การจัดตารางการผลิตแบบไหลเลื่อนยืดหยุ่นชนิดเครื่องจักรขนานที่ไม่สัมพันธ์กันและเวลาปรับตั้งเครื่องขึ้นกับลำดับงานก่อนหน้า

นายจิตติ จึงวัฒนกิจ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต
สาขาวิชาวิศวกรรมอุตสาหการ ภาควิชาวิศวกรรมอุตสาหการ
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2550

A SCHEDULING IN FLEXIBLE FLOW SHOP PROBLEM WITH UNRELATED PARALLEL MACHINES AND SEQUENCE-DEPENDENT SETUP TIMES

Mr. Jitti Jungwattanakit

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy Program in Industrial Engineering
Department of Industrial Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2007

Thesis Title	A SCHEDULING IN FLEXIBLE FLOW SHOP PROBLEM WITH UNRELATED PARALLEL MACHINES AND SEQUENCE-DEPENDENT SETUP TIMES
By	Mr. Jitti Jungwattanakit
Field of Study	Industrial Engineering
Thesis Advisor	Assistant Professor Manop Reodecha, Ph.D.
Thesis Co-advisor	Assistant Professor Paveena Chaovalitwongse, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctoral Degree

[illegible] ....Dean of the Faculty of Engineering
(Associate Professor Boonsom Lerdhirunwong, Dr.Ing.)

THESIS COMMITTEE

[illegible] ....Chairman
(Professor Sirichan Thongprasert, Ph.D.)

[illegible] ....Thesis Advisor
(Assistant Professor Manop Reodecha, Ph.D.)

[illegible] ....Thesis Co-advisor
(Assistant Professor Paveena Chaovalitwongse, Ph.D.)

[illegible] ....External Member
(Associate Professor Singha Chiamsiri, Ph.D.)

[illegible] ....Member
(Assistant Professor Rein Boondiskulchok, D.Eng.)

จิตติ จึงวัฒนกิจ : การจัดตารางการผลิตแบบไหลเลื่อนยืดหยุ่นชนิดเครื่องจักรขนานที่ไม่สัมพันธ์กันและเวลาปรับตั้งเครื่องขึ้นกับลำดับงานก่อนหน้า. (A SCHEDULING IN FLEXIBLE FLOW SHOP PROBLEM WITH UNRELATED PARALLEL MACHINES AND SEQUENCE-DEPENDENT SETUP TIMES). อ.ที่ปรึกษา : ผศ.ดร.มานพ เรี่ยวเดชะ, อ.ที่ปรึกษาร่วม : ผศ.ดร.ปวีณา เชาวลิตวงศ์ 199 หน้า.

วิทยานิพนธ์ฉบับนี้ศึกษาปัญหาการจัดตารางการผลิตแบบไหลเลื่อนยืดหยุ่น (flexible flow shop) โดยมีขั้นตอนการผลิตอย่างน้อยหนึ่งขั้นตอนที่ใช้เครื่องจักรขนานกันที่ไม่สัมพันธ์กัน เวลาที่ใช้เพื่อการปรับตั้งเครื่องจักรจะขึ้นกับลำดับงานก่อนหน้าและเครื่องจักร โดยมีวัตถุประสงค์เพื่อจัดตารางการผลิตที่ทำให้เวลาปิดงานของระบบและจำนวนงานล่าช้ามีค่าน้อยที่สุด ภายใต้ภาวะของการจัดตารางการผลิตแบบสถิต (static) งานวิจัยนี้ได้สร้างโปรแกรมเชิงเส้นจำนวนเต็มแบบผสมและแบบทวิภาคสำหรับการแก้ปัญหา แต่ด้วยปัญหานี้ถือเป็นปัญหาการหาคำตอบที่ดีที่สุดเชิงการจัด (combinatorial optimization problem) ที่ยากเกินไปที่จะใช้แก้ปัญหาที่มีขนาดใหญ่ได้ ดังนั้นจึงได้พัฒนาฮิวริสติก เพื่อหาคำตอบที่ดีโดยใช้เวลาที่สมเหตุสมผล โดยฮิวริสติกนี้มีทั้งหมด 3 ขั้นในการหาคำตอบ ขั้นที่หนึ่งใช้วิธีสร้างเสริม (constructive algorithm) โดยเริ่มต้นจากการสร้างตัวแทนเวลาของแต่ละกระบวนการ แล้วนำแนวคิดของกฎการจ่ายงาน (dispatching rules) หรือฮิวริสติกสำหรับระบบการผลิตแบบไหลเลื่อนสามัญ (simple flow shop heuristics) มาใช้เพื่อสร้างคำตอบเบื้องต้น ขั้นที่สองใช้วิธีการเลื่อนงานหรือการสับเปลี่ยนงานมาปรับปรุงคำตอบ จากนั้นเป็นการนำวิธีเมต้าฮิวริสติกมาใช้เพื่อปรับปรุงคำตอบอีกครั้งในขั้นสุดท้าย ได้มีการทดสอบสมรรถนะของฮิวริสติกที่ใช้ในแต่ละขั้น ด้วยการเปรียบเทียบกับคำตอบที่ดีที่สุดจากกลุ่มปัญหาทดสอบที่มีจำนวนงานมากสุด 50 งาน และ ขั้นตอนมากสุด 20 ขั้นตอน ผลการทดลองชี้ให้เห็นว่าวิธีของ Nawaz Enscore และ Ham ที่เรียกว่าวิธี NEH เป็นวิธีที่เหมาะสมที่สุดในการหาคำตอบเริ่มต้น และวิธีการสับเปลี่ยนงานแบบ all-pairwise-interchanges เป็นวิธีการที่ดีสำหรับปรับปรุงคำตอบ และวิธี simulated annealing เป็นวิธีเมต้าฮิวริสติกที่ดีที่สุดสำหรับการปรับปรุงคำตอบให้ดีขึ้นอีก

ภาควิชา.....วิศวกรรมอุตสาหการ.....	ลายมือชื่อนิสิต.....จิตติ จึงวัฒนกิจ.....
สาขาวิชา.....วิศวกรรมอุตสาหการ.....	ลายมือชื่ออาจารย์ที่ปรึกษา.....[illegible].....
ปีการศึกษา.....2550.....	ลายมือชื่ออาจารย์ที่ปรึกษา.....[illegible].....

467 18069 21: MAJOR INDUSTRIAL ENGINEERING
KEYWORD; FLEXIBLE FLOW SHOP / CONSTRUCTIVE ALGORITHMS / METAHEURISTICS.

JITTI JUNGWATTANAKIT : A SCHEDULING IN FLEXIBLE FLOW SHOP PROBLEM WITH UNRELATED PARALLEL MACHINES AND SEQUENCE-DEPENDENT SETUP TIMES. THESIS ADVISOR : ASST. PROF. MANOP REODECHA, Ph.D., THESIS COADVISOR : ASST. PROF. PAVEENA CHAOVALITWONGSE, Ph.D., 199 pp.

This dissertation studies a scheduling problem of flexible flow shop, where at least one production stage is made up of unrelated parallel machines, and setup times are sequence- and machine-dependent. The objective is to find a schedule that minimizes the makespan and the number of tardy jobs in a static scheduling environment. For this problem, a 0-1 mixed integer programming is formulated. The model is, however, a combinatorial optimization problem which is too difficult to be solved for large-sized problems, and hence, a heuristic is developed to obtain good solutions in reasonable time. The heuristic has three phases. The first phase uses a constructive algorithm. It starts with the generation of the operating time representative for each operation. Then, it uses a dispatching rule or a simple flow shop makespan heuristic to determine an initial solution. The improvement algorithm based on shift moves or pairwise interchanges of jobs is applied to improve the solution in the second phase. After that, a metaheuristic is used to refine the solution in the final phase. Several well-known heuristics are tested in each phase. The performances of the heuristics are compared to one another based on a set of test problems with up to 50 jobs and 20 stages and with an optimal solution for small-sized problems. The computational results indicate that the Nawaz, Enscore, and Ham (NEH) algorithm is most suitable for determining the initial solution, the all-pairwise-interchange approach is good for improving the solution, and the simulated annealing algorithm is best metaheuristic for refining the solution.

Department……Industrial Engineering	Student's signature……จิตติ จึงวัฒนกิจ.
Field of study……Industrial Engineering	Advisor's signature……[illegible]
Academic year……2007	Co-advisor's signature……[illegible]

ACKNOWLEDGEMENTS

I am deeply indebted to my advisors, Assistant Professor Manop Reodecha, Ph.D., and Assistant Professor Paveena Chaovalitwongse, Ph.D., for their useful advice, constant guidance, continuous encouragement, and endless patience throughout my Ph.D. study. I wish to express my first sincere thanks for their strong commitment to assisting me achieve my research goals. Special thanks are also extended to Professor Frank Werner, Ph.D. from Faculty of Mathematics, Otto-von-Guericke-University, Germany, for his great encouragement, continuous suggestions, and unconditional helps while I was working on my research. This dissertation would not have been possible without them.

I am also grateful for the useful suggestions, valuable comments, and positive criticisms that Professor Sirichan Thongprasert, Ph.D., Assistant Professor Rein Boondiskulchok, D.Eng., and Associate Professor Singha Chiamsiri, Ph.D. as members of the examination committee provided me during my study.

I wish to acknowledge Mr.Geoffrey Philip Hattersley for his advice, guidance, and encouragement during my study.

There are many people at industrial engineering department, and I really appreciate their helps. I would like to thank all my friends for their great friendship, Mai, Cherry, Jarr, Arm, Big, Kae, Mee, Chang, P'Paitoon, P'Boy, and Ay.

At last but not least, I wish to express my deepest gratitude to my parents (Nitipong and Tusana) and my grateful thanks to my sisters and brother (Kanokporn, Umpai, and Buncha) and my girlfriend (Benjawan) for their encouragement, warmness, love, belief, and pride in my study that have driven me to become successful.

TABLE OF CONTENTS

Page

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Scheduling problems have long been considered as a part of production planning and control, but they have been rarely found associated with the complex scheduling problems frequently encountered in practice. Traditional scheduling problems have been addressed in many specific forms such as single machine, parallel-machine, flow shop, or job shop problems which are able to be taken by the general forms which are more complex and beneficial than the specific forms (Pinedo and Chao, 1999).

The outline of this chapter is organized as follows: Firstly, the general background is explained to provide the definition of scheduling problems, the scheduling function, and the classification of the scheduling problems. Secondly, the importance of scheduling problems, *i.e.* a flexible flow shop problem, sequence-dependent setup times, release dates of machines, and schedule objectives, is indicated. Thirdly, the statement of the problem is addressed. In this section, a problem example is also illustrated. Next, the objectives of this dissertation are proposed. The dissertation scope and the dissertation contribution are given in the fifth and sixth sections of this chapter. Then, the dissertation methodology is presented. In addition, the rational for the solution approaches is also introduced in this section. The organization of the chapters of this dissertation is framed in the final section.

1.1 General Background

This section presents the general background of scheduling theory. It begins with the definition of scheduling and the general conceptual scheme, providing a useful framework for performing the scheduling function effectively. The scheduling function in a production system and the classification of scheduling problems are also stated in this section.

1.1.1 Definition of Scheduling

Scheduling is the allocation of available and limited production resources (*i.e.* workforce, machines, tools, etc.) to perform a number of jobs over time and to meet certain objectives while respecting a set of constraints (Baker, 1974; Pinedo, 1995; Pinedo and Chao, 1999). Another view of scheduling is defined as the determination of the relative position of jobs with respect to a processing machine, including the assignment of definite times at which processing occurs (Nawaz, Enscore, and Ham, 1983).

Scheduling problems, both practical and theoretical aspects, have captured attention from researchers for several years. They involve the assignment of machines to various jobs and the determination of the order in which the jobs will be performed in order to best satisfy some single criterion or several optimization criteria. Generally, there are two kinds of questions concerned in the fundamental decision of the scheduling functions (Baker, 1974):

1. Which resources will be allocated to perform each job?
2. When will each job be performed?

In other words, the essence of scheduling problems has brought about (1) allocation decisions and (2) sequencing decisions.

In addition, two kinds of feasibility constraints are commonly found in scheduling problems. Firstly, there are *limits on the capacity of available resources*, and secondly, there are *technological restrictions* on the orders in which jobs can be performed (Baker, 1974). The former generally refers to machine or workforce capacities and limitations, whereas the latter includes machine eligibility restrictions and precedence constraints. Machine eligibility restrictions mean that not all machines can process all jobs, that is, some jobs can be processed only on some machines. Precedence constraints require that one or more jobs may have to be completed before another job is allowed to start its processing.

1.1.2 The Scheduling Function in a Production System

The scheduling function in a production system has to interface with various other important functions such as production planning, master scheduling, material requirements planning, capacity planning and so on as illustrated in an information flow diagram in Figure 1.1 (Pinedo and Chao, 1999).

In order to provide for departments in an organization to access the necessary scheduling information and in order to enable departments to provide the scheduling system with relevant information such as order quantities, job due dates, release dates, machine status and so on, a management information system (MIS) or a decision support system (DSS) is probably needed.

The process of scheduling in manufacturing starts with *capacity planning* (also called *long-term planning*) which involves facility and equipment acquisition. *Intermediate-term* (or *medium-term*) *planning* includes *aggregate production planning* and *master production planning*. The *aggregate production planning* (also known as *aggregate scheduling*) is concerned with determining the quantity and timing of production for the intermediate future to minimize cost over the planning period by adjusting production rates, labor levels, inventory levels, overtime work, sub contracting rates, and other controllable variables. The *master production schedule* (MPS) then breaks down the *aggregate plan* and develops an overall *schedule* for outputs. It specifies what is to be made (*i.e.* the number of finished products or items) and when and must be in accordance with a production plan. *Material requirement planning* (MRP) is one popular system which has to interact with other decision-making procedures used within the shop floor. In *short-*

term planning, *schedules* translate capacity decisions, intermediate planning, and master schedules into job sequences and specific assignments of personnel, machinery, and materials (Heizer and Render, 2001).

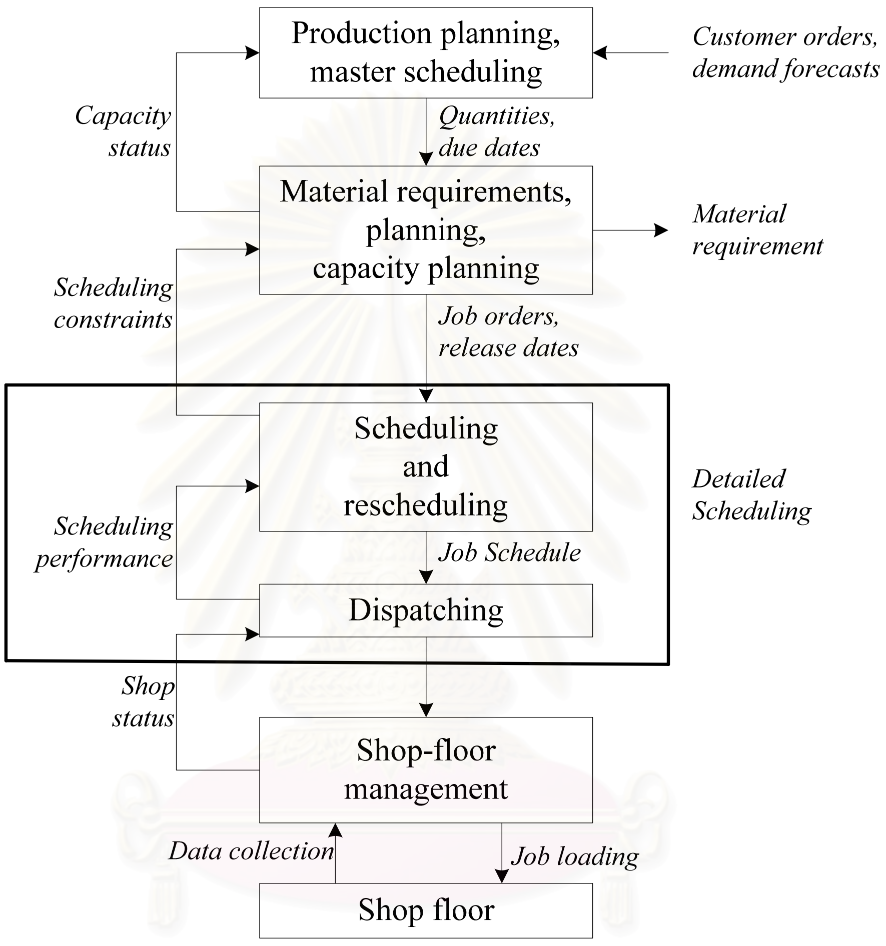


Figure 1.1 An information flow diagram in a manufacturing system

1.1.3 Classification of Scheduling Problems

To consider the classification of the major scheduling problems, it is necessary to characterize the configuration of resources and the nature of tasks. For example, a problem requires one processing step (also called a *single-stage problem*) or several processing steps (called a *multi-stage problem*). A scheduling problem, where the number of jobs to be considered and their release dates are available, is called a *static* case. On the other hand, a scheduling problem, where the number of jobs and related characteristics change over time, is called a *dynamic* case.

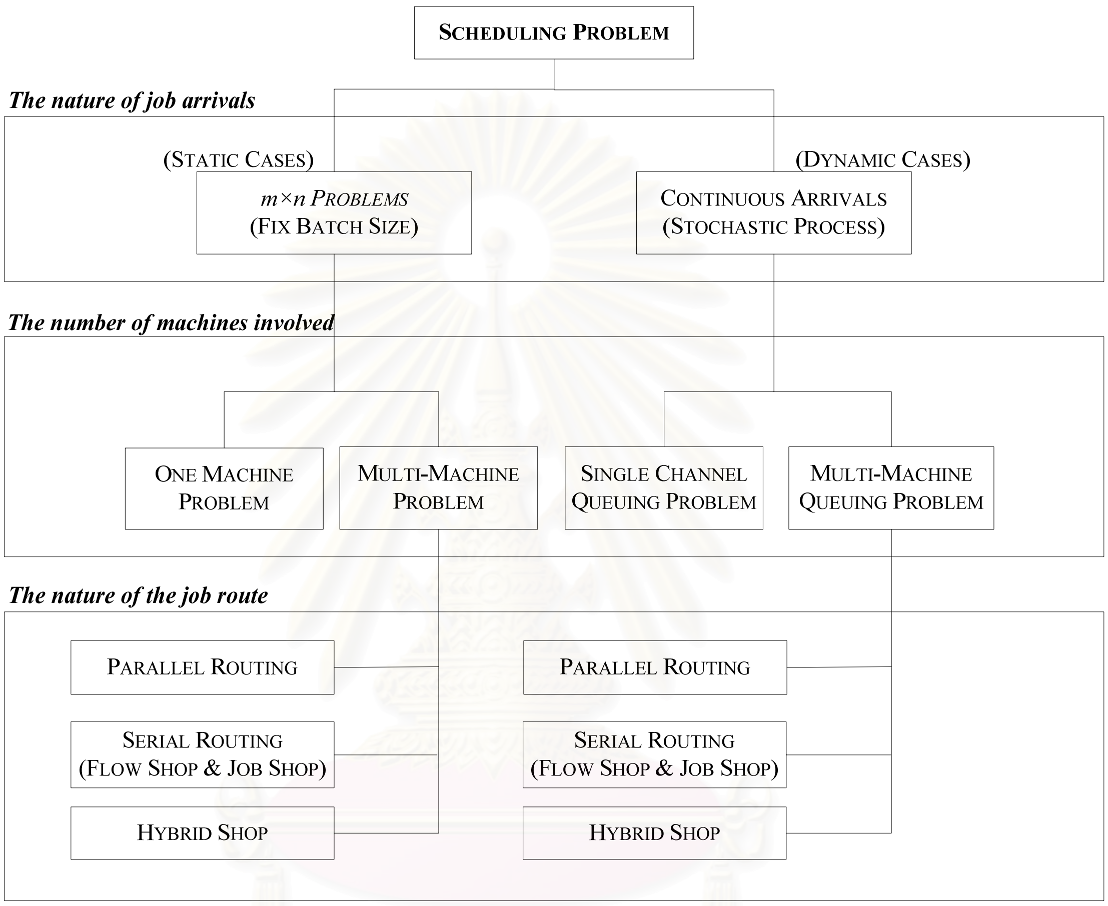

SCHEDULING PROBLEM
The nature of job arrivals
(STATIC CASES)
(DYNAMIC CASES)
m×n PROBLEMS (FIX BATCH SIZE)
CONTINUOUS ARRIVALS (STOCHASTIC PROCESS)
The number of machines involved
ONE MACHINE PROBLEM
MULTI-MACHINE PROBLEM
SINGLE CHANNEL QUEUING PROBLEM
MULTI-MACHINE QUEUING PROBLEM
The nature of the job route
PARALLEL ROUTING
SERIAL ROUTING (FLOW SHOP & JOB SHOP)
HYBRID SHOP
PARALLEL ROUTING
SERIAL ROUTING (FLOW SHOP & JOB SHOP)
HYBRID SHOP

Figure 1.2 Classification of scheduling problems

Let the number of jobs is denoted by *n* and the number of machines by *m*. A schema for a classification of scheduling problems as depicted in Figure 1.2 is presented by Day and Hottenstein (1970). It shows that the scheduling problems have been categorized according to the following three components:

1. *The nature of job arrivals*, namely fixed batch size and continuous arrivals which given by a probability density function,

2. *The number of machines involved*, namely single machine production ($m = 1$) and multi-machine production ($m > 1$), and

3. *The nature of the job route*, namely same route and alternative routes.

The classification of scheduling problems is separated in two groups: (1) $m \times n$ fixed-batch-size problems (or *static cases*) and (2) continuous job arrivals (or *dynamic cases*). The nature of the job arrivals provides the distinction between $m \times n$ fixed-batch-size problems (also called *deterministic problems*) and continuous arrivals (also called *stochastic problems*). In $m \times n$ fixed-batch-size problems, the number of machines and jobs is given in advance. A certain number of jobs arrive in a shop at any time which is known in advance, that is, there are no further jobs unexpectedly arriving, so attention can be focused on scheduling the completely known and available set of jobs. In continuous arrivals, the shop is a continuous process. Jobs arrive intermittently at time that is predictable only in a statistical sense, and job arrivals will continue indefinitely into the future (Conway, Maxwell, and Miller, 1967). Additionally, both the $m \times n$ fixed-batch-size problems and the continuous-arrival problems can be divided into *single-machine problems* and *multi-stage problems*. The single-machine problems are the pure sequencing problems in which an ordering of the jobs completely determines a schedule. Moreover, they are the simplest pure sequencing problems, where there is a single resource or machine. The multi-stage problems are more than one machine. The simple problems are the single-stage sequencing with several machines until the multi-stage problem. The process of such problems in general requires both sequencing and resource allocation decisions, and it is classified as the elementary multi-stage models: parallel machine, flow shop, and job shop systems.

1.2 Importance of the Problem

The importance of scheduling is obvious. It is one of the key functions in modern manufacturing and service companies. Although other functions may be optimized, the success of a company still depends heavily on the performance of scheduling. The importance of scheduling problems is given first in this section. Then, any other importance of scheduling problems related to this dissertation is explained.

1.2.1 Importance of Scheduling Problems

Nowadays, companies are encountered with market demands for a variety of high quality products, and more customers than ever have become selective with rapidly changing needs. The companies must, therefore, compete against these phenomena by making their production systems more flexible, producing the high quality products, and responding rapidly to demand fluctuations. Moreover, some companies face such antagonist decisions as producing items with shorter and shorter life cycles, in small quantities and with the lowest possible costs. Therefore, the ability to make the right product at the right time and at the lowest possible cost remains the key to success for the companies. Hence, the companies need to make schedules that match their production to customer demands by satisfying certain objectives and respecting their set of production constraints.

Practical scheduling, an important decision making process in the operation level, arises in a variety of situations; for example, consider the scheduling of cars to be repaired in a garage, professors to classes in a university, planes to gates at an airport, physicians and nurses to patients in a hospital, production resources to jobs in a manufacturing plant, etc. Therefore, it plays an important role in both most manufacturing and service industries (Pinedo and Chao, 1999).

Due to limited resources existing along the scheduling function, the resources always perform the jobs. However, when the number of jobs is more than the number of resources, the job waiting time or job lateness may occur. Moreover, scheduling programs which do away with an inefficient or inappropriate scheduling process will waste resources and lead to confusion. Therefore, efficiency in scheduling is essential. For example, some companies may use the mathematical techniques or other heuristic methods to allocate scarce resources to a set of jobs. The proper allocation of such resources enables the companies to optimize their objectives in many forms, such as minimizing the time to complete all jobs or minimizing the number of jobs completed after their committed due dates.

Based on the above stated fact that scheduling problems appear in both manufacturing and service industries, this dissertation is however concerned with industrial scheduling problems, where one first has to assign jobs to limited resources and then to sequence the assigned jobs on each resource over time. The scheduling objective, for example, might consist in the building a schedule with the smallest length (minimizing the makespan) and/or meeting job due dates (minimizing the number of tardy jobs, minimizing total tardiness, etc).

1.2.2 Importance of a Flexible Flow Shop Problem

The industrial scheduling problems are interesting in both practical and theoretical viewpoints. The problems have been widely studied since the pioneering work of Johnson (1954) who proposes efficient algorithms for a minimum makespan two-stage flow shop scheduling problem with one machine at each stage and zero release date. All jobs have to pass through a number of stages in the same order, *i.e.* starting at the first stage, at completion going to the second stage and so on until the last stage. Such a production process is called the classical flow shop environment.

In the theoretical aspect, most studies concern many manufacturing systems like a classical flow shop architecture which consists of a single machine on each stage, while in most of practical manufacturing systems for every stage, a number of machines are available that can operate in parallel. Hence, the scheduling problem is more realistic to assume that, at every stage, a number of machines may be available in that it can operate in parallel known as a *flexible flow shop*, *multiprocessor flow shop*, or *hybrid flow shop* problem as depicted in Figure 1.3 (let

m^t be the number of parallel machines at stage t). At each stage, a job needs to be processed by only one of parallel machines. A flexible flow shop problem makes the scheduling problem more complex such that the job processing sequence in each stage may no longer be the same sequence.

