  1
  x1211  =  1
  x1221  =  1
  x1231 + x1232  =  1
  x2111  =  1
  x2121  =  1
  x2131 + x2132  =  1
  x2211  =  1
  x2221  =  1
  x2231 + x2232  =  1
  - w001111 - w121111 - w211111 - w221111     + x1111  =  0
  - w001121 - w121121 - w211121 - w221121     + x1121  =  0
  - w001131 - w121131 - w211131 - w221131     + x1131  =  0
  - w001132 - w121132 - w211132 - w221132     + x1132  =  0
  - w001211 - w111211 - w211211 - w221211     + x1211  =  0
  - w001221 - w111221 - w211221 - w221221     + x1221  =  0
  - w001231 - w111231 - w211231 - w221231     + x1231  =  0
  - w001232 - w111232 - w211232 - w221232     + x1232  =  0
  - w002111 - w112111 - w122111 - w222111     + x2111  =  0
  - w002121 - w112121 - w122121 - w222121     + x2121  =  0
  - w002131 - w112131 - w122131 - w222131     + x2131  =  0
  - w002132 - w112132 - w122132 - w222132     + x2132  =  0
  - w002211 - w112211 - w122211 - w212211     + x2211  =  0
  - w002221 - w112221 - w122221 - w212221     + x2221  =  0
  - w002231 - w112231 - w122231 - w212231     + x2231  =  0
  - w002232 - w112232 - w122232 - w212232     + x2232  =  0
  - w110011 - w111211 - w112111 - w112211     + x1111  =  0
  - w110021 - w111221 - w112121 - w112221     + x1121  =  0
  - w110031 - w111231 - w112131 - w112231     + x1131  =  0
  - w110032 - w111232 - w112132 - w112232     + x1132  =  0
  - w120011 - w121111 - w122111 - w122211     + x1211  =  0
  - w120021 - w121121 - w122121 - w122221     + x1221  =  0
  - w120031 - w121131 - w122131 - w122231     + x1231  =  0
  - w120032 - w121132 - w122132 - w122232     + x1232  =  0
  - w210011 - w211111 - w211211 - w212211     + x2111  =  0
  - w210021 - w211121 - w211221 - w212221     + x2121  =  0
  - w210031 - w211131 - w211231 - w212231     + x2131  =  0
  - w210032 - w211132 - w211232 - w212232     + x2132  =  0
  - w220011 - w221111 - w221211 - w222111     + x2211  =  0
  - w220021 - w221121 - w221221 - w222121     + x2221  =  0
  - w220031 - w221131 - w221231 - w222131     + x2231  =  0
  - w220032 - w221132 - w221232 - w222132     + x2232  =  0
  w001111 + w001211 + w002111 + w002211  =  1
  w001121 + w001221 + w002121 + w002221  =  1
  w001131 + w001231 + w002131 + w002231  =  1
  w001132 + w001232 + w002132 + w002232  =  1
  w110011 + w120011 + w210011 + w220011  =  1
  w110021 + w120021 + w210021 + w220021  =  1
  w110031 + w120031 + w210031 + w220031  =  1
  w110032 + w120032 + w210032 + w220032  =  1

INTEGERS
    w001111
    w001121
    w001131
```

w001132
w001211
w001221
w001231
w001232
w002111
w002121
w002131
w002132
w002211
w002221
w002231
w002232
w110011
w110021
w110031
w110032
w120011
w120021
w120031
w120032
w210011
w210021
w210031
w210032
w220011
w220021
w220031
w220032
w111211
w111221
w111231
w111232
w112111
w112121
w112131
w112132
w112211
w112221
w112231
w112232
w121111
w121121
w121131
w121132
w122111
w122121
w122131
w122132
w122211
w122221
w122231
w122232
w211111
w211121
w211131
w211132
w211211
w211221
w211231
w211232
w212211
w212221
w212231
w212232
w221111
w221121
w221131
w221132
w221211
w221221

w221231
w221232
w222111
w222121
w222131
w222132
x1111
x1121
x1131
x1132
x1211
x1221
x1231
x1232
x2111
x2121
x2131
x2132
x2211
x2221
x2231
x2232

APPENDIX C

Statistical Results for the Evaluations of the Heuristic Performance and the Relative Improvement

C.1 Statistical Results for the Evaluation of the Heuristic Performance (HP)

Dependent Variable: HP_{TSH}

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	11	1858.360009	168.941819	19.14	<.0001