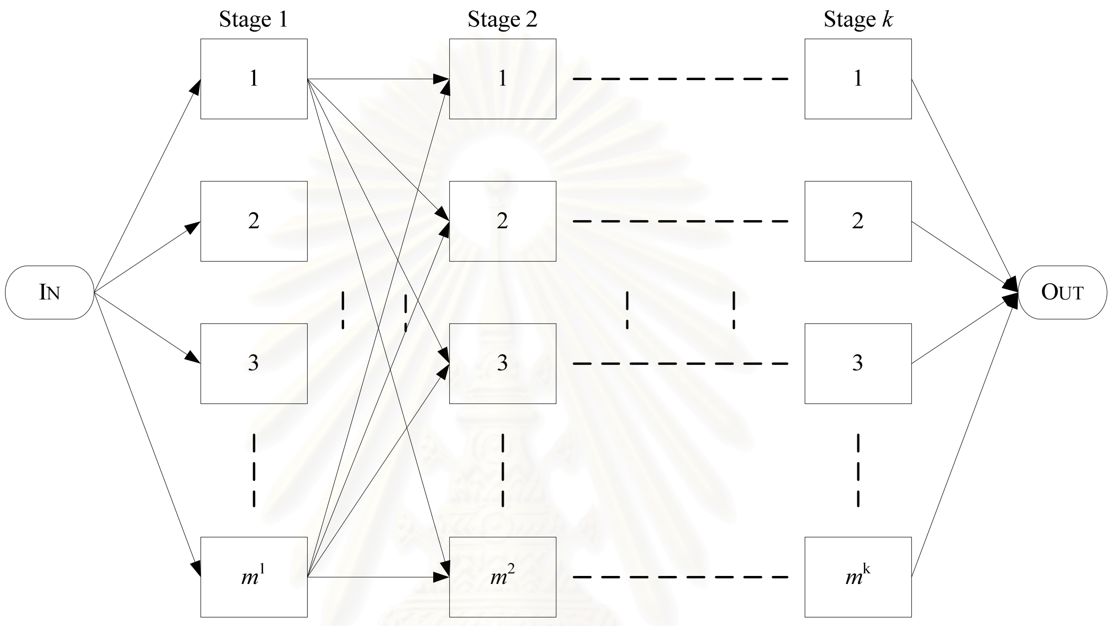


Figure 1.3 A schema of a flexible flow shop environment

For the past three decades, the flexible flow shop scheduling problem has attracted many researchers. Numerous research articles have been published on this topic, *see* the survey in Linn and Zhang (1999), Wang (2005), Quadt and Kuhn (2007), and Allahverdi *et al*. (2008). There are two main reasons for this, among many others (Wang, 2005). Firstly, a flexible flow shop environment is difficult to solve (Garey and Johnson, 1979; Gupta, 1988; Pinedo, 1995). The flexible flow shop problem which has two stages, with one stage having at least two machines, has already been proved to be NP-hard (Hoogeveen, Lenstra, and Veltman, 1996). Thus, it is unlikely that polynomial time algorithms exist for the exact solution of the general problem. Secondly, such a machine scheduling problem can find applications in many real-world applications.

Precisely, this dissertation is mainly concerned with process industries like *e.g.* a glass-container industry (Paul, 1979), a rubber plant (Yanney and Kuo, 1989), a photographic film manufacture (Tsubone *et al.*, 1993), a steel industry (Finke and Medeiros, 2002), a textile manufacture (Karacapilidis and Pappis, 1996), a printed circuit board manufacture (Alisantoso, Khoo, and Jiang, 2003; Hsieh, Chang, and Hsu, 2003) and so on. Such industries are established as multi-stage production flow shop facilities, where at least one production stage is made up of parallel production lines, machines or any other production facility. At some stages, the facilities (*i.e.* machines, lines, etc) are duplicated in parallel to increase the overall capacities of the shop floor, to balance the capacities of the stages, or to either eliminate or reduce the impact of bottleneck stages on the overall shop floor capacities.

Most researchers on the flexible flow shop scheduling problems deal with the two-stage flexible flow shop scheduling. For instance, *see* work of Narasimhan and Mangiameli (1987), Gupta (1988), Deal and Hunsucker (1991), Gupta and Tunc (1991, 1994, 1998), Lee and Vairaktarakis (1994), Chen (1995), Guinet *et al.* (1996), Gupta, Hariri, and Potts (1997), Haouari and M'Hallah (1997), Oguz, Lin, and Cheng (1997), Dessouky, Dessouky, and Verma (1998), Schuurman and Woeginger (2000), Lin and Liao (2003), Guirchoun, Martineau, and Billaut (2005), Haouari, Hidri, and Gharbi (2006), and Low, Hsu, and Su (2008).

Although there are some studies which concern the parallel machines in each stage, they have some non-practical assumptions; for example, they are concentrated on problems with identical machines, that is, each job is processed on any one of the machines in parallel for every stage in the same manner, *see* for instance, Gupta *et al.* (2002), Alisantoso *et al.* (2003), Lin and Liao (2003), and Wang and Hunsucker (2003). However, in the real world situation, 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 they would generally require longer operating time for the same operation. In addition, it may be possible that speeds of machines are still dependent on the jobs assigned. Such a problem is called *a flexible flow shop with unrelated parallel machines*. Hence, in this dissertation, the flexible flow shop problem with unrelated parallel machines is considered, that is, there are different parallel machines at every stage, and speeds of machines are also dependent on the jobs. It is a general form of all kinds of parallel machine problems.

1.2.3 Importance of Sequence-Dependent Setup Times

In some cases where the operator needs to change the configuration of the station in order to process the next job, the change of it may result in an additional production cost or additional time, being necessary to realize the change in the configuration. This process is known as a *changeover* or *setup*. Hence, setup time is the time used to prepare the process of jobs on machines. This includes obtaining tools, positioning work in process material, adjusting tools, returning tools, cleaning up, and inspecting materials. It is very common in many real manufacturing systems.

Scheduling problems involving setup times can be divided into two types. The first type is *sequence-independent* setup time, and the second type is *sequence-dependent* setup time. Setup is sequence-dependent if its duration depends on both the current and the immediately preceding job and is sequence-independent if its duration depends only on the current job to be processed (Allahverdi, Gupta, and Aldowaisan, 1999).

The usual assumption for most scheduling researches is that jobs are sequence-independent. The result of this assumption is that capacity management in these studies requires only the allocation of capacity over the specified time interval. The sum of setup and processing times remains constant for instances of sequence independence, and capacity utilization also remains constant. Such a setup, known as *sequence-independent setup*, has for long been considered negligible and hence ignored or considered as part of the processing time.

However, in some industries, setup is complicated by the fact that the length of setup required depends on the job just completed and on the one about to be started, known as *sequence-dependent setup*. This setup has most significant effects on shop performance as follows: Firstly, the length of setup time directly affects the throughput rate of a production system. Secondly, the cost of unit setup time is usually higher than that of unit processing time since the cost of setup time includes loss of productivity due to machine down-time and the cost of setup itself such as labor cost of a setup technician. Thirdly, the setup change can often be made only by a qualified technician since setup is often a complicated work requiring a higher level of expertise. The qualified workers are limited resources that are not always available.

An example of the sequence-dependent setup operations is in dyeing operations which require setups. Whenever a new color is needed, a dyeing machine must be cleaned. The cleanup time often depends on the color just completed and the color about to be started. In practice, the best sequence is to go from light to dark colors because the cleanup process is easier. In chemical industry, to produce the different chemical compounds may require that some amount of cleansing be carried out between process runs on different compound to insure that tolerably low impurity levels are maintained. In addition, in steel industry, the setup time for the roll changing depends on the type of products just completed and the next type about to be processed.

The importance of setup times has been investigated in several studies. Wilbrecht and Prescott (1969) find that sequence-dependent setup times are significant when a shop is operated at or near full capacity. In a survey of industrial managers, Panwalkar, Dudek, and Smith (1973) discover that about three quarters of the managers have reported at least some operations their schedule requires sequence-dependent setup times, whereas approximately 15 percent of the managers have reported all operations requiring sequence-dependent setup times.

Burns and Daganzo (1987) discuss a flow shop problem with setup costs and distinguish between three different types of setup cost/time as follows:

1. *Waste material* resulting in additional cost owing to, for instance, discard of the paint in the paint shop of an automobile production. This setup cost has only impact on the objective function that minimizes the production costs,
2. *Station downtime* or *labor* required to change the setup. This occurs, for example, when the mounting or a tool needs to be changed. In this case, the schedule of the jobs is directly influenced; it means a sequence without setup time is not possible because some job changes require additional time for preparation, and
3. *Product quality implications* which affect the performance of the station. For instance, the paint quality may temporarily decline when a change of color occurs.

In order to further improve the solution, the setup cost/time is considered completely separated. Allahverdi *et al.* (1999) highlight that when setup cost is directly proportional to setup time, a sequence that is optimal with respect to setup cost is also optimal with respect to setup time. Their survey on setup considerations furthermore considers sequencing problems in flow shop regarding the following characteristics:

1. *Batch setup*: Jobs are grouped into batches and a major setup is incurred when switching between jobs belonging to different batches, whereas a minor setup is incurred for switching between jobs within the batches,

2. *Sequence dependent setup*: Including setup cost/time that depends on the succeeding job gives the possibility to further improve the sequence. In the symmetric case, the appearance of setup cost/time is the same for a change from model *a* to model *b* and for a change from model *b* to model *a*, and
3. *Setup time separable from processing time*: The case in which setup time is separable from processing time leads to the possibility of further reducing the total processing time. This results from the fact that once on a machine *i* a job *j* is finished, and the setup can already be changed way before job $j + 1$ arrives.

1.2.4 Importance of Release Dates of Machines and Jobs

The release date of the machine (or *machine availability*) is the time indicating when the machine becomes available from the previous job in the previous planning period and can start processing in the current planning period. In classical scheduling problems, machines are assumed to be available at time zero onward. However, in most real life industrial settings, a machine can be unavailable at time zero for many different reasons. For instance, some machines may be tied up with unfinished jobs that are carried over from the previous planning period. This machine restriction can also arise from the overlapping of two consecutive planning periods in a system with rolling horizons. It occurs when a new planning period is triggered before all jobs of the previous planning period have been completed, and the assignment of these uncompleted jobs cannot be altered anymore, *e.g.* due to process preparation, To increase the real life situation, the machine availability at nonnegative available time is hence considered in this dissertation.

The release date of the job is also known as the *ready time*. It is the time where a job arrives at the system, that is, the earliest time at which a job can start its processing. Although in the real situation customer orders may not arrive simultaneously, a few papers are devoted to scheduling problems with nonnegative release dates of the jobs. Therefore, it is suitable to concern the job release date where the date is known in advance in this dissertation.

1.2.5 Importance of Scheduling Objectives

The scheduling objectives in such industries may vary. In real life situations, decisions to be made are often constrained by specific requirements. More importantly, these requirements are typically conflicting in nature. The decision making process gets increasingly more complicated with increment in the number of requirements. Modeling and development of the solution methodology for these scenarios have been the challenge for operations researchers. Generally speaking, scheduling problems in real life applications involve optimization of more than one criterion. The due-date related criteria and the makespan criterion may be important and have been used by many researchers, which they are included in this dissertation.

The motivation to consider the problem of generating an efficient set of schedules for the flexible flow shop environment comes from the variety of industrial cases where the criteria related to efficiently utilizing resources and completing orders by their due dates are important. For example, the makespan and the number of tardy jobs are used as the scheduling criteria.

Makespan is a traditional objective for much of the flexible flow-line literature. In addition to its applicability to a periodic multiple product scheduling environment, the makespan criterion can also be considered as a surrogate for capacity maximization and flow time criterion. Pinedo (1995) explains that lower makespan usually indicates an efficient utilization of resources. Moreover when dealing with machines in parallel, the makespan becomes an objective of significant interest. In practice, one often has to deal with the problem of balancing the load on machines in parallel, and by minimizing the makespan the scheduler ensures a good balance (Pinedo, 1995).

The number of tardy jobs may at first appear somewhat artificial and of no practical interest. However, in the real-world it is a performance measure that is often monitored and used to rate managers' performance. It is equivalent to the percentage of on-time shipments (Pinedo, 1995). In many cases, a manufacturer wants not only the conformity to schedule but also the minimization of the number of tardy jobs in order to avoid the penalties that may incur. Thus, it is directly related to improving the customer's satisfaction, which is becoming a more critical issue for management. In this criterion, changing any job's completion time changes the value of the objective, whereas in the makespan criterion, only one job defines the value of the objective.

1.3 Statement of the Problem

This dissertation addresses the job scheduling problem in a system, which is called a *flexible flow shop environment*. There are customer orders (or *jobs*) to be processed in this system. Customer orders may not arrive simultaneously, but their arrival times, called the *release dates*, are known and fixed by customers. The *due dates* of jobs are designated by customers. Each job is allowed to start its processing whenever it is ready, that is, there are no precedence constraints and priority rules. Each job is an entity, and it cannot split into sub-jobs, that is, job splitting is not permitted.

The system (or *machine environment*) consists of many production stages in series, where jobs have to undergo multiple operations in the same order (referred to as a *flow shop environment*). In addition, there are a number of machines in parallel at each stage, where jobs have to be processed on any one of the parallel machines in each stage (referred to as a *flexible flow shop environment*). Even though each machine in each stage can perform all jobs, the time that each job spends on the machines may be different. In this dissertation, the time to perform any job on the machines depends on the jobs and the machines (referred to as *unrelated parallel machines*). There are infinite buffers between all adjacent stages, as well as before the first stage and after the last stage. Figure 1.4 shows a general schematic of the production process under consideration.

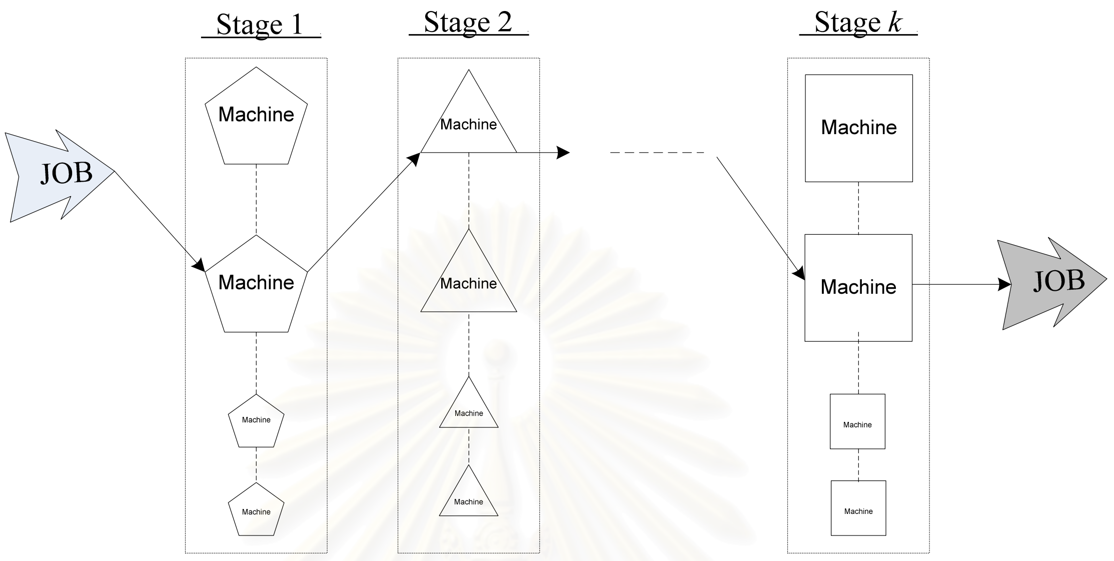


Figure 1.4 A general schematic of the production process

This problem considers only one planning period. The times for the uncompleted jobs from the previous planning period are reserved on the machines (referred to as *machine release dates* or *machine availability*). In other words, machines in the current planning period are available from time non-negative onwards. Moreover, it is assumed that there are no machine breakdowns as well as scheduled or unscheduled maintenance. In addition, the problem is assumed to be a *static* scheduling problem, that is, the current schedule will not change even though there are new jobs that enter the system. Consequently, the new jobs and the jobs that have never been performed in the current planning period will be scheduled in the following planning periods.

Setups need to be considered in the problem. They are the addition processes where the operators need to change the configuration of the station in order to process the next job, and they result in additional production costs and times. The setup of the next job can start whenever the job arrives at the machine and the machine becomes free to process the job (referred to as a *non-anticipatory setup*). In addition, the setup time (cost) depends on the current job and the immediately preceding job (called *sequence-dependent setup time*) during the current planning period. However, there are hidden sequence-dependent setups that occur between the last unfinished job in the previous planning period and the first job in the current planning period on the same machine. Consequently, it seems that the setup time of the job that is processed in the first position on any machines depends on the machine that the job is processed (referred to as *machine-dependent setup time*). In this dissertation, such a machine-dependent setup time is called *changeover time*.

All jobs must be completed without interruption in the system (known as *non-preemption*) and cannot be cancelled before their completion. It is assumed that all jobs are always processed without error and do not revisit the stage that they have already visited. There is no transportation time between two consecutive stages, that is, they are available for processing at a stage immediately after completing processing at the previous stage. No two operations of the same job are processed simultaneously, and no machine processes more than one job at a time.

This problem is now to find a feasible solution consisting of a feasible assignment and a feasible schedule, such that the completion time of the last job to leave the system (called the *makespan*) and the number jobs that cannot be sent to the customers by their due dates (called the *number of tardy jobs*) for the current planning period is minimized by using the positively convex of combination.

1.4 Dissertation Objectives

1. To formulate a mathematical model to solve the problem of a flexible flow shop problem with unrelated parallel machines and sequence-dependent setup times and to produce an optimal schedule in order to minimize the makespan and the number of tardy jobs, and
2. To develop constructive and iterative algorithms to find approximate solutions for large-sized problems.

1.5 Dissertation Scope

The scheduling problem considered is a flexible flow shop problem. Specifically, this dissertation studies a problem that has unrelated parallel machines at least one production stage. All data in this problem are assumed to be known deterministically when scheduling is undertaken. The scheduling problem is characterized by three components:

(1) The machine environments and job characteristics,
(2) The processing characteristics and a set of constraints that must be satisfied for each schedule, and
(3) The objective function characteristics that must be minimized.

Hence, the scope and assumptions of this study are classified in three components as follows:

1.5.1 Machine Environments and Job Characteristics

This section describes the details of machine environments and job characteristics studied in this dissertation as follows:

1.5.1.1 Machine Environments

1) This is a flexible flow shop scheduling problem whose system is a k-stage flexible flow shop ($k \geq 2$) environment. A stage is a machine or set of machines that perform the same operation, and
2) Some stages are made up of a number of machines ($m^t \geq 2$, $\exists t \in \{1, 2, \ldots, k\}$, where m^t be the number of machines at stage t). It has a set of unrelated parallel machines, each of

which can process all jobs but the processing time of each job on each machine may be not necessary to equal.

Let p_{ij}^{t} denote the processing time of job *j* on machine *i* at stage *t*, ps_j^t be the standard processing time of job *j* at stage *t* , and v_{ij}^{t} be the relative speed of machine *i* for job *j* at stage *t*. In general, for each stage *t*, one can distinguish among the following three cases:

a) *Identical parallel machines*: $p_{ij}^{t} = ps_j^t$ for all *i* and *j*,

b) *Uniform parallel machines*: $p_{ij}^{t} = \dfrac{ps_j^t}{v_i^t}$ for all *i* and *j*, where v_i^t is the relative speed of machine *i* at stage *t*, and $v_i^t = v_{ij}^t$ for all *j*, and

c) *Unrelated parallel machines*: $p_{ij}^{t} = \dfrac{ps_j^t}{v_{ij}^t}$ for all *i* and *j*.

1.5.1.2 Job Characteristics

The number of jobs is known and fixed. Each job to be scheduled in the current planning period comes from two sources, the new customer orders in this planning period and the remaining jobs from the previous planning period that are not carried out. Each of the *n* jobs is an entity. Even though the job is composed of distinct operations, no two operations of the same job may be processed simultaneously. There are own due dates (d_j).

1.5.2 Processing Characteristics and a Set of Constraints

This section provides detailed scope and assumptions of processing characteristics and constraints as follows:

1) The release dates or arrival times of all jobs are nonnegative release dates, that is, jobs cannot start their processing times before their release dates (r_j). However, they are assumed to be known with certainty,
2) The sequence-dependent setup times may exist between two consecutive jobs at each stage. Setup can only be preformed after the machine is no longer processing any job and the job for which setup is being performed is ready. Moreover, the length of time required to do the setup of each stage t, s_{lj}^{t}, depends on both the prior job *l* which the machine processes and the current job *j* to be processed. However, it is also assumed that the setup times are equal for all machines in the same stage when changing from one job to another,
3) The machine-dependent setup times occur only when the job assigned to each machine at the first position at each stage in the studied period. It means that the length of setup time (or changeover time, ch_{ij}^{t}) of job *j* depends on the machine performing it since each machine has different unfinished jobs in the pervious planning period, so the length of changeover time depends on the unfinished job in the previous planning period and the first new job in the current planning period, which is the reason why the changeover time depends on machines,
4) Preemption is not permitted; *i.e.* once operation is started, it must be completed without interruption,
5) There are no precedence constraints and no job priority constraints, that is, each job is allowed to start its processing whenever it is ready,
6) Jobs cannot be cancelled before their completion,
7) Each job visits the stage in same order, but it does not revisit a stage that it has already visited, and the process flow for each job is known in advance,

8) Jobs are always processed without error and have no associated priority values,
9) At any time, every job can be processed at most one machine, and every machine can process at most one job,
10) There are infinite buffers between all two consecutive stages as well as before the first stage and after the last stage, so there is no blocking of machines, that is, in-process inventory is allowed,
11) There are no machine breakdowns and scheduled or unscheduled maintenance. The machines are however continuously available from time non-negative a_i^t onwards. The machine release dates (a_i^t) mean the times that machines are reserved to perform the unfinished jobs carried over from the previous planning period,
12) There is no transportation time between stages; jobs are available for processing at a stage immediately after completing processing at the previous stage, and
13) All data used in this dissertation (*i.e.* standard processing time, relative speed of machines, release date, due date, setup time and so on) are randomly generated from the uniform distribution.

1.5.3 Objective Function Characteristics

Let the completion time of job *j* at stage *k*, it means the time that job *j* can leave the system, be C_j^k. The criteria for considering the production improvement in the scheduling method will be:

a) The makespan (C_{max}): The makespan, defined as $\max_{j\in\{1..n\}}\{C_j^k\}$, is equal to the completion time of the last job to leave the system, which is defined as:

$$C_{max} = \max_{j\in\{1,\dots,n\}}\{C_j^k\} \tag{1.1}$$

b) The number of tardy jobs (η_T): The number of jobs whose completion times are larger than their due dates.

Associated with each job j, a due date $d_j \geq 0$. Let $U_j = 1$ if the completion time C_j^k of job j is larger than due date for job j, otherwise $U_j = 0$. The total number of tardy jobs (η_T) is defined as:

$$\eta_T = \sum_{j=1}^{n} U_j \qquad (1.2)$$

1.6 Dissertation Contribution

According to stated details of the importance of problems in Section 1.2, this problem is worth studying. This section addresses the contribution of this dissertation based on two main attractions of this problem (Wang, 2005). Firstly, flexible flow shop problem is a category of machine scheduling problems where most machine scheduling problems are difficult to solve and have been proved to be NP-hard or NP-complete. Secondly, most machine scheduling problems can find applications in the real world.

Gupta (1988) has studied the minimum makespan problem in a two-stage flexible flow shop which $m^1 \geq 2$ and $m^2 = 1$. He states that the minimum makespan problem in a two-stage flexible flow shop is NP-hard when max $\{m^1$ and $m^2\} > 1$. This result is very important because it shows that any minimum makespan problem in a k-stage flexible flow shop is NP-hard since the k-stage flexible flow shop can always be reduced to a two-stage flexible flow shop. Consequently, the flexible flow shop problem is at least that difficult, and it is well worth studying.

Although the processing complexity is a major direction of development, it is clearly shown in the literature review that most of the work is the two-stage flexible flow shop problems. While interesting results have been obtained for these cases, there has been less work done on the *k*-stage ($k \geq 3$) ones. Thus, the complexity level would significantly increase as it extends to three or higher number of stages.

Moreover, although the flexible flow shop problem has been widely studied in the literature, most of the studies related to the flexible flow shop problem are concentrated on the problem with identical processors, *see* for instance, Bratley, Florian, and Robillard (1975), Brah and Hunsucker (1991), Deal and Hunsucker (1991), Gupta and Tunc (1991, 1994, 1998), Rajendran and Chaudhuri (1992), Santos, Hunsucker, and Deal (1995), Guinet and Solomon (1996), Gupta *et al.* (2002), Alisantoso *et al.* (2003), Lin and Liao (2003), and Wang and Hunsucker (2003). In this dissertation, the flexible flow shop problem with unrelated parallel machines is however considered, that is, there are different parallel machines for every stage, and speeds of the machines are also dependent on the jobs.

To simplify the problem, setup times are seldom considered in the scheduling. They are fixed and included in processing times, *see* Negenman (2001) and Engin and Döyen (2004). However, several industries encounter sequence-dependent setup times which result in even more difficult scheduling problem. In this dissertation, sequence-dependent setup time restriction is therefore taken into account as well.

Most researchers suppose that machines are always available at time zero. It means that schedulers can assume that no jobs exist on every machine during scheduling. However, in the real life of scheduling problem machines may be subject to some unavailability periods due to uncompleted jobs on the machine. There has been no serious research work reported on this subject despite the fact that it is the realistic machine scheduling.

Moreover, customer orders may not arrive simultaneously in real-life problems, so assuming that *n* independent jobs are available at zero is not an acceptable assumption. Thus, this dissertation concerns the problem as the unequal release date scheduling problem.

Scheduling criteria are the measures upon which schedules are to be evaluated. Two board classes of criteria are the schedule cost and the schedule performance. Such costs are often difficult to measures, so frequently an important cost-related measure of system performance can be used as a substitute for total cost system (Baker, 1974). In theoretical research, most researchers have addressed single criterion problems. The minimization of makespan is the most widely used criterion, *see* Moursli and Pochet (2000), Schuurman and Woeginger (2000), and Engin and Döyen (2004). However, companies are usually faced with the problem of satisfying several different groups of people. Thus, there is a need for further research in multi-criteria scheduling problems.

It can conclude that this dissertation problem is also of great academic interest. The scheduling literature has dedicated a great deal of attention to the flow shop machine environment while basically ignoring the equally important flexible flow shop machine environment. To simplify the problem, most researchers focus two-stage or *k*-stage problems with identical machines, and the setup times are fixed and included in processing times, in contrast to this problem. At the same time, most scheduling models have concerned the makespan as the objective, while results for models with dual criteria, number of tardy jobs and makespan, are rare, and most of these are limited to very simple cases like the single machine problem or the two-stage problem. To best of the knowledge, this is a brand new problem for serious study in scheduling literature (Allahverdi *et al.*, 2008).

1.7 Dissertation Methodology

After determining the context in which scheduling is being defined, the methodology for selecting a “good” schedule solution is determined. This section addresses the dissertation methodology.

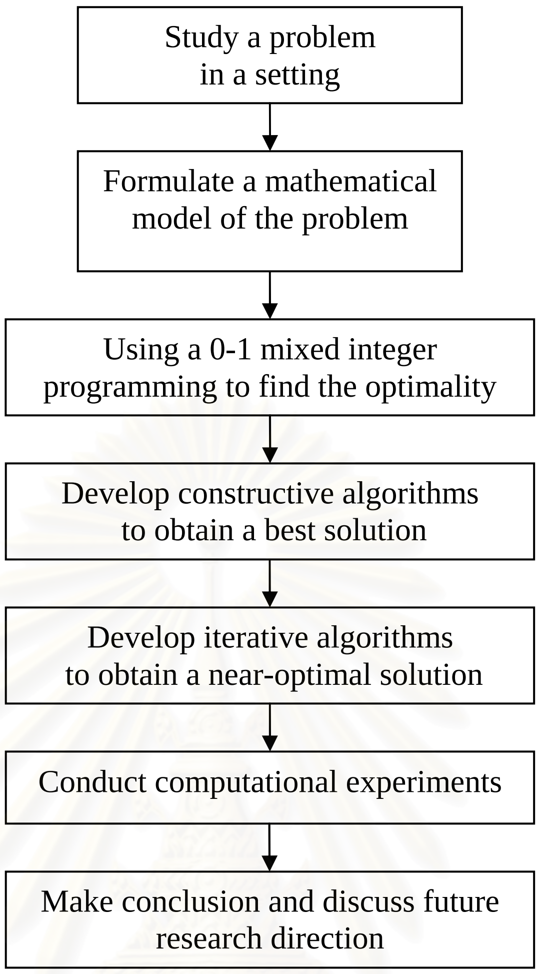

Study a problem in a setting
Formulate a mathematical model of the problem
Using a 0-1 mixed integer programming to find the optimality
Develop constructive algorithms to obtain a best solution
Develop iterative algorithms to obtain a near-optimal solution
Conduct computational experiments
Make conclusion and discuss future research direction

Figure 1.5 The research methodology

Figure 1.5 shows the dissertation methodology. The common solutions to solve the static scheduling problem are described first in this section. Then, a way of dealing with this dissertation problem is presented. Finally, the rational for approaches is addressed.

1.7.1 Common Solutions for Scheduling Problems

Scheduling problems are typically represented as the combinatorial optimization problems. They maximize or minimize functions of many variables subject to some problem specific constraints and some integrality restrictions imposed on all or some of the variables. Often, a combinatorial optimization problem can be modeled as an integer program (Nemhauser and Wolsey, 1999). However, these problems are often very difficult to solve, which is captured by the fact that many such problems are NP-hard or NP-complete. Due to their difficulty and numerous practical importance, a large number of solution techniques for attacking NP-hard integer and combinatorial optimization problems have been developed. The available algorithms can be classified into two main classes: *exact* and *approximate* algorithms.

Exact algorithms are guaranteed to find the optimal solution and to prove its optimality for every finite size instance of a combinatorial optimization problem within an instance-dependent, finite run-time. If optimal solutions cannot be computed efficiently in practice, the only possibility is to trade optimality for efficiency. In other words, the guarantee of finding optimal solutions can be sacrificed for the sake of getting very good solutions in polynomial time. A class of approximate algorithms is that of *heuristic methods*, or simply *heuristics*, and seeks to obtain this goal.

Two techniques from each class that have had significant success are integer programming (IP), as an exact approach, and local search and extensions thereof called metaheuristics, as an approximate approach. The IP approach is a class of methods that relies on the characteristic of the decision variables of being integers. Some well-known IP methods are branch-and-bound, branch-and-cut, dynamic programming and so on, *see e.g.* Brah and Hunsucker (1991), Moursli and Pochet (2000), Sawik (2000, 2002, 2007), and Lee and Asllani (2004). However, for most of the available IP algorithms the size of the instances solved is relatively small, and the computational time increases strongly with increasing instance size, whereas the local search algorithm has been shown to be the most successful class of approximate algorithms, *i.e.* Santos *et al.* (1995, 1996). It yields high-quality solutions by iteratively applying small modifications (local moves) to a solution in the hope of finding a better one. Embedded into metaheuristics designed to escape local optima such as simulated annealing, tabu search, genetic algorithm, or iterated local search, these approaches have been shown to be very successful in achieving near-optimal (and sometimes optimal) solutions to a number of difficult problems, *i.e.* Lee and Asllani (2004).

1.7.2 Solution Methodology for the Dissertation Problem

This section addresses the solution methodology for this dissertation problem. According to the common solutions for scheduling problem in Section 1.7.1, the available algorithms for this dissertation problem can be classified into two main categories: *exact* and *approximate* algorithms. In addition, the approximate algorithms can be classified into two groups: *constructive* algorithms, and *artificial intelligence* (*iterative* algorithms) search techniques (Wang, 2005). Hence, there are three solution approaches used to solve this dissertation problem. These approaches are described below:

(1) Exact algorithms

This dissertation problem is able to take the form of a mathematical model that expresses the desired objectives subject to the constraints set forth in the problem. The most well-known general technique studied in this dissertation is a 0-1 mixed integer programming method, where the solution can be obtained by running the commercial mathematical programming software (Sawik, 2000, 2002, 2007).

(2) Constructive algorithms

The NP-completeness of scheduling problem makes it computationally expensive to use an exact solution technique (Lee and Asllani, 2004; Nearchou, 2004), which gives the optimal solution of a problem. It is impractical for most problems since the solution time would be unacceptably long. That is why an approximate solution technique is needed. Approximate solution techniques, or heuristics, may provide the optimal solution for a problem even though they do not guarantee the optimal solution. This fact implies that there may be some spaces for a heuristic to be improved.

In this dissertation, some dispatching rules, namely the Shortest Processing Time (SPT), Longest Processing Time (LPT), Earliest Due Date first (EDD), Earliest Release Date first (ERD), Minimum Slack Time first (MST), and Slack time per Processing time (S/P) rules, and flow shop makespan heuristics, which are the algorithms given by Palmer (1965), Campbell, Dudek, and Smith (1970), Gupta (1971), and Dannenbring (1977) as well as the insertion heuristic by Nawaz *et al*. (1983), will be used for flexible flow shop environment as the constructive algorithms. Upon these heuristics, the starting job sequence for the first stage is generated by using them. Then, use the greedy search policy to determine the relative position of jobs with respect to any machines. In addition, the new job sequence for the other stages will now be used, by using either permutation or First-In-First-Out rules to construct a schedule for the problem. The solution is then set equal to the best function value obtained by both rules.

Moreover, after several constructive algorithms have been adapted, several fast polynomial improvement heuristics will be applied to the solutions to improve the quality of solution (called *improvement algorithms*).

(3) Iterative algorithms

To increase the near-optimal solution from the constructive algorithms, iterative metaheuristic algorithms or artificial intelligence (AI) search techniques are proposed to solve the flexible flow shop problem in this dissertation. The AI search techniques are different from other heuristics. They search for a good solution by using or learning certain knowledge (usually the knowledge is problem specific), which belongs to the category of AI search techniques. However, they do not guarantee optimal solutions. There are three types of the AI search techniques commonly applied in solving optimization problems in this dissertation: simulated annealing, tabu search, and genetic algorithms.

1.7.3 Rational for the Solution Approaches

This section addresses the rational for solution approaches studied in this dissertation. Thus, the important issue of the solution methodology for this dissertation problem is given. The importance of exact algorithms is presented first. Then, the importance of approximate algorithms (for both constructive and iterative algorithms) is given.

1.7.3.1 Importance of Exact Algorithms

Although the flexible flow shop problem with unrelated parallel machines and sequence-dependent setup times is difficult to solve optimally for a large-sized problem, *i.e.* Brah and Hunsucker (1991) and Kurz and Askin (2004), an exact procedure using a mathematical programming formulation is generally accepted for solving small-sized problems.

In recent years, remarkable improvements have been reported for the integer program when it applies to some difficult problems, *i.e.* Sawik (2000, 2002, 2007). However, for most of the available integer programming algorithm the size of the instances solved is relatively small, and the computational time increases strongly with increasing instance size. Additional problems are often due to the facts that:

1. The memory consumption of exact algorithms may lead to the early abortion of a program,
2. High performing exact algorithms for one problem are often difficult to extend if some details of the problem formulation change, and

3. For many combinatorial problems, the best performing algorithms are highly problem specific, and they require large development times by experts on integer programming.

Nevertheless, there are some important advantages for formulating the exact algorithm from the mathematical programming as follows:

1. Proven optimal solutions can be obtained if the algorithm succeeds,
2. The mathematical programming provides a better understanding of the problem, which will be useful in formulating relaxed problems and in developing heuristic procedures,
3. 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 they may be able to solve practical size problems in near future,
4. Valuable information on the upper/lower bounds to the optimal solution is obtained even if the algorithm is stopped before completion, and
5. A more practical advantage of mathematical methods is that powerful, general-purpose tools such as CPLEX often reach astonishingly good performance.

1.7.3.2 Importance of Approximate Algorithms

In general, the computational effort required to find an optimal solution grows exponentially with the size of the problem. The exact algorithms for finding an optimal solution in polynomial time are therefore unlikely to exist. That is why the approximate solution techniques are necessary. Advantages of the approximate methods are that:

1. In practice, the approximate algorithms are found to be the best performing algorithms for a large number of problems,
2. The approximate algorithms can examine an enormous number of possible solutions in reasonable or acceptable computation time,
3. The approximate algorithms are often more easily adapted to variants of problems, so they are more flexible, and
4. The approximate algorithms are typically easier to understand and implement than exact methods.

However, disadvantages of the approximate algorithms are that typically:

1. The approximate algorithms cannot prove optimality,
2. The approximate algorithms cannot provably reduce the search space,
3. The approximate algorithms do not have well defined stopping criteria (this is particularly true for metaheuristics), and
4. The approximate algorithms often have problems with highly constrained problems where feasible areas of the solution space are disconnected.

It concludes that this dissertation problem belongs to the class of combinatorial optimization problems characterized as NP-hard, so the right way to perform is with heuristic (Nearchou, 2004). According to the advantages of approximate algorithms, several concepts of constructive algorithms, namely SPT, LPT, ERD, EDD, MST, S/P, PAL, CDS, GUP, DAN, and NEH, are adapted as constructive algorithms for this dissertation problem.

The question raised for the constructive algorithms is whether it is possible to improve the solution quality. Sticking into local optima is one drawback of constructive algorithms, while most iterative algorithms (or AI search techniques) always try to find a better solution or escape from a local optimal to find the globally better solutions. This is why iterative algorithms can improve flexible flow shop solutions. Jones, Mirrazavi, and Tamiz (2002) show that 70% of the articles utilize the genetic algorithm as the primary metaheuristic, 24% simulated annealing, and 6% tabu search. The genetic algorithm is so popular because of its flexibility. Thus, to determine near-optimal solutions, the simulated annealing, tabu search, and genetic algorithms are proposed as iterative algorithms.

1.8 Dissertation Organization

This dissertation addresses a scheduling in flexible flow shop problem with unrelated parallel machine and sequence-dependent setup times. It is organized as follows:

Chapter I states the general background and introductory problem to provide the definition of scheduling problems, the scheduling function and the classification of the scheduling problems. The importance of scheduling problems, a flexible flow shop problem, sequence-dependent setup times, release dates of machines, and schedule objectives, is indicated. The statement of the problem and the objective of this dissertation are addressed. The dissertation scope, the dissertation contribution, and the dissertation methodology are presented.

At the beginning of Chapter II, the brief review of machine environments for scheduling problems, which are single machine, parallel machines, flow shop, and job shop, is explained. The survey of the flexible flow shop problem, which consists of the flexible flow shop environment review, the setup time scheduling problem review, the scheduling objective function review, and the flexible flow scheduling procedure review, is presented.

In Chapter III, the mathematical model considering three main conditions, namely unrelated parallel machines, sequence-dependent setup time between jobs, and machine-dependent setup time of a job, is formulated to find the optimal scheduling solution by using the a commercial algebraic modeling language AMPL with CPLEX solver that runs on Windows platform. The total number of possible sequence combinations that are generated by a complete enumeration method is estimated to illustrate the complexity of the problem.

The heuristic solution concepts are proposed in Chapter IV. They will consist of three main kinds of the heuristic approaches which are the constructive algorithms, improvement algorithms, and iterative algorithms. The constructive algorithms proposed are adapted the ideas of the dispatching rules and flow shop heuristic algorithms. Then, the improvement algorithms are proposed by using the neighborhood exchanges to improve the solution obtained from the constructive algorithm. The iterative algorithms based on the artificial algorithms, namely tabu search, simulated annealing, and genetic algorithms, are used to find the schedule solution.

Chapter V provides the computational experiments of the heuristic algorithms that are proposed in Chapter IV. The performance of each algorithm is compared to the best heuristics that are found in these tests on medium- and large-sized test problems, whereas the performance on small-sized test problems is compared to the optimal solution obtained from the 0-1 mixed integer programming in Chapter III. In addition, the recommended heuristic solution approach is proposed.

The research summary of this dissertation is provided in Chapter VI, where conclusion is presented. The future research is briefly discussed in view of theoretical, computational, and empirical research.

CHAPTER II

LITERATURE REVIEW

This chapter reviews the literature that is related to the dissertation problem, whose topic is a scheduling problem in flexible flow shop environment. As stated in Chapter I, the flexible flow shop environment can be seen as the combination of the classical flow shop environment and the single stage parallel machine environment. The classical machine environments for scheduling problems which are single machine, parallel machines, flow shop, and job shop, as well as the complex machine environment, namely the flexible flow shop environment, are explained in this chapter. More other general scheduling studies are found in Baker (1974), Pinedo (1995), and Pinedo and Chao (1999). More details of the flexible flow shop scheduling problems are found in Linn and Zhang (1999), Wang (2005), Quadt and Kuhn (2007), and Allahverdi *et al.* (2008).

This chapter is organized as follows: Firstly, the brief review of machine environments for scheduling problems, which are single machine, parallel machines, flow shop, and job shop problem, is explained. Secondly, the flexible flow shop scheduling problem, which is the problem under consideration, is introduced. Finally, the survey of the flexible flow shop problem is presented. It consists of the flexible flow shop environment review, the setup time scheduling problem review, the scheduling objective function review, and the flexible flow shop scheduling procedure review.

2.1 Machine Environments for Scheduling Problems

There are many important machine environments in the scheduling problems presented in the literature such as single machine, parallel machines, flow shop, and job shop. In this section, the definition of each machine environment is explained.

2.1.1 Single Machine Problem

The single-machine model is the simplest pure sequencing problem; there is one machine that can process at most one job at a time. It requires only the sequencing decision. It has been the most popular of all scheduling problems.

Numerous results exist for a large number of problem specifications. The best known results are the procedures for minimizing a weighted sum of completion times (Smith, 1956) and for minimizing the maximum tardiness (Jackson, 1955); both of these procedures ascertain the optimal job sequence by means of a simple ordering of the jobs, WSPT (Weighted Shortage Processing Time sequencing), which sequences the jobs in non-decreasing order of their ratios of processing time to weight, and EDD (Earliest Due Date sequencing), which sequences the jobs in non-decreasing order of deadlines.

A slightly more complex procedure is that of Moore (1968) for minimizing the number of late jobs. While the previous-mentioned problems have proved to be quite easy, the problem of minimizing weighted tardiness has been considerably more difficult because tardiness is not a linear function of completion time. Wilkerson and Irwin (1971) introduce a suboptimal procedure for minimizing the mean tardiness.

In earlier literature, most researchers examine the basic single-machine model with regular measures of performance, which are non-decreasing in job completion times. However, the recent literature has begun to change with the growing interest in Just-In-Time (JIT) production, which espouses the notation of earliness and tardiness (known as an E/T problem), *see* Ventura and Radhakrishnan (2003), Valente and Alves (2005), and Hino, Ronconi, and Mendes (2005).

2.1.2 Parallel-Machine Problem

The one-stage parallel-machine problem is similar to the single-machine model except that each job may be performed on any of the parallel machines. It requires both sequencing and resource allocation decisions and is an important generalization of the single-machine problem; however, the single-machine problem is introductorily of theoretical interest.

The parallel machine models are divided into three groups: identical parallel machines, uniform parallel machines, and unrelated parallel machines. In the *identical parallel-machine* case, there are *m* machines in parallel, that is, each job requires a single operation and may be processed on any one of the machines. Under *uniform parallel-machine* case, the machines in parallel have different processing speeds. (A set of machines is uniform if the operation time to perform any job on a particular processor is proportional to that machine's speed. In other words, each machine is a scale replica of some base unit.) The *unrelated parallel-machine* case is another generalization of the parallel machine model. The operation time for every job is dependent on the machines, and the speeds of machines are dependent on the jobs.

Unfortunately, most of the researchers assume that the machines are identical, *see* Elmaghraby and Park (1974), Sarin, Ahn, and Bishop (1988), Belouadah and Potts (1994), Ho and Chang (1995), Azizoğlu and Kirca (1999), and Lin and Jeng (2004). The most common objective function studied for the problem is the minimization of makespan, weighted flow time, or weighted tardiness.

2.1.3 Flow Shop Problem

The flow shop model, which is the problem with multiple machines in series, is assumed that all jobs are to be processed on the same route of which an identical precedence ordering of the processing steps, which is unidirectional. It is the simplest multistage scheduling problem, and it is to determine how to sequence the jobs on each machine, where each job must visit each machine in the prescribed order.

A common assumption made by most researchers is to restrict attention to schedules for which the sequence of jobs is identical on each machine, *see* Reeves (1995), Chung, Flynn, and Kirca (2002), and Iyer and Saxena (2004). Such schedules are called *permutation schedules* and have been shown to be optimal for all two-stage and three-stage problem with makespan; however, in general, permutation schedules need not be optimal.

Most of researches have also been limited to considering the makespan criterion with nonpreemptive schedules. The best known result for scheduling in the flow shop environment is that of Johnson (1954) for the two-stage problem. He presents a simple list-scheduling algorithm for minimizing the makespan. While numerous combinatorial optimization procedures have been proposed for solving the general flow shop problem, work on heuristic procedures such as genetic algorithms, simulated annealing, and tabu search has paralleled the work on optimal procedures for the flow shop problem. Most noteworthy heuristics for the makespan criterion are those of Campbell *et al.* (1970) and Nawaz *et al.* (1983).

2.1.4 Job Shop Problem

The job shop model is the most general production scheduling problem in the classification; here there are no restrictions on the processing steps for a job, and alternative routes for a job may be allowed. Moreover, a job may visit a machine more than once. Like the flow shop model, the job shop model has multiple machines in series, but it often has different routes.

Most researchers have assumed that all jobs are nonpreemptive and that the criterion is to minimize the makespan, *see* Kumar and Srinivasan (1996), Sule and Vijayasundaram (1998), and Park, Choi, and Kim (2003).

2.2 Flexible Flow Shop Scheduling Problem

The flexible flow shop environment — also commonly referred to as *hybrid flow shop*, *flow shop with parallel machines*, or *multiprocessor flow shop* — is one complicated type of the machine environments which can be seen as the combination of the parallel-machine and flow shop machine environments. The flexible flow shop model is a generalization of the classical flow shop model and the parallel machine environments which are explained in the previous section. Each job has to be processed first at stage 1, then at stage 2 and so on. Each stage functions as a bank of parallel machines; at each stage job *j* requires only one machine, and usually, any machine can process any job (Pinedo, 1995).

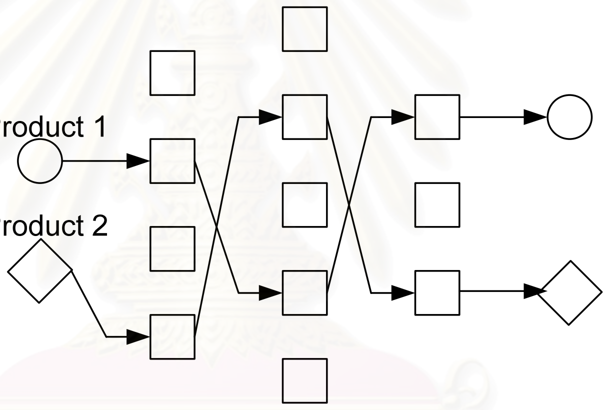


Figure 2.1 The flexible (hybrid) flow shop environment

In the most general problem in a setting of a flexible flow shop environment, there are multiple stages ($t \in \{1, \ldots, k\}$, $k \geq 2$) in series. At least one production stage is made up of m^t parallel machines ($m^t > 1$; $\exists\ t$). Each job has to be processed in stage 1 through stage *k*, in that order. A simple schematic representation of the flexible flow shop environment is illustrated in Figure 2.1.

The parallel machines in each stage may be identical, uniform, or unrelated parallel machines. Under identical parallel machines, the processing time to process a job on each machine in the same stage is similar. Machines are uniform if the processing time to process a job on any machine in the same stage is a constant ratio of its processing time on any other machine (*i.e.* new machines versus old 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 not only particular relationship of its processing time on any other machine, but the time also depends on the job that are processed on such a machine. The flow of jobs through the shop is unidirectional (Linn and Zhang, 1999). The survey of the flexible flow shop problem is presented in the next section.

2.3 Survey of the Flexible Flow Shop Problem

Only in the last decade or so, with the rapidly growing interest in the field of flexible manufacturing systems, the flexible flow shop problem has recently received attention because of its importance from both theoretical and practical points of view (Jin, Yang, and Ito, 2006). Flexible flow shop scheduling problems with the makespan criterion are NP-hard as Gupta has showed in 1988. He proves that the problem is NP-hard when, at any stage, there exist more than one processor. Consequently, the flexible flow shop problem is difficult and is well worth studying. In this section, the survey of the flexible flow shop problem are divided into four subsections, namely the flexible flow shop environment review, the setup time scheduling problem review, the scheduling objective function review, and the flexible flow shop scheduling procedure review. At the end of this section, the summary of the literature in the flexible flow shop problem is presented in Table 2.1.

2.3.1 Flexible Flow Shop Environment Review

Firstly, the review of the flexible flow shop environments is provided. As seen in Figure 2.2, the flexible flow shop environments are divided into two groups, namely 2-stage cases and k –stage ($k > 2$) cases. In earlier studies, the simple two-stage flexible flow shop environments have been considered; for example, Gupta (1988) has studied a two-stage flexible flow shop problem when there are multiple machines at stage 1 ($m^1 \geq 2$), and there is only one machine at stage 2 ($m^2=1$). Sriskandarajah and Sethi (1989) has considered the two-stage flexible flow shop problem with $m^1 \geq 2$ and $m^2=1$ and with $m^1 \geq 2$ and $m^2 \geq 2$. Few papers have been published on the k-stage flexible flow shop scheduling problem (Grangeon, Tanguy, and Tchernev, 1999), *see e.g.* Salvador (1973), Brah and Hunsucker (1991), and Rajendran and Chaudhuri (1992).

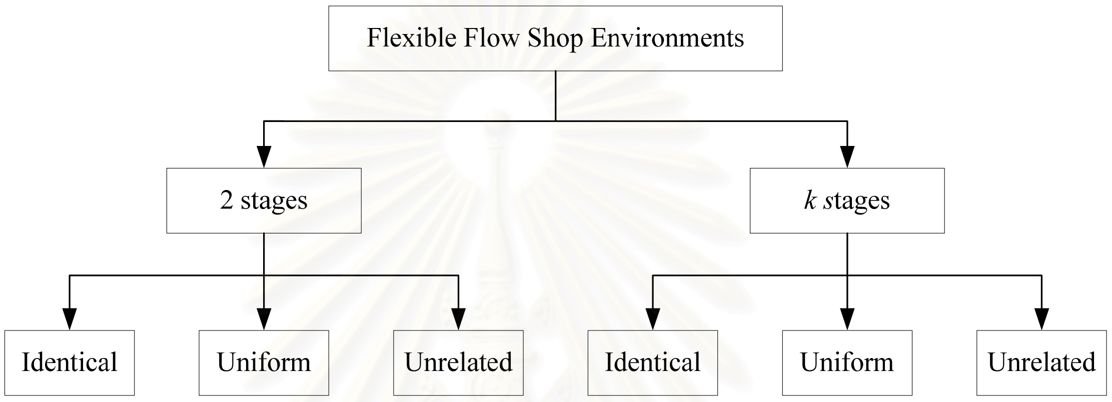


Figure 2.2 Classification of the flexible flow shop environments

For both the 2-stage and k-stage cases, the problems can still be divided into three groups, namely identical parallel machines, uniform parallel machines, and unrelated parallel machines (*see* Section 1.5.1.1). Certainly, the earlier literature has studied the flexible flow shop environment with identical parallel machines; for example, Gupta (1988) has studied the two-stage flexible flow shop problem with identical parallel machines at stage 1. Sriskandarajah and Sethi (1989) have concerned the two-stage flexible flow shop problem with identical parallel machines for both stages. Other two-stage flexible flow shop problems with identical parallel machines are, for example, found in Deal and Hunsucker (1991), Gupta and Tunc (1991, 1994), Guinet, Echalier, and Dussauchoy (1992), Uetake, Tsubone, and Ohba (1995), Kim, Kang, and Lee (1997), Lin and Liao (2003), Guirchoun *et al.* (2005), and Haouari *et al.* (2006). Soewandi and Elmaghraby (2003) study the two-stage flexible flow shop with uniform parallel machines. Low *et al.* (2008) propose the heuristics to solve the two-stage flexible flow shop problem with unrelated parallel machines.

For the *k*-stage ($k > 2$) flexible flow shop problem, most researchers have studied in the case of identical parallel machines, *see e.g.* Brah and Hunsucker (1991), Santos *et al.* (1995, 1996), Negenman (2001), Ruiz and Maroto (2006), and Janiak *et al.* (2007). Little literature has studied in the field of the flexible flow shop problem with uniform parallel machines, *see* Kyparisis and Koulamas (2006). Very few studies have considered the flexible flow shop problem with unrelated parallel machines, *see* Low (2005), Jenabi *et al.* (2007), and Ruiz, Şerifoğlu, and Urlings (2008).

In this dissertation, the problem under consideration is in the case of the *k*-stage problem with unrelated parallel machines, where is the general environment form of all cases of the flexible flow shop environments.

2.3.2 Setup Time Scheduling Problem Review

Secondly, this section provides the setup time flexible flow shop scheduling problem review. The classification of the setup time scheduling problems is shown in Figure 2.3.

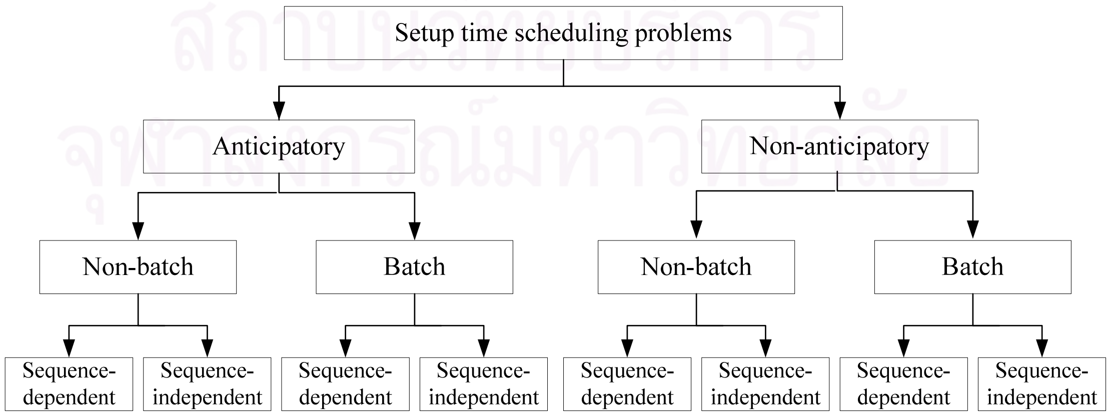


Figure 2.3 Classification of the setup time scheduling problems

Most earlier literature has not concerned the setup time in the flexible flow shop problems, *see e.g.* Deal and Hunsucker (1991), Lee and Vairaktarakis (1994), Santos *et al.* (1995, 1996), Moursli and Pochet (2000), and Sawik (2000, 2002, 2007).