Error	108	953.486950	8.828583		
Corrected Total	119	2811.846959			

R-Square	Coeff Var	Root MSE	TSH Mean
0.660904	3.530798	2.971293	84.15358

Source	DF	Type I SS	Mean Square	F Value	Pr > F
prod	1	550.1085408	550.1085408	62.31	<.0001
mach	2	715.8530017	357.9265008	40.54	<.0001
prod*mach	2	0.2084017	0.1042008	0.01	0.9883
speed	1	561.2985075	561.2985075	63.58	<.0001
prod*speed	1	0.9451875	0.9451875	0.11	0.7441
mach*speed	2	12.7282850	6.3641425	0.72	0.4887
prod*mach*speed	2	17.2180850	8.6090425	0.98	0.3804

Dependent Variable: $HP_{FFSDSTH}$

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	11	2326.077143	211.461558	25.93	<.0001
Error	108	880.626070	8.153945		
Corrected Total	119	3206.703212			

R-Square	Coeff Var	Root MSE	FFS Mean
0.725380	3.639275	2.855511	78.46375

Source	DF	Type I SS	Mean Square	F Value	Pr > F
prod	1	258.867188	258.867188	31.75	<.0001
mach	2	1065.433460	532.716730	65.33	<.0001
prod*mach	2	0.413180	0.206590	0.03	0.9750
speed	1	980.579841	980.579841	120.26	<.0001
prod*speed	1	0.088021	0.088021	0.01	0.9174
mach*speed	2	10.255287	5.127643	0.63	0.5351
prod*mach*speed	2	10.440167	5.220083	0.64	0.5292

C.2 Statistical Results for the Evaluation of the Relative Improvement (RI)

Dependent Variable: RI

Source	DF	Sum of Squares	Mean Square	F Value	Pr > F
Model	11	258.3741492	23.4885590	12.11	<.0001
Error	108	209.5162300	1.9399651		
Corrected Total	119	467.8903792			

R-Square	Coeff Var	Root MSE	IMP Mean
0.552211	20.51920	1.392826	6.787917

Source	DF	Type I SS	Mean Square	F Value	Pr > F
prod	1	47.0877408	47.0877408	24.27	<.0001
mach	2	89.6388867	44.8194433	23.10	<.0001
prod*mach	2	0.4924867	0.2462433	0.13	0.8809
speed	1	118.9821675	118.9821675	61.33	<.0001
prod*speed	1	0.3933075	0.3933075	0.20	0.6534
mach*speed	2	1.3467800	0.6733900	0.35	0.7075
prod*mach*speed	2	0.4327800	0.2163900	0.11	0.8946

C.3 Results of Tukey's test for Comparing the Means for the Heuristic Performance

Since the number of products and deviations in machine speeds have only 2 levels, the comparison of their means for the heuristic performance (HP) can be interpreted using the ANOVA tables in Section C.1 and the summary of the averages of the heuristic performance in Table 7.14. Hence, Tukey's test was performed only to

compare between the three means obtained with different number of machines and stages (mach).

Tukey's Studentized Range (HSD) Test for HP_{TSH}

Alpha	0.05
Error Degrees of Freedom	108
Error Mean Square	8.828583
Critical Value of Studentized Range	3.36085
Minimum Significant Difference	1.5789

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	mach
A	87.0475	40	1
B	84.3397	40	3
C	81.0735	40	2

Tukey's Studentized Range (HSD) Test for $HP_{FFSDSTH}$

Alpha	0.05
Error Degrees of Freedom	108
Error Mean Square	8.153945
Critical Value of Studentized Range	3.36085
Minimum Significant Difference	1.5174

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	mach
A	82.1642	40	1
B	78.3593	40	3
C	74.8678	40	2

C.4 Results of the Tukey's Test for Comparing the Means for the Relative Improvement (RI)

As in Section C.3, Tukey's test was used only to compare between the means obtained with different number of machines and stages (mach).

Tukey's Studentized Range (HSD) Test for RI

Alpha	0.05
Error Degrees of Freedom	108
Error Mean Square	1.939965
Critical Value of Studentized Range	3.36085
Minimum Significant Difference	0.7401

Means with the same letter are not significantly different.

Tukey Grouping	Mean	N	mach
A	7.6507	40	2
A			
A	7.1063	40	3
B	5.6068	40	1