For the literature considering the setup time, it can be divided into two groups, the anticipatory setup cases and the non-anticipatory setup cases. When setup time is separable from the processing time, it could be anticipatory, meaning that the setup of the next job can start as soon as a machine becomes free to process the job since the shop floor control system can identify the next job in the sequence. In such a situation, the idle time of a machine can be used to complete the setup of a job on a specific machine, *see* Ruiz *et al.* (2008). In another situation, setup time is non-anticipatory, that is, the setup operation can start only when the job arrives at a machine as the setup is attached to the job. Most researchers have assumed their setup time as non-anticipatory setup time, *see e.g.* Zandieh, Fatemi Ghomi, and Moattar Husseini (2006), Jenabi *et al.* (2007), and Ruiz *et al.* (2008).

For both anticipatory and non-anticipatory setup cases, the setup time cases can still be divided into two groups: non-batch cases and batch cases. In a non-batch processing environment, a setup time (cost) is incurred prior to the processing of each job, *see* Lin and Liao (2003) and Zandieh *et al.* (2006). For a batch setup processing environment, a setup time (cost) occurs when jobs, *e.g.* machine parts, are processed in batches (pallets, containers, or boxes), and a setup of a certain time or cost precedes the processing of each batch. The definition of a batch is as follows. The jobs are supposed to be partitioned into F, $F \geq 1$, *families*. A batch is a set of jobs of the same family (Allahverdi *et al.*, 2008), *see* Logendran, deSzoeke, and Barnard (2006).

The batch setup time (cost) can be sequence-dependent setup between consecutive families if its duration (cost) depends on the families of both the current and the immediately preceding batches or between two consecutive jobs in the family if its duration (cost) depends on the current job and the immediately preceding job in the family. Likewise, the non-batch setup time can also be sequence-dependent setup if the setup time depended on the current job and the immediately preceding job. For the sequence-independent setup time, the duration (cost) depends only on the current job or the current batch to be processed, so all setup times are assumed to be zero by adding the setup times to the processing times of their jobs or families.

For the non-batch sequence-independent setup time case, *see e.g.* Allaoui and Artiba (2004), and Low (2005). For the non-batch sequence-dependent setup time case, *see e.g.* Kurz and Askin (2003 and 2004), Ruiz and Maroto (2006), and Ruiz *et al.* (2008). In this dissertation, the non-bath sequence-dependent setup time problem is considered.

2.3.3 Scheduling Objective function Review

Majority of studies for the flexible flow shop scheduling problem has been considered to the optimization of a single criterion focusing on the minimum makespan problem, *see e.g.* Brah and Hunsucker (1991), Santos *et al.* (1995, 1996), Engin and Döyen (2004), and Logendran *et al.* (2006). However, there exist other several objectives such as minimizing total flow time, *see* Rajendran and Chaudhuri (1992) and Low (2005), minimizing maximum lateness, *see* Botta-Genoulaz (2000), minimizing weighted maximal tardiness, *see* Lin and Liao (2003), or minimizing total setup time, *see* Liu and Chang (2000).

However, scheduling problems often involve more than one aspect and therefore require multiple criteria analysis (Loukil, Teghem, and Tuyttens, 2005). Despite their importance, scant attention has been given to multiple criteria scheduling problems, *see* Paul (1979) and Gupta *et al.* (2002). In the literature concerning multi-objective scheduling problems, the five main approaches for the multi-objective can be distinguished as follows (Loukil *et al.*, 2005):

(a) *Hierarchical approach*: the objectives considered are ranked in a priority order and optimized in this order; for example, Sawik (2007) proposes a hierarchical approach to complete all the jobs with minimum number of tardy orders as a primary criterion and to level the aggregate production or the total capacity utilization over a planning horizon as a secondary criterion,

(b) *Utility approach*: a utility function or weighting function—often a weighted linear combination of the objectives—is used to aggregate the considered objectives in a single one, *see* Gupta *et al.* (2002). The problem considered in this dissertation belongs to this class,

(c) *Goal programming* (or *satisficing approach*): all the objectives are taken into account as constraints which express some satisficing levels (or goals), and the objective is to find a solution which provides a value as close as possible of the pre-defined goal for each objective. Sometimes one objective is chosen as the main objective and is optimized under the constraint related to other objectives, *see* Markland, Darby-Dowman, and Minor (1990),

(d) *Simultaneous* (or *Pareto*) *approach*: the aim is to generate—or to approximate in case of a heuristic method—the complete set of efficient solutions, *see* Mansouri (2006), and

(e) *Interactive approach*: at each step of the procedure, the decision-maker expresses his preferences in regard to one (or several) solutions proposed so that the method will progressively converge to a satisfying compromise among the considered objectives, *see* Bernardo and Lin (1994).

Each approach has its own advantages and drawbacks as described in general literature on multi-objective optimization; for example, the hierarchical approach, utility approach, and goal programming approach require more parameters or the priori information. The hierarchical approach and utility approach are more pragmatic, but they are unable to generate some efficient solutions. The simultaneous (or pareto) approach is more general or theoretical. The goal programming approach and interactive approach are more oriented to real case studies.

2.3.4 Flexible Flow Shop Scheduling Procedure Review

In this section, the flexible flow shop scheduling procedures are reviewed. Figure 2.4 shows the classification of the flexible flow shop scheduling procedures that are adapted from Winston and Venkataramanan (2003), Wang (2005), and Quadt and Kuhn (2007).

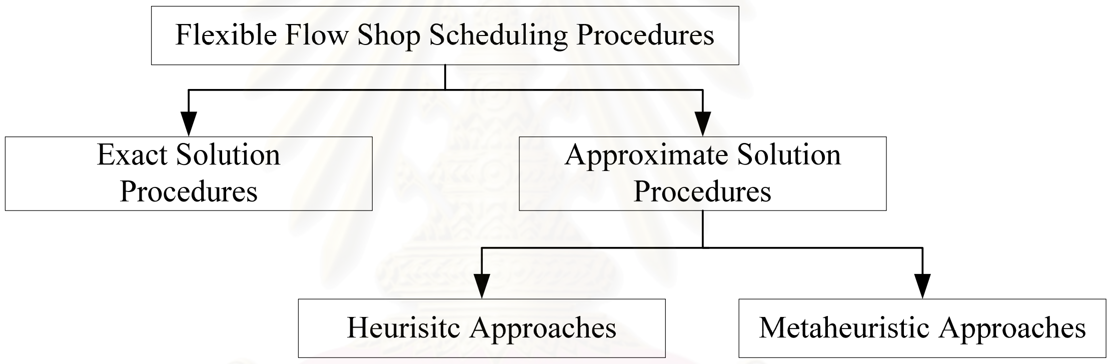

Flexible Flow Shop Scheduling Procedures
Exact Solution Procedures
Approximate Solution Procedures
Heurisitc Approaches
Metaheuristic Approaches

Figure 2.4 Classification of the flexible flow shop procedures

2.3.4.1 Exact Solution Procedures

The majority of optimal procedures for scheduling flexible flow lines are based on a branch and bound approach. Salvador (1973) is among the first to consider a flexible flow shop scheduling problem by using a branch and bound method, which guarantees optimal solutions. His procedure generates a permutation schedule, *i.e.* the same sequence of jobs is used on all stages. A permutation schedule simplifies the problem substantially, as only one sequence of jobs has to be determined. In Brah and Hunsucker (1991), a new branch and bound method, which is able to generate non-permutation schedules with machine idle times between jobs, is developed to minimize the makespan of a flexible flow shop problem. They have adapted the “Enumeration Method” proposed by Bratley *et al.* (1975) for scheduling the single stage parallel machine problem. Their algorithm consists of three parts: lower bound calculation, branching, and node elimination. Their algorithm is very fast for very small instances but already time consuming for small ones. However, the computation time of the suggested procedure is rather long when the problem size increases. They solved a 6-job configuration with 5 stages; all stages are made up of two machines except the middle stage which was made up of 3 machines, in 12 hours running time on a PC XT computer.

Rajendran and Chaudhuri (1992) consider the minimizing total flow time by using a branch and bound algorithm for obtaining a permutation schedule for the flexible flow shop environment. They solve the problem by generating a single sequence of jobs that is valid for all production stages. Portmann *et al.* (1998) also study the flexible flow shop scheduling problem to minimize the makespan. They improve the lower bounds of Brah and Hunsucker (1991) and reduce the number of branches used in the search tree by coupling it with a genetic algorithm (GA), which is employed to derive upper bounds (*i.e.* schedules) during the branch and bound procedure. Their computational experiments is indicated that optimal solutions using their branch and bound approach are 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 one through five, respectively) and 15 jobs with an average deviation of 3% from the results of the branch and bound algorithm.

Moursli and Pochet (2000) use a branching scheme that is an extension of a method for a parallel machine scheduling problem. The schedule is generated in one stage at a time. An approach based on a branch-and-bound approach coupled with constraint propagation is presented. Unlike the above mentioned procedures, it considers the stages simultaneously. The search tree is generated by consecutively selecting a stage and the next job to be scheduled on that stage. Néron, Baptiste, and Gupta (2001) use the so-called concepts of ‘energetic reasoning’ and ‘global operations’ to reduce the computation time of the procedure.

Sawik (2000, 2002) addressed his problems by formulating the flexible flow shop problems with limited buffers in a mixed integer programming format and solving the problems with a CPLEX solver. The mixed integer programming is to minimize makespan for both batch and non-batch scheduling. Of course, only small problems can be solved in this way to get optimal solutions.

Research on optimal procedures with objectives other than makespan minimization is sparse. Again, mostly a branch-and-bound method is employed. Rajendran and Chaudhuri (1992) consider the objective of minimizing mean flow time. They develop a branch-and-bound procedure that generates an optimal permutation schedule.

Azizoğlu, Cakmak, and Kondakci (2001) consider the mean flow time flexible flow shop problem but allow non-permutation schedules. They compare the branching scheme of Brah and Hunsucker (1991) with one that does not generate certain sub-problems because of proven sub-optimality under the mean flow time objective. With the new branching scheme, the algorithm has a substantially lower computation time. A different approach is considered by Harjunkoski and Grossmann (2002). They develop a method that generates an optimal solution by iteratively solving the loading and the sequencing problem consecutively. The objective is to minimize job assignment costs and one-time machine-initialization costs. Setup times are included, but are only dependent on the machine and not on the job. Thus, batching decisions are not made. The algorithm is based on the hybrid mixed integer and constraint programming approach. The loading problem is solved using mixed integer programming. A subsequent constraint programming procedure solves the sequencing part and iteratively adds cuts to the loading problem until an optimal schedule is found.

2.3.4.2 Approximate Solution Procedures

Even though the exact solution approach as stated in the previous section gives the optimal solution, it is impractical for the large-sized problems since the solution time would be unacceptably long. That is why an approximate solution technique is needed. Approximate solution techniques may provide the optimal solution for a problem even though they do not guarantee it. This fact implies that there may be some space for a heuristic to be improved. In this section, the review of the approximate solution procedures is divided into two groups: heuristic approaches and metaheuristic approaches.

The former, heuristic approaches, is first reviewed. Paul (1979) demonstrates that some scheduling theory results can probably be extended to the real-life problem of scheduling the parallel production lines in the glass-container industry. By means of computer simulation, the results indicate that the Shortest processing time (SPT) rule is relatively effective in this special case of scheduling machines in parallel subject to the resource constraints. Wittrock (1985) develops a periodic heuristic algorithm for maximizing the throughput (or minimizing the makespan) by focusing on job loading and time allocating. He also presents a more flexible non-periodic heuristic algorithm for the same problem by taking three steps: machine allocation, job sequencing, and timing (Wittrock, 1988).

Gupta (1988) has studied the minimum makespan problem in a heuristic algorithm for a two-stage flow shop problem. He proposes a heuristic to solve the minimum makespan problem. Computational experiments show that the effectiveness of the proposed heuristic increases as the problem size increases. Sriskandarajah and Sethi (1989) develop simple heuristic algorithms for two-stage flexible flow shop problem. They discuss the worst and average case performance of algorithms of finding minimum makespan schedules; their solutions are based on Johnson’s rule. Deal and Hunsucker (1991) study the two-stage flow shop scheduling problem. A lower bound calculation for the makespan is introduced, and the performance of a Johnson's type ordering is evaluated.

Gupta and Tunc (1991) develop two polynomial bounded heuristic algorithms to determine an acceptable solution to minimize the makespan for two-stage flexible flow shop problem. Their results show that when the number of machines at stage two is greater than or equal to 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 two, they develop two heuristics to minimize the makespan. Their first heuristic based on the premise that Johnson's rule coupled with an appropriate assignment rule should produce an acceptable schedule for the problem. In their second heuristic, they arrange jobs in a non-increasing order according to their processing times at stage two. Computational results indicate that the effectiveness of the algorithms increases with the increase of the total number of jobs. The deviations of the heuristic makespan are relatively large from the lower bounds, and an improved branch and bound algorithm is developed. The maximum number of jobs reported in their work is only eight jobs.

Guinet *et al.* (1992) propose a heuristic for the minimum makespan problem in a two-stage flexible flow shop based on Johnson's rule. They compare this heuristic with the Shortest Processing Time (SPT) and Longest Processing Time (LPT) dispatching rules. They conclude that the LPT rule gives good results for the minimum makespan problem in a two-stage hybrid flow shop environment.

Adler *et al.* (1993) describe the Bagpack Production Scheduling system (BPSS) whose machine environment is a three-stage flexible flow shop. The production process consists of three stages, but not all orders have to go through all the three stages. There are three objectives to minimize the sum of tardiness, the sum of setup times, and the work-in-process inventory. They identify the bottleneck stage, and they schedule the jobs at the bottleneck stage and schedule the jobs at the non-bottleneck stages, respectively. However, the procedure is very application-specific, and no computational experience is reported. Gupta and Tunc (1994) consider the two-stage flow shop scheduling problem. The setup and removal times of each job at each stage are separated from the processing times. They propose heuristic algorithms that are empirically tested to determine the effectiveness in finding an optimal.

Global lower bounds for the flow shop problem with multiple processors are proposed by Santos *et al.* (1995). In the absence of a known optimal solution for NP-complete problems, strong lower bounds can be effective tools when used to evaluate the quality of sub-optimal solution methodologies. For example, suppose a lower bound on the unknown optimal solution is developed for an NP-complete flexible flow shop scheduling problem. Further suppose that a feasible solution is found for this problem which lies within 5% of the lower bound. Obviously, it can be said that the solution obtains within 5% of the optimal solution. This allows the practitioner to decide if a possible improvement of at most 5% is worth the further expenditure of time and effort. Santos *et al.* (1996) investigate scheduling procedures which seek to minimize the makespan in the static flow shop with multiple processors scheduling environment. Their method is to generate an initial permutation schedule based on the Palmer, CDS, Gupta, and Dannenbring flow shop heuristics, and that would then be followed by the application of First in First out (FIFO).

Gupta *et al.* (1997) consider a non-preemptive two-stage hybrid flow shop problem. The objective is to find a schedule which minimizes the maximum completion time or makespan. Several lower bounds are derived and are tested in a branch and bound algorithm so as to limit the size of the search tree. They propose several heuristics, all based on Johnson’s algorithm. Kyparisis and Koulamas (2006) deal with the multistage flexible flow shop scheduling problem with uniform parallel machines in each stage and the objective of minimizing makespan. They develop several well-known heuristics to solve the problems.

The later, metaheuristic approaches, is reviewed. To obtain a near-optimal solution, the metaheuristic algorithms (or the artificial intelligent techniques) have also been proposed. The most well-known metaheuristic algorithms are the simulated annealing, tabu search, and genetic algorithms (Jones *et al.*, 2002). Gourgand, Grangeon, and Norre (1999) present several simulated annealing (SA)-based algorithms for the flexible flow shop problem. A specific neighborhood is used, and the authors apply the methods to a realistic industrial problem. Jin *et al.* (2006) consider the flexible flow shop with identical parallel machines. They propose two approaches to generate the initial job sequence and use a simulated annealing algorithm to improve it. It can be seen that a simulated annealing algorithm has been successfully applied to various combinatorial optimization problems. For an extensive survey of the theory and applications of the simulated annealing algorithm, *see* Koulamas, Antony, and Jaen (1994).

Furthermore, Nowicki and Smutnicki (1998) propose a tabu search (TS) algorithm for the flexible flow shop makespan problem. Logendran *et al.* (2006) tackle with a flexible flow scheduling problem within the context of sequence dependent setup times in shops. Their objective is to minimize the makespan on the shop floor. Three different algorithms based on tabu search are developed.

A genetic algorithm (GA) has been widely used in many previous works for the flowshop makespan problem, *see* Werner (1984) and Reeves (1995). Cheng, Gen, and Tozawa (1995) address the earliness/tardiness scheduling problem with identical parallel machines, and they apply a GA approach to solve this problem. Ruiz, Maroto, and Alcaraz (2005) use a GA approach to deal with the permutation flow shop scheduling problem with sequence-dependent setup times. For the flexible flow shop problem, *see* Bertel and Billaut (2004), Kurz and Askin (2004), Bolat, Al-Harkan, and Al-Harbi (2005), Ruiz and Maroto (2006), and Jenabi *et al.* (2007).

For other metaheuristics, for example, Engin and Döyen (2004) deal with the flexible flow shop with identical parallel machines by using the artificial immune system. Artificial immune system (AIS) is computational systems inspired by theoretical immunology, observed immune functions, principles and mechanisms in order to solve problems. Their objective is to minimize the makespan.

Table 2.1 The summary of the literature in the flexible flow shop problem

References	**# of stages**	**Parallel machines**	**Setup**	**Objectives**	**Approaches**
Salvador (1973)	*k*	Identical	-	Makespan	Branch-and-bound
Paul (1979)	*k*	Identical	-	Total tardiness, number of tardy jobs	Simulation
Wittrock (1985)	*k*	Identical	-	Makespan	Heuristics
Gupta (1988)	2	Identical	-	Makespan	Heuristics
Wittrock (1988)	*k*	Identical	-	Makespan, queuing	Heuristics
Sriskandarajah and Sethi (1989)	2	Identical	-	Makespan	Heuristics
Brah and Hunsucker (1991)	*k*	Identical	-	Makespan	Branch-and-bound
Deal and Hunsucker (1991)	2	Identical	-	Makespan	Heuristics
Gupta and Tunc (1991)	2	Identical	-	Makespan	Heuristics
Guinet, Echalier, and Dussauchoy (1992)	2	Identical	-	Makespan	Heuristics, mixed integer programming
Rajendran and Chaudhuri (1992)	*k*	Identical	-	Mean flow time	Branch-and-bound

Table 2.1 The summary of the literature in the flexible flow shop problem (cont.)

References	**# of stages**	**Parallel machines**	**Setup**	**Objectives**	**Approaches**
Adler, Fraiman, Kobacker, Pinedo, Plotnicoff, and Wu (1993)	3	Identical	Yes	Tardiness, setup time, work-in-process inventory	Heuristics
Gupta and Tunc (1994)	2	Identical	Yes	Makespan	Heuristics
Lee and Vairaktarakis (1994)	2	Identical	-	Makespan	Heuristics
Santos, Hunsucker, and Deal (1995)	*k*	Identical	-	Makespan	Lower bound
Uetake, Tsubone, and Ohba (1995)	2	Identical	-	Total flow time, makespan	Heuristics
Santos, Hunsucker, and Deal (1996)	*k*	Identical	-	Makespan	Heuristics
Gupta, Hariri, and Potts (1997)	2	Identical	-	Makespan	Branch-and-bound, heuristics
Haouari and M'Hallah (1997)	2	Identical	-	Makespan	Simulated annealing, tabu search
Kim, Kang, and Lee (1997)	2	Identical	Yes	Makespan	Heuristics
Nowicki and Smutnicki (1998)	*k*	Identical	-	Makespan	Tabu search
Portmann, Vignier, Dardilhac, and Dezalay (1998)	*k*	Identical	-	Makespan	Branch-and-bound
Riane, Artiba, and Elmaghraby (1998)	3	Identical	-	Makespan	Branch-and-bound

Table 2.1 The summary of the literature in the flexible flow shop problem (cont.)

References	# of stages	Parallel machines	Setup	Objectives	Approaches
Brah and Loo (1999)	*k*	Identical	-	Makespan, mean flow time	Heuristics
Gourgand, Grangeon, and Norre (1999)	*k*	Identical	-	Makespan	Simulated annealing
Botta-Genoulaz (2000)	*k*	Identical	-	Maximum lateness	Heuristics
Liu and Chang (2000)	*k*	Identical	Yes	Setup time	Lagrangian relaxation
Moursli and Pochet (2000)	*k*	Identical	-	Makespan	Branch-and-bound
Sawik (2000)	*k*	Identical	-	Makespan	Mixed integer programming
Azizoğlu, Cakmak, and Kondakci (2001)	*k*	Identical	-	Mean flow time	Branch-and-bound
Negenman (2001)	*k*	Identical	-	Makespan	Simulated annealing, tabu search
Néron, Baptiste, and Gupta (2001)	*k*	Identical	-	Makespan	Branch-and-bound
Soewandi and Elmaghraby (2001)	3	Identical	-	Makespan	Heuristics
Gupta, Krüger, Lauff, Werner, and Sotskov (2002)	*k*	Identical	-	Earliness and tardiness penalties, weighted completion time, and the costs of due date assignment.	Heuristics
Harjunkoski and Grossmann (2002)	*k*	Identical	-	Assignment and initialization costs	Mixed integer programming
Sawik (2002)	*k*	Identical	-	Makespan	Mixed integer programming

Table 2.1 The summary of the literature in the flexible flow shop problem (cont.)

References	# of stages	Parallel machines	Setup	Objectives	Approaches
Alisantoso, Khoo, and Jiang (2003)	*k*	Identical	-	Makespan	Immune algorithm
Kurz and Askin (2003)	*k*	Identical	Yes	Makespan	Heuristics
Lin and Liao (2003)	2	Identical	Yes	weighted maximal tardiness	Heuristics
Soewandi and Elmaghraby (2003)	2	Uniform	-	Makespan	Heuristics
Wang and Hunsucker (2003)	*k*	Identical	-	Makespan	Heuristics
Allaoui and Artiba (2004)	*k*	Identical	-	Makespan, maximum tardiness, flow time, number of tardy jobs	Heuristics, simulated annealing
Bertel and Billaut (2004)	*k*	Identical	-	Weighted number of tardy jobs	Greedy algorithm, genetic algorithm
Engin and Döyen (2004)	*k*	Identical	-	Makespan	Immune algorithm
Kurz and Askin (2004)	*k*	Identical	Yes	Makespan	Greedy algorithm, genetic algorithm
Wardono and Fathi (2004)	*k*	Identical	-	Makespan	Tabu search
Bolat, Al-Harkan, and Al-Harbi (2005)	3	Identical	-	Makespan	Branch-and-bound, genetic algorithm
Guirchoun, Martineau, and Billaut (2005)	2	Identical	-	Total completion time	Mixed integer programming
Low (2005)	*k*	Unrelated	-	Makespan	Simulated annealing
Haouari, Hidri, and Gharbi (2006)	2	Identical	-	Makespan	Branch-and-bound

Table 2.1 The summary of the literature in the flexible flow shop problem (cont.)

References	# of stages	Parallel machines	Setup	Objectives	Approaches
Jin, Yang, and Ito (2006)	*k*	Identical	-	Makespan	Simulated annealing
Kyparisis and Koulamas (2006)	*k*	Uniform	-	Makespan	Heuristics
Logendran, deSzoeke, and Barnard (2006)	*k*	Identical	Yes	Makespan	Tabu search
Ruiz and Maroto (2006)	*k*	Identical	Yes	Makespan	Genetic algorithm
Zandieh, Fatemi Ghomi, and Moattar Husseini (2006)	*k*	Identical	Yes	Makespan	Immune algorithm
Janiak, Kozan, Lichtenstein, and Oğuz (2007)	*k*	Identical	-	Total weighted earliness, total weighted tardiness, total weighted waiting time	Tabu search, simulated annealing
Jenabi, Fatemi Ghomi, Torabi, and Karimi (2007)	*k*	Unrelated	Yes	Setup costs, inventory holding costs	Genetic algorithm, simulated annealing
Sawik (2007)	*k*	Identical	-	Number of tardy jobs, the total capacity utilization	Mixed integer programming
Low, Hsu, and Su (2008)	2	Unrelated	-	Makespan	Heuristics
Ruiz, Şerifoğlu, and Urlings (2008)	*k*	Unrelated	Yes	Makespan	Mixed integer programming, heuristics

CHAPTER III

A MATHEMATICAL PROGRAMMING SOLUTION APPORACH

This chapter provides the mathematical programming formulation for scheduling the flexible flow shop problem with unrelated parallel machines that minimizes both objective functions of makespan (C_{max}) and number of tardy jobs (η_T). The model formulation considers three main conditions, namely unrelated parallel machines, sequence-dependent setup time between two consecutive jobs, and machine-dependent setup time of a job. The formulation can be used to find the optimal schedule by using commercially available software for the mixed integer programming. A numerical example is also illustrated. The example problem has been modeled by using an advanced algebraic modeling language AMPL with CPLEX solver that runs on Windows platform. AMPL enables the complete separation to be kept between the model file and the data file, and the model is written in a form very close to the mathematical formulation (Sawik, 2000, 2002, 2007).

This chapter is organized as follows: Firstly, the introduction is explained to give the definition and importance of the mathematical programming. Secondly, the problem under consideration is described. Thirdly, a descriptive example is used to explain the problem under consideration. The 0-1 mixed integer programming formulation is presented in the next section, which consists of the assumptions, objective function, and constraints as well as a numerical example that is presented to illustrate the application of the proposed model. Then, the total number of possible sequence combinations that are generated by a complete enumeration method is estimated to illustrate the complexity of the problem. Finally, a conclusion will be presented.

3.1 Introduction

Two fundamental decisions of a scheduling problem are allocation and sequencing decisions (Baker, 1974). The former answers the question what machine should be allocated to which job, and the latter answers the question how to sequence the jobs. The aim of these decisions is to obtain a best schedule that satisfies some measures of effectiveness (such as minimization of the makespan, mean flow time, tardiness, or number of tardy jobs), called the model's *objective function*(*s*), and satisfies the production constraints (such as production requirement, resource capacities, or operation procedures), called *constraint functions*. The variables whose values are able to control and influence the performance of the system are called *decision variables*. The scientific approach to decision making usually involves the use of one or more mathematical models. A mathematical model is a mathematical representation of an actual situation that may be used to make better decisions or simply to understand the actual situation better (Winston, 2004).

In recent years, most researches have been reported for the mathematical model; for example, Sawik (2000) presents a mixed integer programming formulation for scheduling a flexible flow line with the finite intermediate buffers and uses the commercial software to solve the problem. Sawik (2002) still presents a mixed integer programming model for the same problem in 2000, but all parts are scheduled in batches of parts of the same type, and within the batch individual parts are processed consecutively part-by-part. Damodarn and Srihari (2004) propose a mixed integer programming formulation for the flow shop problem with no buffers. Tang and Liu (2007) present a mixed integer programming model for a real-life order scheduling problem for the production of steel sheets. Bhattacharya and Bose (2007) develop the mathematical model for scheduling the continuous processing units and test the model by using the commercial software.

In general, mathematical programming models ensure the optimal schedules, but the CPU time required to find proven optimal schedules for realistic large-sized problems still can be very high. However, the mathematical models are worth formulating because the proven optimal solutions obtained for small-sized problems can also be used to evaluate the performance of various heuristics that are developed to find the approximate solution (Sawik, 2000, 2002, 2007; Lee and Asllani, 2004). Moreover, the development of the computer devices is improving with a fast pace, in which they are developed with larger memories and may be able to solve practical problems in near future. In addition, the mathematical model can be used to understand the actual situation better (Winston, 2004).

3.2 Problem Description

In this dissertation, the flexible flow shop system consists of *k* stages in series, as shown in Figure 3.1. Each stage *t* (*t* = 1, …, *k*) is made up of m^t unrelated parallel machines. Each job *j* (*j* = 1,…, *n*) must be processed without preemption on exactly one machine in each of the stages sequentially, that is, each job must be processed in stage 1 through stage *k* in that order. There are infinite buffers between all adjacent stages as well as before the first stage and after the last stage. The order of processing the jobs in every stage is identical, that is, all jobs have the same routing and do not revisit a stage the jobs have already visited.

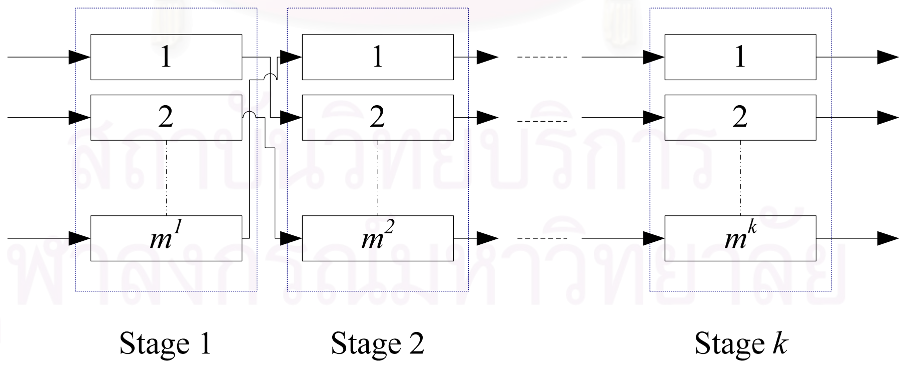


Figure 3.1 The flexible flow shop environment

At the beginning of a current planning period, there are a fixed number of jobs that must be ordered to be processed on this flexible flow shop system so that the makespan and the number of tardy jobs are minimized. It is presumed that the schedule is static in that the decision variables do not involve sequences of decisions over multiple planning periods, that is, the current schedule will not change even though there are new jobs that enter the system. Those new jobs will be scheduled in the following planning period. This implies that it is possible that there are jobs scheduled in the previous planning period that are not yet finished; they must be processed as scheduled before the jobs that are scheduled in the current planning period.

Let ps_j^t be the standard processing time for job *j* at stage *t*, which is usually measured on the average processor. Their actual processing times of jobs may be lesser than the standard processing times, when jobs operate at a higher efficiency machine, and the actual processing times may be greater than the standard processing times, when jobs operate at the lower efficiency machines.

Let v_{ij}^t be the relative speed of machine *i* on which job *j* is processed at stage *t*, which is a relative machine speed that is compared to the average speed machine in the system. The value of a relative machine speed of job *j* on machine *i* (v_{ij}^t) is equal to 1, when the machine *i* that processes job *j* is an average efficiency machine for the job *j* in the system. The v_{ij}^t value is lesser than 1, when the machine *i* that processes job *j* is the lower efficiency machine for the job *j*, whereas such a value is greater than 1 if the machine *i* that processes job *j* is the higher efficiency machine for the job *j*. In this dissertation, it is assumed that the actual processing time of job *j* on machine *i* at stage *t* is equal to the standard processing time of job *j* at stage *t*, whenever the relative machine speed of job *j* on machine *i* at stage *t* is equal 1. Hence, the processing time of job *j* that must be processed on machine *i* at stage *t* is equal to ps_j^t / v_{ij}^t.

Let s_{lj}^t be the "sequence-dependent" setup time between job *l* and job *j*, where job *j* is to be processed directly after job *l* on the same machine at stage *t*. The sequence-dependent setup time is the time needed on any machine to install, clean, or remove something before the next job is processed, which the length of setup required depends on the job just completed and on the one about to be started. Thus, the total operating time of job *j* (O_j^t) that is processed directly after job *l* on machine *i* at stage *t* is equal to $s_{lj}^t + ps_j^t / v_{ij}^t$.

Moreover, it is possible that there are some unfinished jobs on each machine from the previous planning period, so to take into account of the sequence-dependent setup time between a unfinished job in the previous planning period and a new job in the current planning period, the machine-dependent setup time of job *j* (ch_{ij}^t) that is processed on machine *i* in the first position at stage *t* is used in this research, this is, the total operating time of job *j* (O_j^t) that is processed on machine *i* in the first position at stage *t* is equal to $ch_{ij}^t + ps_j^t / v_{ij}^t$. To avoid the confusion between the words of sequence-dependent setup time and machine-dependent setup time, in this dissertation, the sequence-dependent setup time is called "*setup time*" and the machine-dependent setup time is called "*changeover time*".

Also, the unfinished jobs still affect the availability of machine *i* at stage *t*, a_i^t, which is the time that the pervious jobs are still processed on each machine. In other words, job *j* that is ready in the system to be assigned on machine *i* in stage *t* cannot start its processing before the machine availability. Moreover, each job *j* still has two information, a release date (r_j) that is the time the job will arrive in the system (in other words, job *j* cannot start its processing before its release date) and a due date (d_j) that is the time the job is promised to the customer.

Machines are available from time non-negative onwards, without breakdowns and scheduled or unscheduled maintenance. At any time, every machine can process at most one job. Each job is always processed at most one machine at any time and cannot be interrupted during its processing. Each job does not require any other job to be completed before such a job is allowed to start its processing. Each job cannot be split. Jobs are always processed without error and cannot be cancelled before their completion.

3.3 A Descriptive Example

Now suppose that there is a two-stage flexible flow shop environment, which consists of two unrelated parallel machines in stage 1 and one machine in stage 2. Two jobs will be scheduled in such an environment. The standard processing time (ps_j^t), release dates (r_j), and due dates (d_j) of two jobs are given in Table 3.1. The relative machine speed of machine i at stage t (v_{ij}^t) is listed in Table 3.2. The changeover time (ch_{ij}^t) and setup time (s_{ij}^t) are given in Table 3.3 and Table 3.4 respectively. The machine availability (a_i^t) is shown in Table 3.5.

Table 3.1 The standard processing time (ps_j^t), release dates (r_j), and due dates (d_j) for job j

job j	1	2
ps_j^1	5	7
ps_j^2	7	4
r_j	1	2
d_j	20	22

Table 3.2 The relative machine speed of job j on machine i at stage t (v_{ij}^t)

	job j	1	2
Stage t	machine i		
1	1	1.25	1.00
	2	1.00	0.70
2	1	1.00	1.00

Table 3.3 The matrix of changeover time of job j on machine i at stage t (ch_{ij}^t)

	job j	1	2
Stage t	machine i		
1	1	3	1
	2	1	2
2	1	2	1

Table 3.4 The matrix of setup time from job *l* to job *j* at stage *t* (s_{lj}^{t})

	to job *j*	1	2
Stage *t*	from job *l*		
1	1	-	2
	2	1	-
2	1	-	1
	2	3	-

Table 3.5 The availability of machine *i* at stage *t* (a_i^t)

	m_1^1	m_2^1	m_1^2
a_i^t	2	1	7

For this problem, each job has to be processed at stage 1 first and then at stage 2, and at each stage job *j* requires only one machine. This assumes that job 1 is processed on machine 1 (m_1^1) and job 2 is processed on machine 2 (m_2^1) at stage 1. Then, at stage 2, job 1 is processed on the machine (m_1^2) at stage 2 before job 2 is processed on the same machine (m_1^2). The schedule can be generated as follows: Firstly, although job 1 arrives at the system at unit time 1 (*see* r_1 = 1), job 1 starts its processing at unit time 2, because there are some unfinished jobs from the previous planning period on machine 1 at stage 1 (*see* a_1^1 = 2). Before job 1 starts its processing, the changeover (*see* ch_{11}^1) such as clean-up process is hold. Hence, job 1 can start at unit time 5 ($a_1^1 + ch_{11}^1$, a_1^1 = 2 and ch_{11}^1 =3). Due to the unrelated problem, the processing time of job 1 is equal to 4 time units (ps_1^1/v_{11}^1, ps_1^1 = 5 and v_{11}^1 = 1.25), so the completion time of job 1 at stage 1 is unit time 9. Let C_j^t be the completion time of job *j* at stage *t*, so C_1^1 = 9. Similarly, job 2 arrives at the system at unit time 2 (max{ r_2 , a_2^1 }, r_2 = 2 and a_2^1 = 1). The changeover time is hold about 2 time units (*see* ch_{22}^1 = 2), so job 2 starts its processing at unit time 4 (max{ r_2 , a_2^1 } + ch_{22}^1, r_2 = 2, a_2^1 = 1, and ch_{22}^1 = 2). Again, due to the unrelated problem, the processing time of job 2 is equal to 10 time units (ps_2^1/v_{22}^1, ps_2^1 = 7 and v_{22}^1 = 0.70), so the completion time C_2^1 of job 2 at stage 1 is unit time 14 (*see* Figure 3.2).

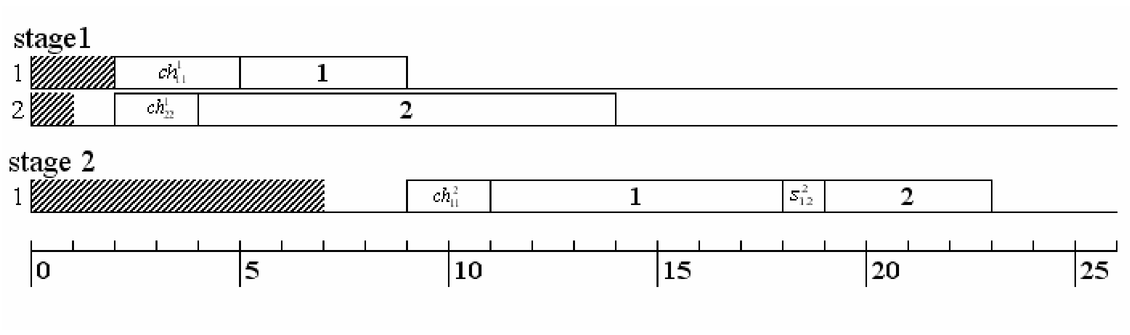


Figure 3.2. A Gantt chart for the example problem

For stage 2, the system has one machine at this stage, so only a sequencing function is considered. Now, job 1 is assumed to be processed at the first position of the machine at stage 2, but job 1 cannot start its processing before it finishes at stage 1, that is, job 1 at stage 2 can start after unit time 9 (*see* C_1^1 = 9). However, since job 1 is processed on the machine at the first position, it is able to start whenever all jobs from the previous planning period complete (*see* a_1^2). Hence, job 1 arrives at stage 2 at unit time 9 (max{C_1^1, a_2^1}, C_1^1 = 9 and a_2^1 = 7), and job 1 can start after the changeover process of job 1 on machine 1 at stage 2 (*see* ch_{11}^2), so it starts its processing at unit time 11 (max{C_1^1, a_2^1}$+ch_{11}^2$, C_1^1 = 9, a_2^1 = 7, and ch_{11}^2 = 2). Then, job 1 spends 7 time units (ps_1^2/v_{11}^2, ps_1^2 = 7 and v_{11}^2 = 1.00), and it completes at unit time 18 (=C_1^2). Next, job 2 is assigned to the same machine, and it arrives at stage 2 whenever it completes its processing at stage 1. Hence job 2 arrives at stage 2 at unit time 14 (*see* C_2^1), but it cannot start any process because at that time job 1 is still processed on the machine, so it must wait until unit time 18 (*see* C_1^2 = 18). At the point of unit time 18, before job 2 starts its processing, the setup time between job 1 and job 2, called *sequence-dependent setup time,* such as a cleaning process occurs and spends 1 time unit (*see* s_{12}^2 = 1). Thus job 2 starts its processing at unit time 19 (C_1^2 + s_{12}^2, C_1^2 = 18 and s_{12}^2 = 1). Then, job 2 spends 4 time units (ps_2^2/v_{12}^2, ps_2^2 = 4 and v_{12}^2 = 1.00), and it completes at unit time 23 (=C_2^2).

In this dissertation, two performance measures, which are the makespan and number of tardy jobs, are considered. The makespan is equivalent to the completion time of the last job to leave the system. In this example, job 2 is the last job to leave the system, so the makespan (C_{max}) is equal to 23 time units (*see* C_2^2 = 23). To consider the due dates compared to the completion time of both jobs, there are one job that is tardy (*i.e.* Job 2, d_2 = 22 unit time and C_2^2 = 23 unit time, completes after its due date, so job 2 is a tardy job). Thus, the number of tardy jobs (η_T) is equal to 1.

Table 3.6 Possible feasible schedule results of an example problem

Schedule No.	Schedule results	C_1^2	C_2^2	C_{max}	η_T
1	Stage 1: 1 – 2 // - Stage 2: 1 – 2	18	23	23	1
2	Stage 1: 1 – 2 // - Stage 2: 2 – 1	34	24	34	2
3	Stage 1: 2 – 1 // - Stage 2: 1 – 2	24	29	29	2
4	Stage 1: 2– 1 // - Stage 2: 2 – 1	30	20	30	1
5	Stage 1: 1 // 2 Stage 2: 1 – 2	18	23	23	1
6	Stage 1: 1 // 2 Stage 2: 2 – 1	29	19	29	1
7	Stage 1: 2 // 1 Stage 2: 1 – 2	16	21	21	0
8	Stage 1: 2 // 1 Stage 2: 2 – 1	22	12	22	1
9	Stage 1: – // 1 – 2 Stage 2: 1 – 2	16	21	21	0
10	Stage 1: – // 1 – 2 Stage 2: 2 – 1	34	24	34	2
11	Stage 1: – // 2 – 1 Stage 2: 1 – 2	29	34	34	2
12	Stage 1: – // 2 – 1 Stage 2: 2 – 1	29	19	29	1

However, the above detailed example is one of all 12 schedules that can be generated for a given example problem. In this dissertation, a result of a schedule can be presented by using a job code. For example, the result from Figure 3.2 is represented as stage 1: 1 // 2 and stage 2: 1 – 2, that is, at stage 1, job 1 is processed on machine 1 and job 2 is processed on machine 2, and at stage 2, job 1 is processed before job 2 on the same machine. For another example, 1 – 2 // -, it means that there are 2 machines in the stage where job 1 and job 2 are processed on machine 1 and no job is processed on machine 2. Table 3.6 shows all possible feasible schedules generated under such machine and job environments. It is found that only 2 schedules, a schedule No.7 and a schedule No.9, generate a good solution both criteria, C_{max} and η_T. However, if the problem size increases, it is impossible to generate all schedules since the number of schedules generated will exponentially rise (*see* Section 3.5).

3.4 Mathematical Formulation

This section presents a 0-1 mixed integer linear programming model with the objectives of minimizing both criteria, namely the makespan and the number of tardy jobs. Before proceeding with the mathematical formulation, the notations of parameters and decision variables used in formulating model are defined. The mathematical formulation is then provided.

Indices:

t	Stage index, t = 1, 2, 3, . . . , k
i	Machine index, i = 1, 2, 3, . . . , m^t
j, l	Job index, j, l = 1, 2, 3, . . . , n

Parameters:

r_j	Release date of job j
m^t	Number of parallel machines at stage t
d_j	Due date of job j

s_{lj}^{t}	Setup time between job *l* and job *j* at stage *t*
ch_{ij}^{t}	Changeover time of job *j* if job *j* is assigned to machine *i* at the first position at stage *t*
ps_{j}^{t}	Standard processing time of job *j* at stage *t*
v_{ij}^{t}	Relative machine speed of machine *i* at stage *t* for job *j*
a_{i}^{t}	Time when machine *i* at stage *t* becomes available

Variables:

X_{ijl}^{t}	1 if job *j* is scheduled immediately before job *l* on machine *i* at stage *t*, and 0 otherwise
O_{j}^{t}	Operating time of job *j* at stage *t*
C_{j}^{t}	Completion time of job *j* at stage *t*
C_{max}	The makespan is equivalent to the completion time of the last job to leave the system
U_j	A Boolean variable; 1 if job *j* is tardy, and 0 otherwise
T_j	Tardiness of job *j*
η_T	The total number of tardy jobs in the schedule

Before formulating the model, the assumptions of the model are first given. Then, the objective function of the optimization model is presented. Then, the given constraints are addressed. Finally, a numerical example is also illustrated.

3.4.1 Assumptions

The problem is formulated under the following assumptions:

1. The problem is assumed to be a static and deterministic scheduling environment,
2. Machines are available from time non-negative onwards, with no breakdowns and scheduled or unscheduled maintenance,
3. The ready times for all jobs are non-negative,

4. Non-anticipatory sequence-dependent setup times exist between two consecutive jobs at each stage. After completing processing of one job and before beginning processing of the next job, some sort of setup must be performed,

5. Machine-dependent setup times occur only when the job assigned to each machine at the first position at each stage in a period studied,

6. Job processing cannot be interrupted (*i.e.* no preemption is allowed) and jobs have no associated priority values,

7. There are no precedence constraints, that is, any job can be allowed to start without any other job completion condition,

8. Jobs are always processed without error and cannot be cancelled before their completion,

9. Each job visits the stages in same order but does not revisit a stage that it has already visited,

10. At any time, every job can be processed at most one machine, and every machine can process at most one job,

11. There are infinite buffers between all stages as well as before the first stage and after the last stage, and

12. There is no travel time between stages; jobs are available for processing at a stage immediately after departing at previous stage.

3.4.2 Objective Function

The need to consider multiple criteria in scheduling is widely recognized (Loukil *et al.,* 2005). Either a hierarchical (also called *Lexicographic*) or a simultaneous approach can be adopted (Gupta and Ruiz-Torres, 2005). Under a hierarchical or lexicographic approach, the criteria are ranked in order of importance; the first criterion is optimized first, the second criterion is then optimized, subject to achieving the optimum with respect to the first criterion and so on. For simultaneous optimization, there are two approaches. Firstly, a single objective function can be constructed by forming a linear combination of the various criteria, which is then optimized. Secondly, all efficient (also called *Pareto Optimal*) schedules can be generated, where an efficient schedule is one in which any improvement to the performance with respect to one of the criteria causes deterioration with respect to one of the other criteria.

For a linear combination of the various criteria, some researchers deal with the mutli-criteria by minimizing the positively weighted convex sum of multi-objectives in form $f(x) = w_1 \times f_1(x) + w_2 \times f_2(x) ..... + w_n \times f_n(x)$, where $\sum_{h=1}^{n} w_h = 1$ and $w_h \geq 0$, $h \in \{1,..., n\}$. It is used to aggregate the considered objectives in a single one. For example, Murtadi and Taboun (2001) study the bicriteria of the makespan and the number of tardy jobs for scheduling on identical parallel machines. They develop the solution techniques for multi-objective problems (MOP) to minimize the positively weighted convex sum of their objective in form $f(x) = w_1 \times f_1(x) + w_2 \times f_2(x)$, where $w_2 = (1 - w_1)$. Rajendran and Ziegler (2003) study a static flow shop problem to minimize the sum of flow time and tardiness of jobs by using the weighted sum of both objectives. This technique corresponds to weighted decision making, where the weight values are determined by the truth values. Lee and Asllani (2004) study the single machine problem with dual criteria. They minimize the flow time and the tardiness by using the weighted sum of both objectives. They choose the weight values by using the holding cost for job *j* per unit time in the flow time criteria and the tardiness cost for job *j* per unit time in tardiness criteria as the weight values.

For this dissertation, the motivation to consider the problem of generating an efficient set of schedules for the flexible flow shop environment comes from the variety of industrial cases, where the criteria related to efficiently utilizing resources and completing orders by their due date are important. In the case that if there are many optimal solutions from one criterion, considering the other criteria may assist to choose the best solution better than randomly selecting from them. Therefore, the makespan and the number of tardy jobs are used as the scheduling criteria.

The first objective

According to the dissertation objectives, the first goal is to minimize the makespan C_{max}, *i.e.* the completion time of the last job at the last stage.

Let the completion time of job *j* at stage *k*, it means the time that job *j* can leave the system, be C_j^k, then.

$$C_{max} = \max_{j \in \{1,\ldots,n\}} \{ C_j^k \} \tag{3.1}$$

The second objective

The second goal is to minimize the number of tardy jobs. Associated with each job *j* is a due date $d_j \geq 0$, let $U_j = 1$ if the completion time C_j^k of job *j* is larger than due date for job *j*, and otherwise $U_j = 0$. The total number of tardy jobs (η_T) is defined as:

$$\eta_T = \sum_{j=1}^{n} U_j \tag{3.2}$$

A simplest method to combine multiple objective functions into a scalar fitness solution (Z_1) is the following weighted sum approach:

$$Z_1 = w_1 \times C_{max} + w_2 \times \eta_T \tag{3.3}$$

where w_1 and w_2 are constant weights of each objective criterion. The constant weight w_1 may represent the holding cost per time unit, whereas the constant weight w_2 may represent the penalty cost per tardy-job unit.

For example, it can also apply a single-objective function, where the weights w_1 and w_2 are fixed as follows (Murata, Ishibuchi, and Tanaka, 1996), *e.g.* w_1: w_2 = 100:1, 50:1, 20:1, 15:1, 10:1, 5:1, 2:1, 1:1, 1:2, 1:5, 1:10, 1:15, 1:20, 1:50, and 1:100. Hence, if the w_1: w_2 is set as 1:100, it means that the constant weight w_2 is one-hundred times as high as the constant weight w_1. Moreover, it can use w_1: w_2 as 1:0 or 0:1, that is, it means that the problem is considered as a makespan problem or a number-of-tardy-job problem, respectively.

However, from the data generations in this dissertation, the first criterion C_{max} mostly dominates the second criterion η_T. It will not concern the ratio of w_1: w_2 that the constant weight w_1 is greater than the constant weight w_2, since it seems to be considered only one criterion C_{max}. When the w_1: w_2 is set as 1:1, it seems that the value of C_{max} greatly dominates the value of η_T, but it can get a benefit from such a ratio, that is, assuming that the two best schedules of the C_{max} problems are generated as a schedule 1 (C_{max} = 23 time units and η_T = 1 job unit) and a schedule 2 (C_{max} = 23 time units and η_T = 2 job units), so the schedule 1 will be chosen.

Similarly, if the objective (Z_2) is set to seek a schedule that minimizes a positively weighted convex sum of the makespan and the number of tardy jobs, the objective function value for this dissertation is defined by:

$$Z_2 = \lambda C_{max} + (1 - \lambda)\eta_T \tag{3.4}$$

where $0 \leq \lambda \leq 1$. λ denotes the weight (or relative importance) given to C_{max} and η_T.

The relative importance (λ) and the constant weights (w) are interchangeable, that is, if it sets the relative importance (λ) to be 0.01, it means that the w_1: w_2 is set as 0.01: 099 (λ : 1 – λ) or 1: 99. The meaning of 1:99 is that, for example, the users consider that the penalty cost for the tardy jobs is more significant than the holding cost in their system. As stated above, it does not concern the ratio of w_1: w_2 that the constant weight w_1 is greater than the constant weight w_2 expect for the ratio of w_1: w_2 as 1:0, so the weight (or relative importance, λ) will be set to be 0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, or 1.0 (, or w_1: w_2 = 0:1, 1: 999, 1:199, 1:99, 1:19, 1:9, 1:1, or 1:0, respectively). For the application of this objective function, users can choose arbitrarily w_1 and w_2. Then they are only scaled into a convex combination. This is equivalent (because only the objective function value is multiplied by a constant — this does not influence the optimal solution) for both variants.

3.4.3 Constraint Functions

From the above idea of scheduling the flexible flow shop example problem in Section 3.3, it can construct the constraint functions as follows:

The variable X_{i0l}^{t} = 1 if the job *l* is sequenced as the first job on machine *i* at stage *t*, and $X_{ij(n+1)}^{t}$ = 1 if job *j* is sequenced as the last job on machine *i* at stage *t*.

❍ Constraints about job sequencing on all *k* stages:

These constraints, Constraints (1) through Constraints (7), need to ensure that the partial schedule on each machine at each stage is feasible.

Constraints (1) ensure that a job *l* at stage *t* scheduled on a machine *i* must be immediately preceded by exactly one different job *j*.

$$\sum_{i=1}^{m^t}\sum_{j=0}^{n} X_{ijl}^{t} = 1 \qquad t = 1, \ldots, k\,;\, l = 1, \ldots, n \qquad (1)$$

Constraints (2) ensure that a job *j* at stage *t* scheduled on a machine *i* must be immediately followed by exactly one different job *l*.

$$\sum_{i=1}^{m^t}\sum_{l=1}^{n+1} X_{ijl}^{t} = 1 \qquad t = 1, \ldots, k\,;\, j = 1, \ldots, n \qquad (2)$$

Hence, Constraints (1) and Constraints (2) ensure that only one job is assigned to each sequence position at each stage.

Constraints (3) ensure that only one job *l* will be assigned to the first position on each machine at each stage.

$$\sum_{l=1}^{n+1} X_{i0l}^{t} = 1 \qquad t = 1, \ldots, k\,;\ i = 1, \ldots, m^t \qquad (3)$$

Constraints (4) ensure that only one job *j* will be assigned to the last position on each machine at each stage.

$$\sum_{j=0}^{n} X_{ij(n+1)}^{t} = 1 \qquad t = 1, \ldots, k\,;\ i = 1, \ldots, m^t \qquad (4)$$

Constraints (5) assure that, after the job has been finished at any stage, it cannot be reprocessed at the same stage.

$$X_{ijj}^{t} = 0 \qquad t = 1, \ldots, k\,;\ i = 1, \ldots, m^t;\ j = 1, \ldots, n \qquad (5)$$

Constraints (6) force the construction of a consistent sequence at every stage. It means that if job *l* at stage *t* scheduled on a machine *i* is immediately preceded by job *j* then a job that is immediately preceded by job *l* at the same stage must be on same machine *i*.

$$\sum_{j=0}^{n} X_{ijl}^{t} = \sum_{j=1}^{n+1} X_{ilj}^{t} \qquad t = 1, \ldots, k\,;\ i = 1, \ldots, m^t\,;\ l = 1, \ldots, n \qquad (6)$$

Constraints (7) specify the decision variables X_{ijl}^{t} as binary variables.

$$X_{ijl}^{t} \in \{0,1\} \qquad t = 1, \ldots, k\,;\ i = 1, \ldots, m^t;\ j = 0, \ldots, n;\ l = 1, \ldots, n+1 \qquad (7)$$

❍ Constraints about finding the operating time:

Constraints (8) determine the operating time O_j^t for every job *j*, which is dependent on the machine and stage, where the job is processed. Due to the machine eligibility restrictions, the processing speed of a machine which cannot operate a job *j* at stage *t* is equal to the very small constant, *i.e.* lesser than the reciprocal value of sum of all job processing times and setup times to specify that job *j* cannot be processed on machine *i* of stage *t*.

$$O_j^t = \sum_{i=1}^{m^t} \sum_{l=1}^{n+1} \frac{ps_j^t}{v_{ij}^t} X_{ijl}^t \qquad t = 1, \ldots, k\,;\, j = 1, \ldots, n \qquad (8)$$

❍ Completion time forcing constraints:

Constraints (9) are a set of disjunctive constraints. It states that, if job *j* and job *l* are scheduled on the same machine at a particular stage with job *j* scheduled before job *l*, then job *j* must complete the processing before job *l* can begin. This constraint set forces job *l* to follow job *j* by at least the processing time of job *l* plus the setup time from *j* to *l* if job *l* is immediately scheduled after job *j*. The value of *B* is set to a very big constant, *i.e.* greater than the sum of all job processing times and setup times.

$$C_l^t - C_j^t \geq s_{jl}^t + O_l^t + ((\sum_{i=1}^{m^t} X_{ijl}^t) - 1)B \qquad t = 1, \ldots, k\,;\; j = 1, \ldots, n; \; l = 1, \ldots, n\,;\, j \neq l \qquad (9)$$

Constraints (10) ensure that the completion time of every job at each stage is a non-negative value.

$$C_j^t \geq 0 \qquad t = 1, \ldots, k;\; j = 1, \ldots, n \qquad (10)$$

Constraints (11) specify the conjunctive precedence constraints for the jobs, which states that a job cannot start its processing at stage *t* + 1 before it finishes at stage *t*, that is, job *l* at stage *t* to complete after it completes at stage *t*-1, plus its processing time at stage *t*, plus the setup time from its predecessor to *l* or the changeover time if job *l* is assigned to machine *i* at the first position at stage *t*.

$$C_l^t - C_l^{t-1} \geq \sum_{i=1}^{m^t}\sum_{j=1}^{n} X_{ijl}^{t} s_{jl}^{t} + \sum_{i=1}^{m^t} ch_{il}^{t} X_{i0l}^{t} + O_l^t \qquad t = 1, \dots, k;\ l = 1, \dots, n \qquad (11)$$

Constraints (12) apply only to stage one, stating that a job *j* cannot start its processing at stage one before its release date.

$$C_j^0 = r_j \qquad j = 1, \dots, n \qquad (12)$$

Constraints (13) apply only to jobs that are processed at the first sequence on each machine; that is, the job cannot start its processing before machine availability.

$$C_j^t \geq \sum_{i=1}^{m^t} a_i^t X_{i0j}^{t} + \sum_{i=1}^{m^t} ch_{il}^{t} X_{i0l}^{t} + O_j^t \qquad t = 1, \dots, k;\ j = 1, \dots, n \qquad (13)$$

Constraints (14) link the makespan decision variable.

$$C_{\max} \geq C_j^k \qquad j = 1, \dots, n \qquad (14)$$

❍ Tardiness forcing constraints:

Constraints (15) determine the correct value of the lateness (L_j)

$$T_j \geq C_j^k - d_j \qquad j = 1, \dots, n \qquad (15)$$

Constraints (16) specify only the positive lateness as the tardiness (T_j = max $\{0, C_j^k - d\}$).

$T_j \geq 0$ $\qquad j = 1, \ldots, n \qquad (16)$

Hence, Constraints (15) and Constraints (16) determine the correct value of the tardiness (T_j).

The value of B is set to a very large constant, *i.e.* greater than the sum of all job processing times and setup times.

Constraints (17) assure that if the tardiness (T_j) value is greater than zero the value of U_j must be greater than zero as well, otherwise the value of U_j is zero.

$BU_j \geq T_j$ $\qquad j = 1, \ldots, n \qquad (17)$

Constraints (19) specify the decision variables U_j as binary variables.

$U_j \in \{0,1\}$ $\qquad j = 1, \ldots, n \qquad (18)$

Hence, Constraints (17) through Constraints (18) link the decision variable of the number of tardy jobs; that is, if tardiness is larger than zero, the job is tardy; otherwise this job is not tardy.

3.4.4 A Numerical Example

In this section, the simple problem is illustrated. Due to the positively weighted convex sum of objectives, the weights λ for each problem can be 0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, or 1.0. They are solved by using AMPL and CPLEX8.0.0. This example consists of five jobs and three stages. The first stage has two unrelated parallel machines, and others have three unrelated parallel machines. Thus, set t to be

3, m^1 to be 2, and m^2 and m^3 to be 3. Table 3.7 shows the standard processing times, the release date, and the due date for every job. Machine availability (a_i^t) is given in Table 3.8. Table 3.9 gives the information about the relative machine speed (v_{ij}^t) of the machines which is dependent on the job. Table 3.10 and Table 3.11 show the matrix of changeover and setup times, respectively.

Table 3.7 The standard processing time at stage *t* (ps_j^t), release date (r_j) and due date (d_j) for job *j* of a two-stage flexible flow shop

job *i*	1	2	3	4	5
ps_j^1	71	47	74	14	88
ps_j^2	89	28	19	52	73
ps_j^3	23	47	40	35	35
r_j	3	130	166	52	69
d_j	275	302	313	204	315

Table 3.8 The availability of machine *i* at stage *t* (a_i^t)

	m_1^1	m_2^1	m_1^2	m_2^2	m_3^2	m_1^3	m_2^3	m_3^3
a_i^t	16	54	77	67	71	137	139	143

Table 3.9 The relative machine speed of job *j* on machine *i* at stage *t* (v_{ij}^t)

	job *j*	1	2	3	4	5
Stage *t*	machine *i*					
1	1	0.976	0.826	0.760	1.120	1.270
	2	1.042	0.796	0.754	1.186	0.706
2	1	0.868	0.772	1.228	0.850	0.958
	2	1.102	1.168	1.192	0.730	0.904
	3	0.916	1.180	0.844	1.246	1.234
3	1	0.910	1.024	0.910	0.898	1.174
	2	1.192	1.006	1.258	0.982	1.144
	3	1.180	0.970	1.288	1.018	0.922

Table 3.10 The matrix of changeover time of job *j* on machine *i* at stage *t* (ch_{ij}^{t})

Stage *t*	job *j* / machine *i*	1	2	3	4	5
1	1	35	44	35	20	33
	2	45	20	46	25	27
2	1	44	19	40	34	13
	2	30	27	28	43	47
	3	7	23	41	15	26
3	1	41	50	5	14	9
	2	7	13	39	44	29
	3	17	5	43	17	28

Table 3.11 The matrix of setup time from job *j* to job *l* at stage *t* (s_{jl}^{t})

Stage *t*	to job *l* / from job *j*	1	2	3	4	5
1	1	-	17	16	43	3
	2	30	-	9	0	10
	3	14	48	-	35	30
	4	0	19	11	-	39
	5	23	44	34	25	-
2	1	-	24	18	50	7
	2	37	-	49	2	23
	3	49	45	-	49	35
	4	12	31	13	-	43
	5	15	4	50	11	-
3	1	-	43	17	18	19
	2	19	-	18	47	5
	3	43	35	-	45	10
	4	26	50	4	-	17
	5	3	26	29	27	-

Table 3.12 Results of calculations

λ	Schedule	Value	*Cmax*	η_T
0	Stage 1: 1 – 5 – 3 // 4 – 2	3	1166.76	3
	Stage 2: 5 – 2 // 1 – 3 // 4			
	Stage 3: 4 – 3 // 5 – 2 // 1			
0.001	Stage 1: 5 – 2 // 4 – 1 – 3	3.358	360.897	3
	Stage 2: - // 1 – 3 // 4 – 5 – 2			
	Stage 3: 5 – 1 // 2 // 4 – 3			
0.005	Stage 1: 5 – 2 // 4 – 1 – 3	4.789	360.897	3
	Stage 2: - // 1 – 3 // 4 – 5 – 2			
	Stage 3: 5 – 1 // 2 // 4 – 3			
0.01	Stage 1: 5 – 2 // 4 – 1 – 3	6.579	360.897	3
	Stage 2: - // 1 – 3 // 4 – 5 – 2			
	Stage 3: 5 – 1 // 2 // 4 – 3			
0.05	Stage 1: 5 – 2 // 4 – 1 – 3	20.895	360.897	3
	Stage 2: - // 1 – 3 // 4 – 5 – 2			
	Stage 3: 5 – 1 // 2 // 4 – 3			
0.1	Stage 1: 5 – 2 // 4 – 1 – 3	38.790	360.897	3
	Stage 2: 1 // 3 // 4 – 5 – 2			
	Stage 3: 5 – 1 // 2 // 4 – 3			
0.5	Stage 1: 5 – 2 // 4 – 1 – 3	178.687	353.374	4
	Stage 2: 4 // 1 – 3 // 5 – 2			
	Stage 3: 5 – 1 // 4 – 3 // 2			
1	Stage 1: 5 – 2 // 4 – 1 – 3	353.374	353.374	5
	Stage 2: 4 – 3 // 1 // 5 – 2			
	Stage 3: 5 – 1 // 4 – 3 // 2			

From these input data, the results are found by using a modeling language AMPL with CPLEX solver that runs on Windows platform. The results of the calculations are shown in Table 3.12. The CPU time for finding the solution is about 60 seconds, but the CPU time is up to 30,000 seconds for the six jobs, *see* Jungwattanakit *et al.* (2005).

3.5 Complexity of the Problem

In this section, the complexity of the problem is discussed. As can be described in the previous section, there is a queue at each stage of the flexible flow shop, and all of n jobs can be processed on any one of the m^t machines at stage t (t= 1, …, k). When the job has been performed through the last stage k, it is complete and can finish the system at this point.

Considering only one stage, there are two decision functions to generate the schedules, assigning the jobs to machines and sequencing those jobs on each machine. For the first part, the total possible solutions (TPS_1) generated to assign the n jobs to m^t machines at stage t (remember that m^t is the number of machines at stage t) that are equal to:

$$TPS_1 = (m^t)^n \qquad (3.5)$$

Assuming that there are n_i^t jobs that are processed on machine i at stage t, and the summation of n_i^t jobs for all machine i in each stage t is equal to n jobs. Hence the total possible solutions (TPS_2) to sequence the n_i^t jobs on each machine at stage t are equal to:

$$TPS_2 = \prod_{i=1}^{m^t} n_i^t\,! \qquad (3.6)$$

Consequently, the total possible solutions (TPS_3) generated of two functions, allocation and sequencing functions, are equal to:

$$TPS_3 = TPS_1 \times TPS_2 = (m^t)^n \prod_{i=1}^{m^t} n_i^t\,! \tag{3.7}$$

From Equation (3.7), it can be noticed that the maximum number of the total possible solutions TPS_3 will occur when there is an $n_i^t = n$ and otherwise zero. Thus the total maximum possible solutions (TPS_{3max}) generated are equal to:

$$TPS_{3max} = (m^t)^n n! \tag{3.8}$$

Moreover, it can be noted that the minimum value of the total possible solutions TPS_3 will occur when each value of n_i^t is nearly equal. Thus the total minimum possible solutions (TPS_{3min}) generated are equal to:

$$TPS_{3min} = (m^t)^n \prod_{i=1}^{m^t} \left\lfloor \frac{n}{m^t} \right\rfloor ! \tag{3.9}$$

where $\left\lfloor \frac{n}{m^t} \right\rfloor$ is the largest integer number that is lesser than or equal to $\frac{n}{m^t}$

Consequently, considering all k stages, the total possible sequence combinations (TPC) of the problem under consideration are equal to:

$$\prod_{t=1}^{k} \left(\left(m^t\right)^n \prod_{i=1}^{m^t} \left\lfloor \frac{n}{m^t} \right\rfloor ! \right) \le TPC \le \prod_{t=1}^{k} \left(\left(m^t\right)^n n! \right) \tag{3.10}$$

From Equation (3.10), it is found that the total possible sequence combinations depend on the number of jobs (n), the number of stages (k), and the number of machines (m^t) of each stage.

Table 3.13 shows the total number of possible sequence combinations of the problems with 5 stages and 5 machines per stage, when the number of jobs increases.

Although the given machine environment is quite small, the total number of possible sequence combinations is so large when the number of jobs increases.

Table 3.13 Total number of possible sequence combinations for any number of jobs of the problems with 5 stages and 5 machines per stage

Number of jobs	Total possible sequence combinations of the problem	
	minimum	maximum
5	2.98E+17	7.42E+27
10	2.98E+42	5.59E+67
15	7.53E+71	1.01E+113
20	2.53E+104	6.72E+161

Table 3.14 shows the total number of possible sequence combinations of the problems with 5 jobs and 5 machines per stage, when the number of stages increases. Similarly, although the given machine environment is quite small, the total number of possible sequence combinations is so large when the number of stages increases.

Table 3.14 Total number of possible sequence combinations for any number of stages of the problems with 5 jobs and 5 machines per stage

Number of stages	Total possible sequence combinations of the problem	
	minimum	maximum
5	2.98E+17	7.42E+27
10	8.88E+34	5.50E+55
15	2.65E+52	4.08E+83
20	7.89E+69	3.02E+111

A highest current computer technology, an Intel Core 2 Dual Core processor, has upped its clock speed to 3.16 GHz (or 3.16E+09 Hz), that is, the computer can generate its clocks to 3.16E+09 cycles per second. With an optimistic assumption that one clock cycle generates one solution, the computer can generate 3.16E+09 solutions per second, so it would be at least 2.99 years to solve the 5-job 5-stage 5-

machine-per-stage problems for the minimum total possible sequence combinations, at least 7.92E+52 years to solve the 5-job 20-stage 5-machine-per-stage problem, or at least 2.53E+87 years to solve the 20-job 5-stage 5-machine-per-stage problem. Hence, the difficulty of a problem rises when the number of jobs and/or the number of stages and/or the number of machines per stage becomes large.

It is not practical to evaluate all possible sequence combinations to find the optimal sequence. This means there is a need for a method or an algorithm which can make it possible to avoid extensive computations and find a quick solution. Of course, there is drawback to such algorithms; they do not guarantee optimally.

For the above reason, although the problem under consideration is difficult to solve optimally for large-sized problems, an exact solution using mathematical programming formulation is still generally necessary for solving small-sized problems. Some important advantages for formulating the exact algorithm from the mathematical programming are shown in Chapter I (Section 1.7.3.1).

3.6 Conclusion

This chapter develops the mathematical model to minimize a positively weighted convex sum of the makespan and the number of tardy jobs for the flexible flow shop problem with unrelated parallel machines and sequence dependent setup time. This chapter begins with the introduction section that gives the definition and the importance of the mathematical programming. It says that even though the CPU time required to find an optimal schedule by using the mathematical programming is very high, the mathematical models are worth formulating because the optimal solutions obtained for small-sized problems can also be used to evaluate the performance of various heuristics that are developed to find the approximate solution. Moreover, the large-sized problem may be able to solve in near future since the development of the computer devices is improving with a fast pace.

Then, this chapter follows with a descriptive example problem that is a two-stage flexible flow shop environment, which consists of two unrelated parallel machines in stage 1 and one machine in stage 2 in Section 3.3. In that section, a detail of scheduling process of one possible solution is presented. The total number of possible sequence combinations generated from such a problem is equal to 12.

Next, in Section 3.4, the mathematical formulation is presented. It starts with the assumptions of the model. The objective function, why a positively weighted convex sum of makespan and the number of tardy jobs as performance measure of the schedules is used in this dissertation, is presented. It concludes the meaning of the constant weights (w) and the relative importance (λ) that are interchangeable, that is, users can choose arbitrarily w_1 and w_2, then they are only scaled into a convex combination in terms of the relative importance (λ). However, the limitation of this model is that users have to choose their suitable values of w_1 and w_2 for their environment before using the model. A numerical example to illustrate a use of the mathematical model for the problem under consideration by using an AMPL/CPLEX solver, which is a commercial solver, is given. Although the AMPL/CPLEX solver is considered one of the fastest commercial solvers available and is faster than an enumerative search, it still requires tremendous computational time.

In Section 3.5, under an intuitive method that is called a complete enumeration, the total number of possible sequence combinations that will be generated by such an approach is estimated. It is concluded that the total number of possible sequence combinations is so large when the number of jobs and/or number of machines per stage and/or number of stages increases. Attempts to find all solutions are unsuccessful as they require too much CPU time. Thus, it is hard to find the optimal solution or even best solution by using either a complete enumeration or a mathematical model. Hence, in the next chapter, the development of some heuristics to obtain a good solution in acceptable time will be proposed.

CHAPTER IV

HEURISTIC SOLUTION CONCEPTS

This chapter provides the heuristic solution concepts that are used to find the approximation solution of the flexible flow shop problem with unrelated parallel machines. The proposed heuristic solution concepts consist of three main kinds of the heuristic algorithms, which are the *constructive*, *improvement*, and *iterative* algorithms (Winston and Venkataramanan, 2003).

The outline of this chapter is as follows: The introduction gives the definition and importance of the heuristic solution approach. Then, the conceptual framework for heuristics is explained. The next section proposes the schedule construction approach, where a starting job sequence for the first processing stage is known. Then, the constructive algorithms are proposed for the determination of the starting job sequence for the first stage for the flexible flow shop problem in the next section. The improvement algorithms are proposed in the next section. Some iterative algorithms are proposed, namely simulated annealing, tabu search, and genetic algorithms as well as other variants with choices of an initial solution for the iterative algorithms. Finally, a conclusion will be drawn.

4.1 Introduction

The main limitation of the exact solution approach is the high memory consumption of finding the optimal solution, so the heuristic approach the behind idea of which is easy to find the good solution is appeared. In combinatorial optimization (Cook *et al.*, 1998), the heuristic approach is the technique designed to solve a problem that ignores whether the output can be proven to be correct, but which usually produces a good result. It is intended to gain computational performance or conceptual simplicity at the cost of accuracy or precision. The aim is to achieve a good enough output rather than exact output, but this is rewarded with a great computational performance able to turn intractable problems into tractable ones (Gagliardi, 2007).

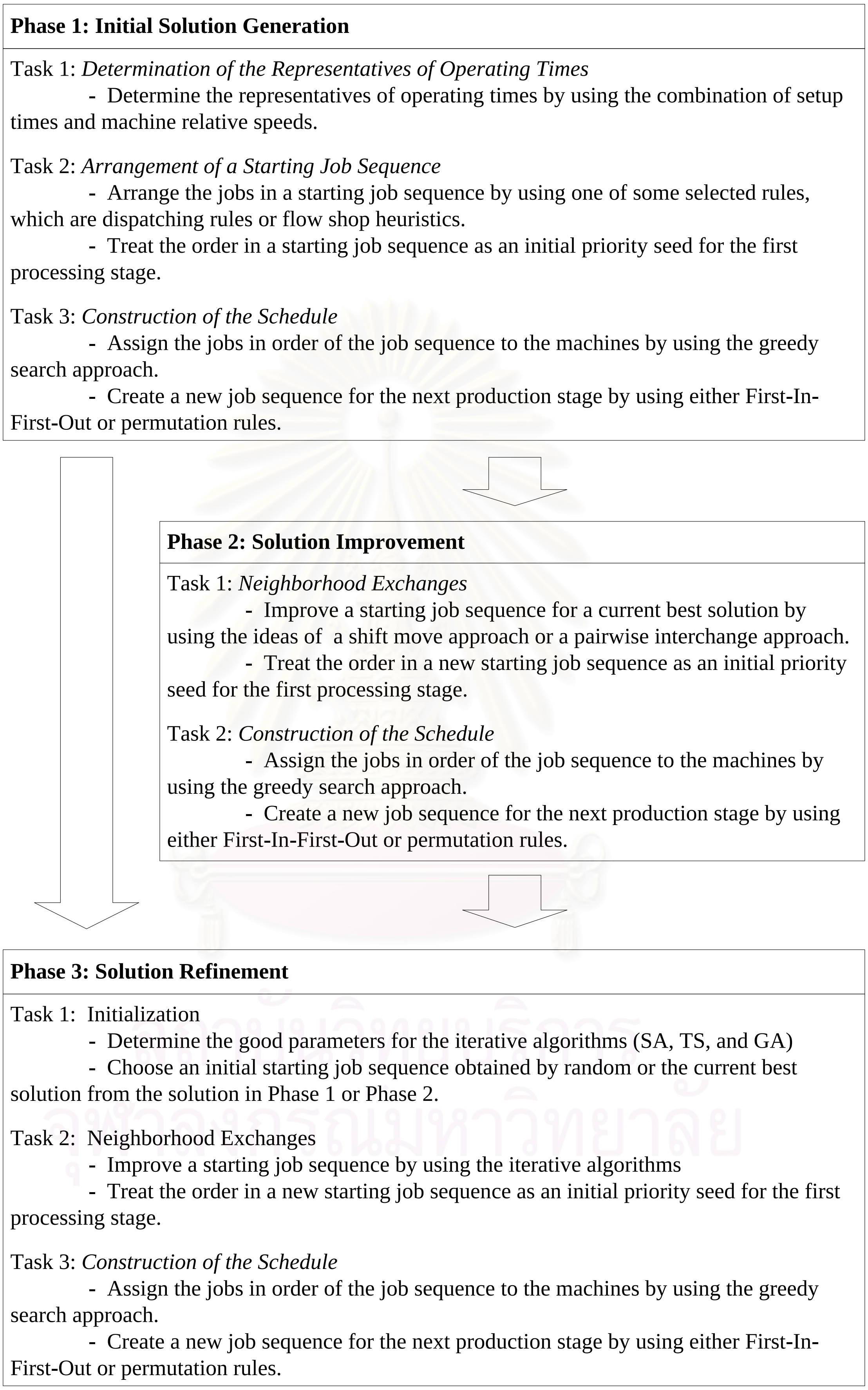
As stated in Chapter I (*see* Section 1.2.2), a flexible flow shop problem is a category of machine scheduling problems which is difficult to solve (Garey and Johnson ,1979; Gupta, 1988; Pinedo, 1995); even though the problem has two stages with one stage having at least two machines, it has already been proved to be NP-hard (Hoogeveen *et al.*, 1996). From the previous chapter, the complexity of the problem makes it computationally expensive to use an exact solution technique, which gives an optimal solution of the problem. Consequently, it is impractical to find the optimal solution. This means there is a need for an algorithm which can make it possible to avoid extensive computations and find a quick solution. That is why an a*pproximate solution technique* (or a *heuristic solution approach*) is needed. A heuristic solution approach may provide the optimal solution for the problem even though such an approach does not guarantee the optimality of its solution. This fact implies that there may be some space for a heuristic solution approach to be improved (Wang, 2005). Consequently, various types of approximation algorithms have been proposed. Some importance advantages and disadvantages are stated in Chapter I (*see* Section 1.7.3.2).

4.2 Conceptual Framework for Heuristics

Solving a flexible flow shop problem requires finding a good schedule that satisfies both some measures of effectiveness and the production controls. The main idea of the proposed algorithm framework comes from the following questions:

1. Which machines will be allocated to perform each job?
2. When will each job be processed?

Figure 4.1 shows the initial concept of the heuristic solution approach. The concept used in this dissertation is proposed in three phases, which are the initial solution generation, the solution improvement, and the solution refinement.



Phase 1: Initial Solution Generation

Task 1: *Determination of the Representatives of Operating Times*
- Determine the representatives of operating times by using the combination of setup times and machine relative speeds.

Task 2: *Arrangement of a Starting Job Sequence*
- Arrange the jobs in a starting job sequence by using one of some selected rules, which are dispatching rules or flow shop heuristics.
- Treat the order in a starting job sequence as an initial priority seed for the first processing stage.

Task 3: *Construction of the Schedule*
- Assign the jobs in order of the job sequence to the machines by using the greedy search approach.
- Create a new job sequence for the next production stage by using either First-In-First-Out or permutation rules.

Phase 2: Solution Improvement

Task 1: *Neighborhood Exchanges*
- Improve a starting job sequence for a current best solution by using the ideas of a shift move approach or a pairwise interchange approach.
- Treat the order in a new starting job sequence as an initial priority seed for the first processing stage.

Task 2: *Construction of the Schedule*
- Assign the jobs in order of the job sequence to the machines by using the greedy search approach.
- Create a new job sequence for the next production stage by using either First-In-First-Out or permutation rules.

Phase 3: Solution Refinement

Task 1: Initialization
- Determine the good parameters for the iterative algorithms (SA, TS, and GA)
- Choose an initial starting job sequence obtained by random or the current best solution from the solution in Phase 1 or Phase 2.

Task 2: Neighborhood Exchanges
- Improve a starting job sequence by using the iterative algorithms
- Treat the order in a new starting job sequence as an initial priority seed for the first processing stage.

Task 3: *Construction of the Schedule*
- Assign the jobs in order of the job sequence to the machines by using the greedy search approach.
- Create a new job sequence for the next production stage by using either First-In-First-Out or permutation rules.

Figure 4.1 Initial concept of the heuristic solution approach

The *initial solution generation* phase is to generate the initial feasible solution. It consists of three tasks. Firstly, it has to determine the representatives of operating times, since the processing and setup times for every job are unknown until all jobs have already been assigned to machines. Secondly, a starting job sequence is constructed. All jobs are arranged by using one of some selected rules which are adapted from dispatching rules or flow shop heuristics. The order in a starting job sequence is treated as an initial priority seed for scheduling in the first production stage. Finally, the construction of the schedule is carried on to find the schedule solution. In this task, there are two main approaches. The first approach is the greedy search approach that is used to assign the jobs to the machines in the stage, and the second approach is the use of either First-In-First-Out (FIFO) or permutation rules to create a new job sequence for the next production stage. All tasks of this phase are made up of the *constructive algorithms* in this dissertation.

Next, the *solution improvement* phase is proposed to improve the current solution obtained from the initial solution generation in Phase 1. Two tasks are implemented in this phase. The first task is the neighbor exchanges. It is applied on the starting job sequence, that is, it considers first only the current starting job sequence that is obtained from the particular *constructive algorithm* and exchanges the job order in such a current starting job sequence. Then, the second task is to construct the schedule output. Again, two approaches of assigning the jobs to the machines and determining the new job sequence are used. The *improvement algorithms* proposed in this dissertation are followed in this phase.

The *solution refinement* in Phase 3 is applied. This phase is used as the *iterative algorithm* in this dissertation. Its tasks are the initialization, neighborhood exchanges, and construction of the schedule. The initialization in this phase consists of determining the good parameter for the *iterative algorithms* (namely, the simulated annealing, tabu search and genetic algorithms) and choosing an initial starting job sequence obtained by random or the best solution from either the *constructive algorithms* in Phase 1 or the *improvement algorithms* in Phase 2. Then, the neighborhood exchanges are applied to improve the initial starting job sequence. They use the idea of the *iterative algorithms* (known as *artificial intelligent algorithms*) to function as the neighborhood exchanges. Finally, in each iteration of the neighborhood exchanges, the schedule construction approach is still used to determine the schedule output in order to evaluate the fitness or improvement of the new starting job sequence.

That is all the initial concept of the heuristic solution approach. Then, more details of the conceptual framework based on the initial concept of the heuristic solution approach are explained. On the ideas of heuristic procedures (Winston and Venkataramanan, 2003), the heuristic techniques are characterized by using a particular approach for obtaining a good solution in efficient time (called a *constructive algorithm*) and for incrementally improving to an existing solution by neighborhood exchanges or local search (called an *improvement algorithm*). As a result, they tend to get trapped in a local optimal solution until next attempts to develop other general heuristics that can work on a variety of problems have met with the development of solution methodologies based on an artificial intelligence method (called an *iterative algorithm*). Consequently, three kinds of heuristics, namely constructive, improvement, and iterative algorithms, are proposed in this dissertation.

Before concerning the heuristic algorithms, the *schedule construction approach* is presented to construct the schedule output (*see* Section 4.3). The proposed schedule construction approach is based on the idea of Santos *et al.* (1996). It starts with a *starting job sequence*, which may be randomly generated. The starting job sequence is represented by a permutation-based code (or job code) using integers; for example, the nine jobs can be coded as the starting job sequence [9 3 6 5 8 7 2 4 1]. The stages are scheduled separately. Considering the jobs in the order of the starting job sequence, each job is loaded on the machine with the minimum completion time in the first considered production stage (referred to as a *greedy search*). The idea is to balance evenly the workload in a heuristic way as much as possible. Then, the schedule construction approach uses the particular rules (*i.e.* the

FIFO and permutation rules) to generate a *new job sequence* for the next production stage. Again, considering the jobs in the new job sequence, each job is loaded on the machine with the minimum completion time in the next considered production stage. Repeat the steps of the algorithms until all production stages are considered (*see* Approach 1). Figure 4.2 shows a flow diagram of the schedule construction approach.

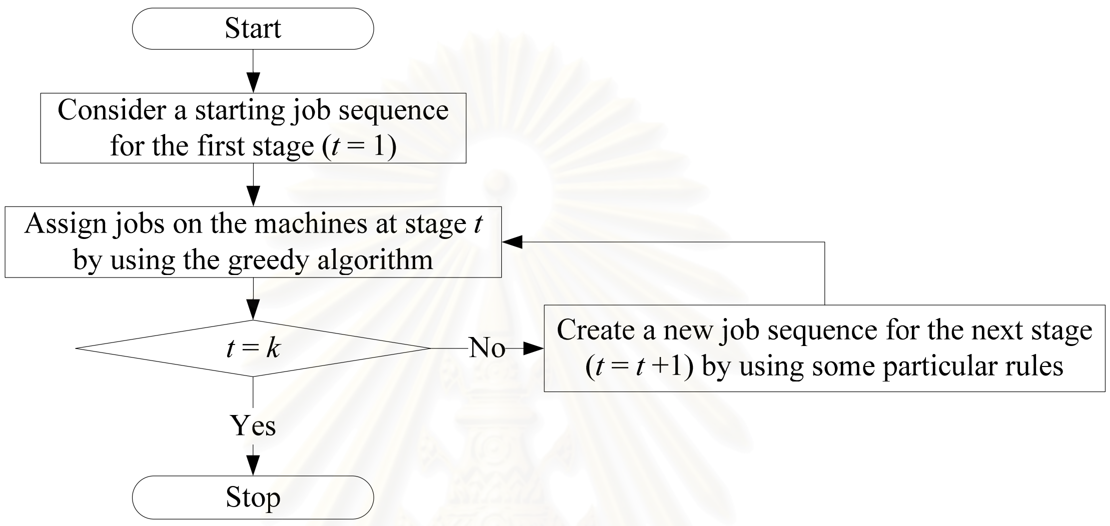

Start
Consider a starting job sequence for the first stage (t = 1)
Assign jobs on the machines at stage t by using the greedy algorithm
$t = k$
No
Create a new job sequence for the next stage ($t = t$ +1) by using some particular rules
Yes
Stop

Figure 4.2 A flow diagram of the schedule construction approach

Firstly, the *constructive algorithms* are proposed (*see* Section 4.4). From the schedule construction approach, it is noticed that the schedule output depends on the starting job sequence. Santos *et al.* (1996) and Wang and Hunsucker (2003) conclude their studies that the quality of a schedule is improved by using some particular rules to determine a starting job sequence for the first stage. Consequently, in this dissertation, a starting job sequence is created by using some particular rules. The job operating times are required to find a starting job sequence for the first stage, but, due to the unrelated parallel machines and sequence-dependent setup times, the processing and setup times for every job are dependent on the machine and the previous job. This means that job operating times is not known until an assignment of jobs to machines for the corresponding stage has been done. Thus, before finding the staring job sequence for the first stage, a approach for finding the representatives of the relative machine speeds and the setup times will be proposed (*see* Algorithm 1).

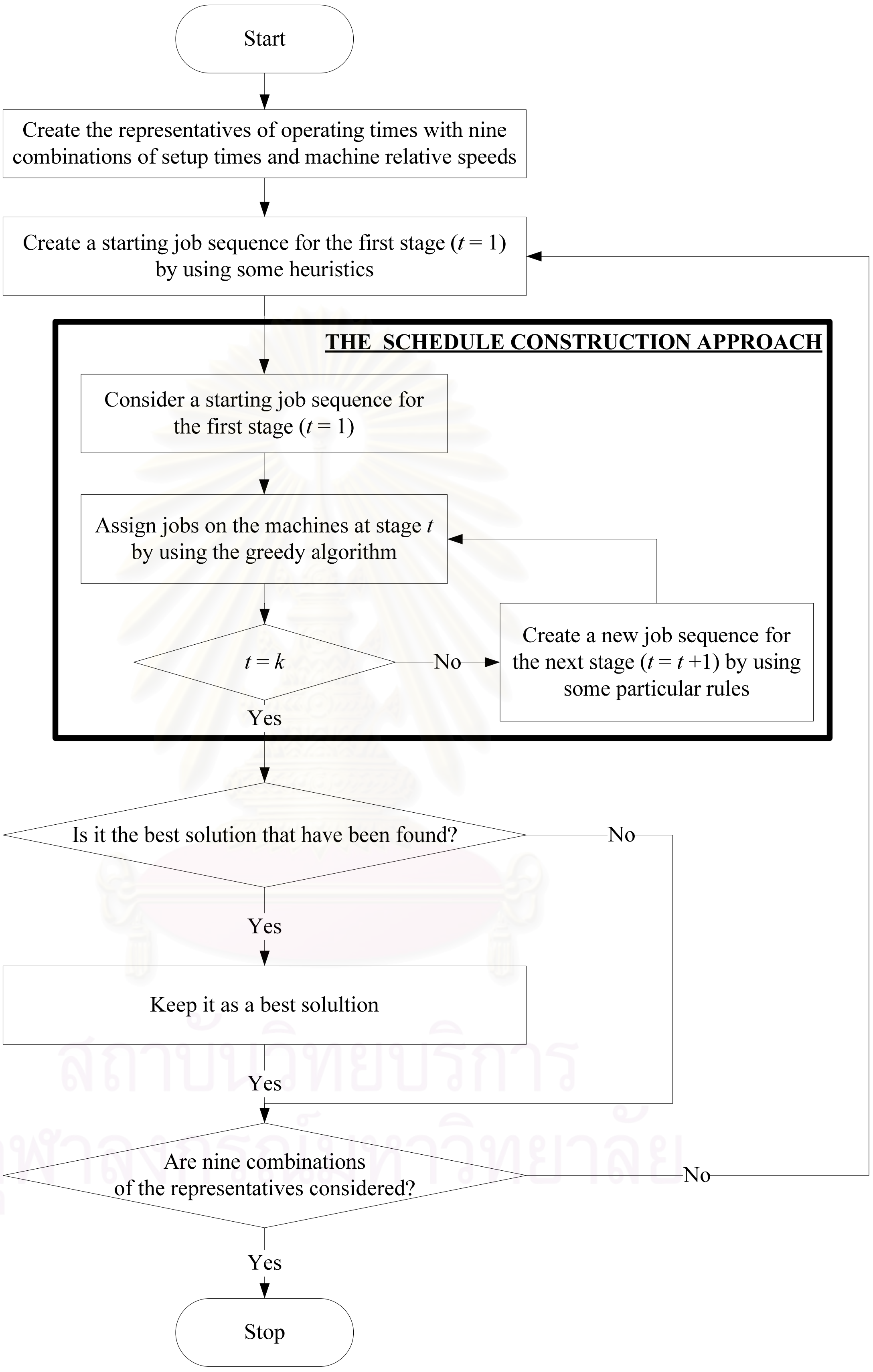

Start
Create the representatives of operating times with nine combinations of setup times and machine relative speeds
Create a starting job sequence for the first stage (t = 1) by using some heuristics
THE SCHEDULE CONSTRUCTION APPROACH
Consider a starting job sequence for the first stage (t = 1)
Assign jobs on the machines at stage t by using the greedy algorithm
$t = k$
No
Create a new job sequence for the next stage ($t = t$ +1) by using some particular rules
Yes
Is it the best solution that have been found?
No
Yes
Keep it as a best solultion
Yes
Are nine combinations of the representatives considered?
No
Yes
Stop

Figure 4.3 A flow diagram of the constructive algorithms

After generating the representatives of operating times, the algorithms will generate a starting job sequence and use the schedule construction approach to construct a schedule solution. Now, this step will consider how to arrange the jobs in a starting job sequence for the first stage. For the first idea, it will concern the information of the job operating times as a whole, that is, each representative of operating time of all stages will be added to the total representative of operating time of each job to find a starting job sequence for the first stage. Then, it will apply some simple dispatching rules, namely SPT, LPT, ERD, EDD, MST, and S/P, to arrange all jobs in the starting job sequence (*see* Section 4.4.1).

However, the problem under consideration is a combination of the parallel machine problem and the flow shop problem, so the idea of the well-known flow shop heuristics should be adapted in this problem; for example, the ideas of using the priority of the different operating times at each stage in the Palmer, CDS, Gupta, and Dannenbring heuristics should be considered for sequencing jobs in the starting job sequence. Moreover, the probably best flow shop heuristic, called the NEH algorithm, based on the idea of the insertion algorithm is also adapted to find the starting job sequence. Consequently, this dissertation will investigate the influence of using the adapted flow shop heuristics, namely Palmer, CDS, Gupta, Dannenbring, and NEH, for determining the starting job sequence (*see* Section 4.4.2). Figure 4.3 shows a flow diagram of the constructive algorithms.

The computational efficiency of the constructive algorithms will be tested in the next chapter. The result will show the performance of the constructive algorithms by comparing with the best solution that is found in the tests for the medium-and large-sized test problems (*see* Section 5.3.1) and the optimal solution for the small-sized test problems (*see* Section 5.4). It will conclude which algorithm gives the good performance in average.

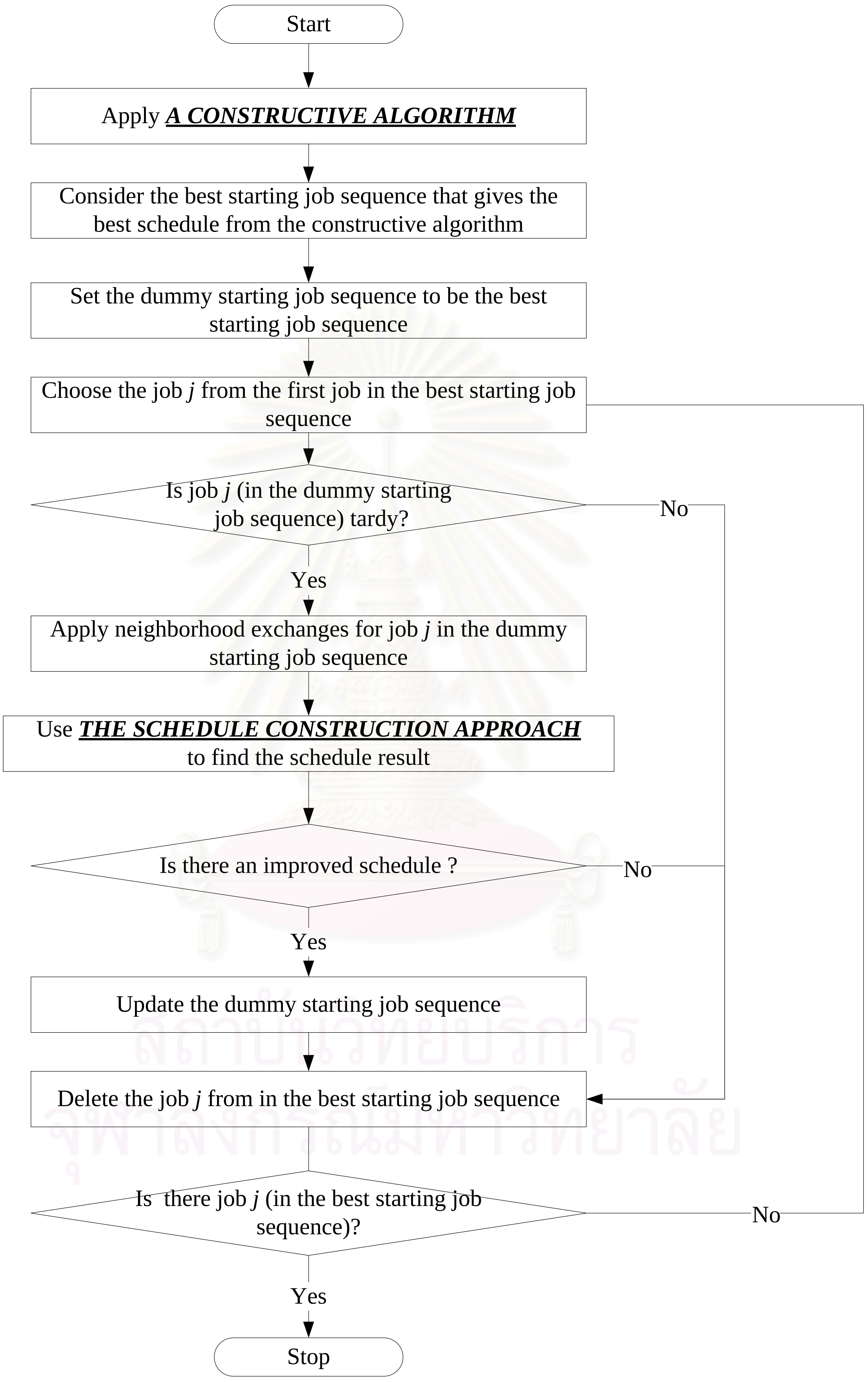

Start
Apply A CONSTRUCTIVE ALGORITHM
Consider the best starting job sequence that gives the best schedule from the constructive algorithm
Set the dummy starting job sequence to be the best starting job sequence
Choose the job j from the first job in the best starting job sequence
Is job j (in the dummy starting job sequence) tardy?
No
Yes
Apply neighborhood exchanges for job j in the dummy starting job sequence
Use THE SCHEDULE CONSTRUCTION APPROACH to find the schedule result
Is there an improved schedule ?
No
Yes
Update the dummy starting job sequence
Delete the job j from in the best starting job sequence
Is there job j (in the best starting job sequence)?
No
Yes
Stop

Figure 4.4 A flow diagram of the improvement algorithms

Next, after obtaining a good solution from the constructive algorithms, it can obtain incremental improvement (called an *improvement algorithm*) to an existing solution by neighborhood exchanges (Winston and Venkataramanan, 2003). Now, it is noticed that the constructive algorithms do not consider the minimization of the number of tardy jobs, so this phase will provide the idea of improving the current solution by using some approaches, namely *pairwise interchange* and *shift move* strategies, for jobs that are tardy. The first improvement idea (called a *shift move approach* or an *insertion approach*) is based on the fact that the tardy job should be shifted to perform earlier or later if the solution can improve. The next improvement idea (called a *pairwise interchange approach*) is based on the fact that the tardy job may be exchanged with a job that is assigned ahead or later.

Figure 4.4 shows the flow diagram of the improvement algorithms, where the idea will apply on a starting job sequence that obtains from the constructive algorithm and still use the schedule construction approach to find the schedule solution (*see* Section 4.5). Again, the performance of the improvement algorithms is provided in Chapter V. It will compare with the best solution that is found in the tests for the medium-and large-sized test problems (*see* Section 5.3.2) and the optimal solution for the small-sized test problems (*see* Section 5.4). It will conclude which algorithm gives the good performance in average.

As a result, those algorithms tend to get trapped in a local optimal solution, but they can apply other general heuristics with the development of solution methodologies based on an artificial intelligence method (called an *iterative algorithm*). Jones *et al.* (2002) show that 70% of the articles utilize a genetic algorithm as the primary metaheuristic, 24% a simulated annealing, and 6% a tabu search. Thus, to determine near-optimal solutions, the simulated annealing, tabu search, and genetic algorithms are proposed in this dissertation as the iterative algorithms (*see* Section 4.6). Again, it will apply the iterative algorithms on a starting job sequence and use the schedule construction approach to construct a schedule solution in order to evaluate the fitness of each solution generated in each iteration.

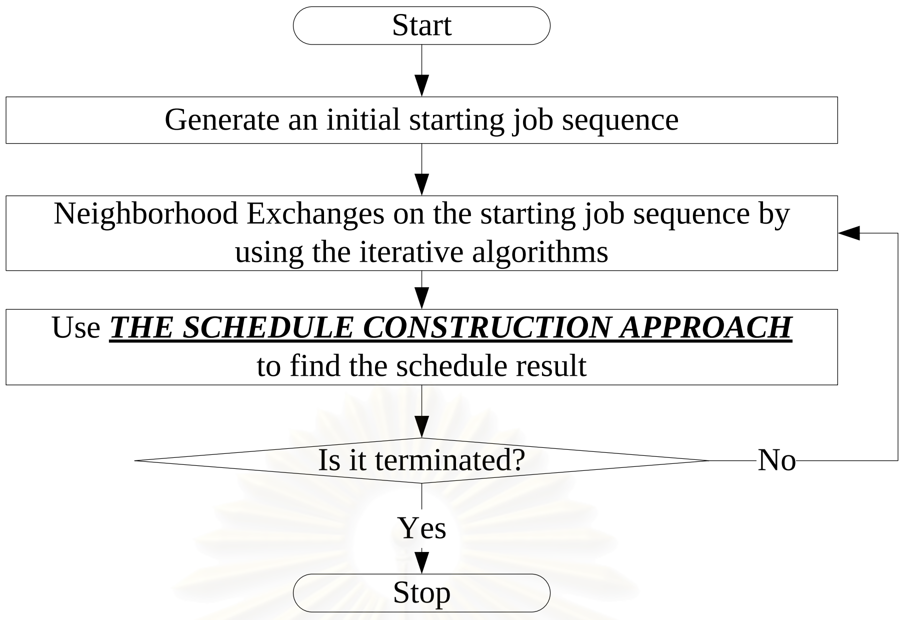

Start
Generate an initial starting job sequence
Neighborhood Exchanges on the starting job sequence by using the iterative algorithms
Use THE SCHEDULE CONSTRUCTION APPROACH to find the schedule result
Is it terminated?
No
Yes
Stop

Figure 4.5 A flow diagram of the iterative algorithms

However, it is noted that each iterative algorithm firstly generates a starting job sequence as an initial solution by random. Reeves (1995), Wang and Zheng (2003), and Grabowski and Wodecki (2004) have used the NEH algorithm as the initial solution instead of a random initial solution for their iterative methods, namely a tabu search and a genetic algorithm. Janiak *et al.* (2007) have used the best solution among their constructive algorithms such ECT, EDD, and EDP as an initial solution for the simulated and tabu search algorithms. These imply that a better initial solution gives a better schedule solution. Consequently, it will apply this idea on the iterative algorithms by using a solution that is found by both constructive algorithms and improvement algorithms as a biased initial solution (or a part of the initial population for the genetic algorithm) (*see* Section 4.7). Figure 4.5 shows all concepts of the iterative algorithms. The performance of the iterative algorithms which is compared with the best solution is shown in Section 5.3.3 and which is compared with the optimal solution is shown in Section 5.4.

After all algorithms are tested in Chapter V, each algorithm will be concluded under the data generation that is given in Section 5.2. Then, one heuristic solution algorithm that gives the good performance in average will be recommended to use for solving the problem under consideration (*see* Section 5.5).

4.3 Schedule Construction Approach

In this section, the algorithm of the schedule construction approach is proposed. As stated in Section 4.2, such an approach is based on the idea of the greedy search approach (*see* Figure 4.2). The purpose of the schedule construction approach is to construct a schedule that answers two questions, which machine will be allocated to process each job and when each job will be processed. This approach is also developed to evaluate the performance of the schedule which is given by a starting job sequence at the first stage and to evaluate the fitness of the solution in each iteration of the iterative algorithms in Section 4.6.

The idea of the greedy search approach is used in many researches of the flexible flow shop problem, *see* Sundararaghavan, Kunnathur, and Viswantha (1997), Santos *et al.* (1996), and Lin and Liao (2003), that is, jobs are assigned as soon as possible to the machines at every stage using the starting job sequence determined for the first stage. For other stages, a new job sequence is constructed, and it still uses the greedy search approach to distribute jobs to one of machines. There are basically two approaches for constructing the new job sequence for other stages. The first way is that for the other stages, *i.e.* from stage two to stage k, jobs are ordered according to their completion times at the previous stage. This means that the First-In-First-Out (FIFO) rule is used to find the new job sequence for the next stage by means of the job sequence of the previous stage. The advantages of the FIFO rule are simple and easy to understand, that is, any jobs come first, and then they should serve first. However, because of this nonpreemptive scheduling, short processes which are at the back of the queue have to wait for the long process at the front to finish, so if short processes are sometime yielded to process first, it may be possible to improve the solution. Consequently, to omit the idea of the FIFO rule and to believe that the jobs in the starting job sequence should use all stages, the second way proposed is to sequence the jobs for other stages by using the same job sequence as for the first stage, called the *permutation* rule (Pinedo, 1995).

In this dissertation, the ideas stated above are made for the modifications for the problem under consideration. Approach 1 shows the steps of the schedule construction approach, where a starting job sequence for the first processing stage is known. It consists of a greedy approach which constructs a schedule for *n* jobs at a particular stage and particular rules (*i.e.* either the FIFO or permutation rules) to find the new job sequence for the next production stage. The objective of this greedy approach is to minimize the flow time and the idle time of the machines. The idea is to balance evenly the workload in a heuristic way as much as possible.

Approach 1: Schedule construction

Input: Assume now that a starting job sequence (ω^1) for the first stage has already been determined, $\omega^1 = [\omega^1[1]\ \ \omega^1[2]\ \ \dots\ \ \omega^1[n]]$, where $\omega^1[q]$ is a job at position *q*, $q \in \{1, \dots, n\}$, in a starting job sequence (ω^1) for stage 1, *i.e.* $\omega^1 = [2\ \ 1\ \ 3]$, it means that job number 2 is the first job to be scheduled on any machines or $\omega^1[1] = 2$, and job number 1 and 3 are followed respectively.

Step 1: Set stage *t* to be 1.

Step 2: Let a job sequence π be ω^t; ω^t is a job sequence for stage *t*.

Step 3: Initialize the machine available time $av^t[i] = a_i^t$, $i \in \{1, \dots, m^t\}$.

Step 4: Assign the first job *j* in the job sequence π to the machine *i* at stage *t* that has the minimum completion time by using the following formula.

<u>*Case1*</u> Job *j* is assigned to the machine *i* at the first position:

$$C_{ij}^{t/} = \max\{r_j^t + \frac{ps_j^t}{v_{ij}^t} + ch_{ij}^t,\quad av[i] + \frac{ps_j^t}{v_{ij}^t} + ch_{ij}^t\};$$

<u>*Case 2*</u> Job *j* is assigned to the machine *i* at any other position:

$$C_{ij}^{t/} = \max\{r_j^t + \frac{ps_j^t}{v_{ij}^t} + s_{lj}^t,\quad av[i] + \frac{ps_j^t}{v_{ij}^t} + s_{lj}^t\}.$$

Step 5: If there is only machine *i** that gives the minimum completion time $C^{t*} = C_{i^*j}^{t/}$ of job *j* then assign job *j* on machine *i** at stage *t*, and then go to Step 8.

Step 6: For every machine i^* that gives the minimum completion time $C_{i^*j}^{t/}$ of job j, calculate the waiting time w_{i^*j} of these machines if job j is assigned on it by using calculation:

<u>*Case1*</u> Job j is assigned to the machine i at the first position, for every machine i^*:

$$w_{ij} \quad = \quad C_{ij}^{t/} - av^t[i] - \frac{ps_j^t}{v_{ij}^t} - ch_{ij}^t\,;$$

<u>*Case2*</u> Job j is assigned to the machine i at any other position for every machine i^*:

$$w_{ij} \quad = \quad C_{ij}^{t/} - av^t[i] - \frac{ps_j^t}{v_{ij}^t} - s_{lj}^t\,.$$

Step 7: Select only one machine i^* where the smallest idle time w_{i^*j} of job j.

Step 8: Update the available time of machine i^*; $av^t[i^*] = C_{i^*j}^{t/}$.

Step 9: Store the completion time C_j^t of job j be equal to $C_{i^*j}^{t/}$.

Step 10: Let r_j^{t+1} be C_j^t

Step 11: Update unscheduled job sequence $\pi = \pi - \{j\}$. If $\pi \neq \phi$ then go to Step 4.

Step 12: <u>*Case1*</u> *First-In-First-Out Approach*:

Update ω^{t+1} such that current completion time of job $\omega^{t+1}[n]$ at stage $t \leq$ current completion time of job $\omega^{t+1}[n+1]$ at stage t;

<u>*Case2*</u> *Permutation Approach*:

Update $\omega^{t+1} = \omega^1$.

Step 13: If $t < k$ (k is the total number of stages), then $t = t + 1$, and go to Step 2.

Step 14: Return the best solution.

4.4 Constructive Algorithms

From the previous section, it is noticed that the starting job sequence used for the first stage is important for the schedule construction approach. In order to determine the starting job sequence for the first stage by some heuristics, remember that the processing and setup times for every job are dependent on the machine and the previous job. This means that they are not fixed until an assignment of jobs to machines for the corresponding stage has been done. Thus, for applying a schedule construction approach for fixing the starting job sequence for stage one, finding the representatives of the relative machine speeds and the setup times is necessary (*see* Algorithm 1).

The representatives of relative machine speed ($v_{ij}^{\prime t}$) and setup time ($s_{lj}^{\prime t}$) for stage *t*, *t* = 1, … , *k*, use the minimum, maximum, and average values of the data. Thus, the representative of operating time of job *j* at stage *t* is the sum of the processing time $ps_j^t \big/ v_{ij}^{\prime t}$ plus the representative of the setup time $s_{lj}^{\prime t}$. Nine combinations of relative machine speeds and setup times will be used in the suggested algorithm. The starting job sequence for the first stage is then fixed as the starting job sequence with the best function value obtained by all combinations of the nine different relative machine speeds and setup times.

The idea of these combinations is to deal with unknown processing and setup times before assigning jobs to machines by using the expected time, the pessimistic time, and the optimistic time. Firstly, it is assumed that each job has an equal chance to assign on each machine at each stage, so the expected values are used to apply, that is, the algorithm uses the average values of both relative machine speeds and setup times to generate the representatives of the operating times. However, there are some chances that some jobs will assign to any machines that spend the operating times longer or shorter than the average values (especially, they should assign to the machine that spends time shorter). This means that the results are better than using only the average values as the representative values. To take these uncertainties into account and to find tune the value of the operating time, this idea will use the nine combinations of the both relative machine speeds and setup times to generate any operating time values that are in the ranges of the possible operating times and to increase the search space of the algorithms presented in this dissertation.

Then, the determination of the starting job sequence for the first stage for the problem is proposed. For the first idea, it will concern the information of the job operating times as a whole, that is, each representative of operating time of every stage is summed as the total representative of operating time of each job to find a starting job sequence for the first stage. It concerns the problem as a single model and uses the simple dispatching rules to determine the starting job sequence; for example, Kurz and Askin (2004) propose their starting job sequence by using the SPT rules and then follow by the greedy approach to distribute the jobs to the machines.

For another idea, it can use the well-known flow shop heuristics to create the starting job sequence, by a reason that it should concern the priority of the different operating times at each stage; for example, Santos *et al.* (1996) have used the Palmer, CDS, Gupta, and Dannenbring heuristics to determine their starting job sequence, and Wang and Hunsucker (2003) have used the CDS algorithm to determine their starting job sequence, as well.

Consequently, in this dissertation, the constructive algorithms are adapted and developed by using one of several basic dispatching rules and flow shop heuristics to determine the starting job sequence and then using the schedule construction approach to find the schedule solution (*see* Figure 4.3).

Algorithm 1: Constructive algorithm

Input: Relative machine speeds (v_{ij}^{t}) and setup times (ch_{ij}^{t} and s_{lj}^{t}).

Step 1: Determine the representatives of relative machine speeds $v_{ij}^{/t}$ and setup times $s_{lj}^{/t}$ for t=1,…,k.

Step 1(a): If *speed* =1 then $v_{ij}^{/t}$ = min{v_{ij}^{t}; $\forall i$ = {1, …, m^t} }, $\forall j$ = {1, …, n};
If *speed* =2 then $v_{ij}^{/t}$ = max{v_{ij}^{t}; $\forall i$ = {1, …, m^t} }, $\forall j$ = {1, …, n};
If *speed* =3 then $v_{ij}^{/t}$ = average{v_{ij}^{t}; $\forall i$ = {1, …, m^t} }, $\forall j$ = {1, …, n}.

Step 1(b): If *setup* = 1 then $s_{lj}^{/t}$ = min{ s_{lj}^{t}; $\forall l$ = {1, …, n}, ch_{ij}^{t}; $\forall i$ = {1, …, m^t} }, $\forall j$ = {1, …, n};

If *setup* = 2 then $s_{lj}^{/t}$ = max{ s_{lj}^{t}; $\forall l$ = {1, …, n}, ch_{ij}^{t}; $\forall i$ = {1, …, m^t} }, $\forall j$ = {1, …, n};

If *setup* = 1 then $s_{lj}^{/t}$ = average{ s_{lj}^{t}; $\forall l$ = {1,…, n}, ch_{ij}^{t}; $\forall i$ = {1,…, m^t} }, $\forall j$ = {1, …, n}.

Step 2: Set *speed* to be 1.

Step 3: Set *setup* to be 1.

Step 4: Determine the representatives of operating time ($O_j^{/t}$) of each job and each stage by using equation:

$$O_j^{/t} \quad = \quad \frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t} \qquad (4.1)$$

Step 5: Determine the starting job sequence ω^1 of the first stage, by using the representatives of operating time $O_j^{/t}$ to find such a job sequence with one of some modified particular rules (dispatching rules and flow shop heuristics).

Step 6: Construct a schedule output by using ***the schedule construction approach*** (*see* Approach 1)

Step 7: If *setup* < 3, then *setup* = *setup* + 1, and go to Step 4.

Step 8: If *speed* < 3, then *speed* = *speed* + 1, and go to Step 3.

Step 9: Return the best solution.

4.4.1 Dispatching Rules

The dispatching rules are the simple algorithms to construct the solution. They can be classified in a number of ways. One such classification is as follows (Holthaus and Rajendran, 1997):

1. Process-time based rules,
2. Due-date based rules,
3. Combined rules, and
4. Rules that are neither process-time based nor due-date based.

The algorithms that cover four types of dispatching rules are chosen. The Shortest Processing Time (SPT) is an example of a process-time based rule that ignores the due-date information of jobs. The SPT rule has been found to minimize the mean flow time and a good performance with respect to the mean tardiness objective and has also been observed under highly loaded conditions in the shop (Baker, 1974). Another method of a process-time based rule is the Longest Processing Time (LPT) rule, the advantage of which is to keep jobs with short processing times for later because these jobs are useful at the end for balancing the workload under the parallel-machine problem (Pinedo, 1995). The reason is to keep jobs with short processing times to be assigned and sequenced without affecting the workload balance. However, for the flexible flow shop problems with unrelated parallel machine, it is necessary to adapt the SPT and LPT rules by using the representatives of the operating times as stated above. Then, the best solution is selected among the nine combinations of relative machine speeds and setup times (*see* Approach 2 that is Step 5 in Algorithm 1)

Approach 2: Process-time based rule

Input: Representative of operating time $O_j^{/t}$ from Step 4 in Algorithm 1.

Step 1: For each job *j*, *j* = 1,…, *n*, determine its total representatives of operating time $TO_j^{/} = \sum_{t=1}^{k} O_j^{/t}$.

Step 2: <u>*Case1*</u> The Shortest Processing Time (SPT) rule:
Sort the jobs in ascending order of the $TO_j^{/}$ values, if any two jobs have the same $TO_j^{/}$ values, sort them in an arbitrary order, and set it to be ω^1, where ω^1 is a starting job sequence for the first stage.

Case2 The Longest Processing Time (LPT) rule:
Sort the jobs in descending order of the $TO_j^{/}$ values, if any two jobs have the same $TO_j^{/}$ values, sort them in an arbitrary order, and set it to be ω^1, where ω^1 is a starting job sequence for the first stage.

For due-date based rules, the Earliest Due Date (EDD) rule is proposed. Followed by the EDD rule, the next job to be processed is the one with the earliest due date. Consequently, the jobs in the starting job sequence ω^1 for the first stage are sorted according to non-decreasing due dates of the jobs.

Rules can be combined to make use of both process-time and due-date information, *e.g.* the Minimum Slack Time rule, etc. The Minimum Slack Time first (MST) rule is a variation of the EDD rule. This rule concerns the remaining slack of each job, defined as its due date minus the processing time required to process a job. Similar to the MST rule, another rule concerned in this problem is the Slack time per Processing time (S/P), where its slack time is divided by the processing time required as well (*see* Approach 3 that is Step 5 in Algorithm 1)

Approach 3: Combined rule

Input: Representative of operating time $O_j^{/t}$ from Step 4 in Algorithm 1.

Step 1: For each job j, j = 1,…, n, determine its total representatives of operating time $TO_j^{/} = \sum_{t=1}^{k} O_j^{/t}$.

Step 2: *Case1* The Minimum Slack Time first (MST) rule:
Sort the jobs in a non-decreasing order of the ($d_j - TO_j^{/}$) values, if any two jobs have the same ($d_j - TO_j^{/}$) values, sort them in an arbitrary order, and set it to be ω^1, where ω^1 is a starting job sequence for the first stage.

<u>*Case2*</u> The Slack time per Processing time (S/P) rule:
Sort the jobs in a non-decreasing order of the $(d_j - TO_j^l)/TO_j^l$ values, if any two jobs have the same $(d_j - TO_j^l)/TO_j^l$ values, sort them in an arbitrary order, and set it to be ω^1, where ω^1 is a starting job sequence for the first stage.

Finally, a rule that is neither process-time based nor due-date based concerned in this research is the Earliest Release Date first (ERD) rule which is equivalent to the well-known the First-In-First-Out (FIFO) rule. The ERD rule, in a sense, minimizes the variation in the waiting times of the jobs at a machine (Pinedo and Chao, 1999). Thus, the jobs in the starting job sequence ω^1 for the first stage are sorted according to non-decreasing release dates of the jobs.

Consequently, the basic dispatching rules given by Shortest Processing Time (SPT), Longest Processing Time (LPT), Earliest Release Date first (ERD), Earliest Due Date first (EDD), Minimum Slack Time first (MST), and Slack time per Processing time (S/P) rules are considered to arrange the jobs in the staring job sequence for the first stage in the constructive algorithms. Moreover, they are used mainly for comparison purposes and to have a broad spectrum of solutions in the initial solution (or population) of the iterative algorithm as proposed in Section 4.7.

4.4.2 Flow Shop Heuristics

From Section 4.4.1, it is noticed that the dispatching rules concern the total operating times for all stages instead of the operating times in each stage. Consequently, now it has moved the dispatching rules to the flow shop heuristics that concern the operating time in each stage.

The flow shop heuristics can be classified into four groups as follows (Hejazi and Saghafian, 2005):

1. Heuristics based on slope indices,
2. Heuristics based on Johnson’s Rule,
3. Heuristics based on both slope indices and Johnson’s Rules, and
4. Otherwise.

The Palmer Rule is an approach that assigns a weight or “index” to every job and then arranges the sequence by sorting the jobs according to the assigned index. His idea is to give priority to jobs that have a tendency of progressing from short times to long times as they move through the stages. It is noted that his algorithm behinds the concept of the LPT algorithm at the last stage and the SPT algorithm at the first stage. This idea has been used in later papers; for example, Gupta proposes a modification of Palmer's slope index which exploits some similarities between scheduling and sorting problems.

For the CDS rule, it is in the class of heuristics based on Johnson’s algorithm. In this case, several schedules are constructed, and the best one is given as a result. The heuristic is known as CDS and builds $k-1$ schedules (where k is the total number of stages) by clustering the k original machines into two virtual machines and solving the generated two machine problem by repeatedly using Johnson’s rule.

Dannenbring’s heuristic is a method that mixes the previous ideas of Johnson’s algorithm and Palmer's slope index. In this case, a virtual two machine problem is defined as it is in the CDS heuristic, but instead of directly applying Johnson’s algorithm over the processing times, two weighting schemes are calculated, and then Johnson’s algorithm is applied. The weighting schemes give the processing times for the jobs in the two virtual machines.

The NEH heuristic is based neither on Johnson's algorithm nor on slope indexes. It is based on the idea that jobs with high processing times on all the machines should be scheduled as early in the sequence as possible.

Notice that the first four algorithms of flow shop makespan heuristics, namely the Palmer, CDS, Gupta, and Dannenbring methods, try to minimize makespan while the insertion heuristic, the NEH algorithm, can be used for any regular optimization criterion and for the multi-criteria problem under consideration as well. Consequently, the well-known flow-shop makespan heuristics, which are Palmer, CDS, Gupta, and Dannenbring as well as the insertion heuristic by Nawaz, Enscore, and Ham, are adapted.

4.4.2.1 Palmer

A heuristic developed by Palmer (1965), in an effort to use Johnson's rule, proposes a *slop order index* to sequence the jobs on the machines based on the processing times. The idea is to give priority to jobs that have a tendency of progressing from short times to long times as they move through the stages. It means that the first stage sequence can be generated based upon a non-increasing order of the slope indices.

Let S (j) be the slope index for job j and O_j^t be the operating time of job j at stage t. Palmer's slope index is calculated as follows:

$$S(j) = -\sum_{t=1}^{k} \left\{ [k-(2t-1)]O_j^t \right\} \tag{4.2}$$

To illustrate Palmer's method for use in the flow shop environment, the example given in Table 4.1 will be utilized.

Table 4.1 Standard processing times and due date for every job of a three-stage flow shop problem

Job *j*	1	2	3	4	5
ps_j^1	11	13	20	9	11
ps_j^2	12	15	10	12	18
ps_j^3	20	18	12	20	15
d_j	40	40	75	45	60

Since there is only one machine for every stage, the operating times for every job are equal to its standard processing times ($O_j^t = ps_j^t$). The slope indices for the five jobs are now calculated as follows:

$$S(j) \quad = - [2O_j^1 + 0O_j^2 - 2O_j^3], \textit{ i.e.}$$

1. $S(1)$ $= -2(11) - 0(12) + 2(20)$ $= 18;$
2. $S(2)$ $= -2(13) - 0(15) + 2(18)$ $= 10;$
3. $S(3)$ $= -2(20) - 0(10) + 2(12)$ $= -16;$
4. $S(4)$ $= -2(9) - 0(12) + 2(20)$ $= 22;$
5. $S(5)$ $= -2(11) - 0(18) + 2(15)$ $= 8.$

Palmer's heuristic sequences the jobs in non-increasing order of the slope indices. For the job set in Table 4.1, this heuristic yields the starting job sequence ω^1 = [4 1 2 5 3] for the first stage. Palmer's heuristic yields a makespan value of 106. Using a 0-1 mixed integer programming formulation, one can confirm that 106 is the optimal makespan value as well.

Now, the modified Palmer's method (in the following denoted by PAL) for the flexible flow shop problem with unrelated parallel machines and sequence-dependent setup times is developed as follows.

Let ps_j^t be the standard processing time of job j at stage t, $v_{ij}^{/t}$ be the representative of the relative machine speeds on machine i at stage t for job j, and $s_{lj}^{/t}$ be the representative of the setup time between job *l* and job *j* at stage *t*. Then, $S(j, v_{ij}^{/t}, s_{lj}^{/t})$ denotes the slope index for job *j* at the relative machine speed $v_{ij}^{/t}$ and setup time $s_{lj}^{/t}$. PAL's slope index for the flexible flow shop problem with unrelated parallel machines and setup times is calculated as follows:

$$S(j, v_{ij}^{/t}, s_{lj}^{/t}) = -\sum_{t=1}^{k} \left\{ [k-(2t-1)]\left(\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t}\right) \right\} \tag{4.3}$$

Since the processing times and the setup times for every job are dependent on the machine and the previous job, respectively, the representative of the operating time $O_j^{/t}$ of job *j* at stage *t* in Palmer's method adaptation is the sum of the processing time $ps_j^t / v_{ij}^{/t}$ plus the setup time $s_{lj}^{/t}$. To construct a schedule for the overall problem, set $v_{ij}^{/t}$ to be the minimum, maximum, and average relative machine speeds and $s_{lj}^{/t}$ to be the minimum, maximum, and average setup times regardless for the machine and the previous job. All these nine combinations of relative machine speeds and setup times will be used as described in Algorithm 1, and finally, the solution with the best objective function value obtained by all different relative machine speeds and setup times is taken.

4.4.2.2 Campbell, Dudek, and Smith

Campbell *et al.* (1970) develop one of the most significant heuristic methods for the makespan flow shop scheduling problem, known as the CDS algorithm. Its strength lies in two properties: (1) it uses Johnson's rule in a heuristic fashion and (2) it generally creates several schedules from which a "best" schedule can be chosen. In so doing, *k* – 1 subproblems are created, and Johnson's rule is applied to each of the subproblems. Thus, *k* – 1 job sequences are generated. Since Johnson's algorithm is a two-stage algorithm, a *k*-stage problem must be collapsed into a two-stage problem. Let *g* be a counter for the *k* – 1 sub-problems, the operating times for the "first" stage are denoted as *a*(*j*, *g*), where *j* denotes the job, and *g* denotes the *g*-th subproblem. Similarly, *b*(*j*, *g*) denotes the "second" stage operating times of job *j* and sub-problem *g*. Given these notations, the operating times are calculated by the following two formulas:

$$a(j,g) = \sum_{t=1}^{g} O_j^t \qquad (4.4)$$

and

$$b(j,g) = \sum_{t=k-g+1}^{k} O_j^t \qquad (4.5)$$

For each of the subproblems, Johnson's algorithm provides a job sequence using the values *a*(*j*, *g*) and *b*(*j*, *g*). Once Johnson's sequence is created, the problem is then returned to the consideration of all *k* stages.

Again, due to the unrelated parallel machines, the constructed processing time for the "first" stage is denoted as $a(j,g,v_{ij}^{/t},s_{lj}^{/t})$, where *j* denotes the job, *g* denotes the *g*-th subproblem, and $v_{ij}^{/t}$ and $s_{lj}^{/t}$ are the representatives of the relative machine speed and setup time, respectively. Similarly, $b(j,g,v_{ij}^{/t},s_{lj}^{/t})$ denotes the "second" stage processing time. Given these definitions, the constructed processing times are calculated according to the following two equations:

$$a(j,g,v_{ij}^{/t},s_{lj}^{/t}) = \sum_{t=1}^{g} \left(\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t}\right) \qquad (4.6)$$

and

$$b(j,g,v_{ij}^{/t},s_{lj}^{/t}) = \sum_{t=k-g+1}^{k} \left(\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t}\right) \qquad (4.7)$$

To generate the starting job sequence, Johnson's ordering is created, and the problem is then returned to the consideration of *k* stages by calling Approach 1 for all nine combinations of relative machine speeds and setup times as considered in Algorithm 1.

4.4.2.3 Gupta

Gupta (1971) provides an algorithm, denoted by GUP, in a similar manner as algorithm PAL by using a slope index. Denote *G*(*j*) as the slope index generated by Gupta's method for job *j*. Then *G*(*j*) is calculated as follows:

$$G(j) = \frac{e_j}{\min\limits_{1 \le g \le k-1} \{O_j^g + O_j^{g+1}\}} \tag{4.8}$$

;where $e_j = \begin{cases} 1 & if \quad O_j^1 > O_j^k \\ -1 & if \quad O_j^1 \le O_j^k \end{cases}$.

After calculating *G*(*j*) for all jobs, the jobs are subsequently ranked in a non-decreasing order of the slope indices.

Under the Gupta adaptation rule, let $G(j, v_{ij}^{/t}, s_{lj}^{/t})$ be the slope index of algorithm GUP for job *j* at relative machine speed $v_{ij}^{/t}$ and setup time $s_{lj}^{/t}$. The slope index of algorithm GUP for the flexible flow shop with unrelated parallel machines and setup times is then calculated from:

$$G(j, v_{ij}^{/t}, s_{lj}^{/t}) = \frac{e_j}{\min\limits_{1 \le g \le k-1} \{(\frac{ps_j^g}{v_{ij}^{/g}} + s_{lj}^{/g}) + (\frac{ps_j^{g+1}}{v_{ij}^{/g+1}} + s_{lj}^{/g+1})\}} \tag{4.9}$$

;where $e_j = \begin{cases} 1 & if \quad (\frac{ps_j^1}{v_{ij}^{/1}} + s_{lj}^{/1}) > (\frac{ps_j^k}{v_{ij}^{/k}} + s_{lj}^{/k}) \\ -1 & if \quad (\frac{ps_j^1}{v_{ij}^{/1}} + s_{lj}^{/1}) \le (\frac{ps_j^k}{v_{ij}^{/k}} + s_{lj}^{/k}) \end{cases}$

4.4.2.4 Dannenbring

Like PAL’s rule, Dannenbring (1977) develops a method by using Johnson’s algorithm as the foundation. Furthermore, the CDS and PAL algorithms are also exhibited. Dannenbring constructs only one two-stage problem, but the processing times for the constructed jobs reflect the behavior of PAL’s slope index. In the following, this method is denoted by DAN. Denote *a*(*j*) and *b*(*j*) as the operating times for the constructed two-stage problem. The calculations of *a*(*j*) and *b*(*j*) are as follows:

$$a(j) = \sum_{t=1}^{k} (k - t + 1)(O_j^t) \tag{4.10}$$

and

$$b(j) = \sum_{t=1}^{k} t \times O_j^t \tag{4.11}$$

After calculating *a*(*j*) and *b*(*j*) for all jobs, the jobs are subsequently ranked by applying Johnson’s algorithm to generate the starting job sequence for stage one.

Under the DAN adaptation rule, the operating times for the flexible flow shop problem with unrelated parallel machines and setup times are calculated as follows:

$$a(j, v_{ij}^{/t}, s_{lj}^{/t}) = \sum_{t=1}^{k} \left\{ (k - t + 1)\left(\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t}\right) \right\} \tag{4.12}$$

and

$$b(j, v_{ij}^{/t}, s_{lj}^{/t}) = \sum_{t=1}^{k} \left\{ t \times \left(\frac{ps_j^t}{v_{ij}^{/t}} + s_{lj}^{/t}\right) \right\} \tag{4.13}$$

4.4.2.5 Nawaz, Enscore, and Ham

Nawaz *et al.* (1983) develop probably the best constructive heuristic method for the permutation flow shop makespan problem, called the NEH algorithm. It is based on the idea that a job with a high total operating time on the machines should be placed first at an appropriate relative order in the sequence. Thus, jobs are sorted in non-increasing order of their total operating time requirements. The final sequence is built in a constructive way, adding a new job at each step and finding the best partial solution (*see* Approach 4).

Approach 4: Nawaz, Enscore, and Ham rule

Input: Representative of operating time $O_j^{/t}$ from Step 4 in Algorithm 1.

Step 1: The total operating times for job *j* is calculated using the formula :

$$P_j = \sum_{t=1}^{k} O_j^{/t} \text{ , } j = 1, \ldots, n. \qquad (4.14)$$

Step 2: The *n* jobs are sorted in non-increasing order of their total operating time P_j on the machines. Then the first two jobs (those with largest P_j) are taken, and the two possible schedules containing them are evaluated. The sequence with better objective function value is taken for further consideration.

Step 3: Take every remaining job in the sorted list calculated in Step 2 and find the best schedule by placing it at all possible positions in the sequence of jobs that are already scheduled. For example, if [*j*1 *j*2 *j*3] is the current sequence of scheduled jobs, and job *r* is the remaining job with largest P_r in the sorted list, then job *r* could be placed at four positions: [*r* *j*1 *j*2 *j*3], [*j*1 *r* *j*2 *j*3], [*j*1 *j*2 *r* *j*3] or [*j*1 *j*2 *j*3 *r*]. The sequence with best objective function value among the four considered is selected for further extension.

For example, the NEH algorithm inserts a third job into the previous partial solution in such a way that the resulting sequence gives the best objective function value (*i.e.* the relative position of the previous job sequence remains fixed). The algorithm repeats the process for the remaining jobs according to the initial ordering of the total operating time requirements.

Again, to apply the NEH algorithm to the flexible flow shop problem with unrelated parallel machines, the total operating times for calculating the starting job sequence for the first stage are calculated according to Step 4 of Algorithm 1 for the nine combinations of relative machine speeds and setup times. Contrary to the algorithms presented before, the NEH algorithm constructs job sequences by considering the minimization of the convex combination of the makespan and the number of tardy jobs.

4.5 Improvement Algorithms

Unlike constructive algorithms, improvement heuristics start with an already built schedule and try to improve it by some given procedures. As proposed the algorithms in the previous section, after obtaining a good solution in efficient time, it can obtain incremental improvement (called an *improvement approach*) to an existing solution by neighborhood exchanges or local search (Winston and Venkataramanan, 2003). Their use is necessary since the constructive algorithms (especially some algorithms that are adapted from pure makespan heuristics and some dispatching rules such as the SPT and LPT rules) do not consider due dates (and therefore, they do not consider the minimization of the number of tardy jobs).

This section proposes the improvement algorithms for the current solution by using some algorithms, namely pairwise interchange and shift move strategies, to improve the overall function value by dealing mainly with the due date criterion (*see* Figure 4.4). The first improvement idea (called a *shift move approach* or an *insertion approach*) is based on the fact that the tardy job should be shifted to perform earlier or shifted to perform later if the solution can improve. The next improvement idea (called a *pairwise interchange approach*) is based on the fact that the tardy job may be exchanged with a job that is assigned ahead or later.

Consequently, in this dissertation, the iterative algorithms described in the following and in Section 4.6 are based on the shift move (SM) and the pairwise interchange (PI) neighborhoods.

The SM neighborhood repositions a chosen job. This means that an arbitrary job π_r at position r is shifted to position i, while leaving all other relative job orders unchanged. If $1 \le r < i \le n$, it is called a *right shift* and yields $\pi' = [\pi_1 \ \ldots \ \pi_{r-1} \ \pi_{r+1} \ \ldots \ \pi_i \ \pi_r \ \pi_{i+1} \ \ldots \ \pi_n]$. If $1 \le i < r \le n$, it is called a *left shift* and yields $\pi' = [\pi_1 \ \ldots \ \pi_{i-1} \ \pi_r \ \pi_I \ \ldots \ \pi_{r-1} \ \pi_{r+1} \ \ldots \ \pi_n]$. For instance, assume that randomly one starting job sequence solution for the first stage in the current generation is selected, say [4 9 8 7 3 1 6 2 5], and then randomly a couple of job positions for performing the shift is selected, *e.g.* positions 2 and 7 (in this case, it is a right shift). The new starting job sequence solution will be [4 8 7 3 1 6 ***9*** 2 5]. However, if positions 7 and 2 are randomly selected (*i.e.* it is a left shift), the new starting job sequence solution will be [4 ***6*** 9 8 7 3 1 2 5]. In the SM neighborhood, the current starting job sequence solution has $(n-1)^2$ neighbors.

The PI neighborhood exchanges a pair of arbitrary jobs π_r, and π_i, where $1 \le i, r \le n$ and $i \ne r$. Such an operation swaps the jobs at positions r and i, which yields $\pi' = [\pi_1 \ \ldots \ \pi_{r-1} \ \pi_I \ \pi_{r+1} \ \ldots \ \pi_{i-1} \ \pi_r \ \pi_{i+1} \ \ldots \ \pi_n]$. For example, assume that the current starting job sequence solution is [4 9 8 7 3 1 6 2 5], and then randomly the couple of job positions to be exchanged is selected, *e.g.* positions 1 and 3. Thus, the new starting job sequence solution will be [***8*** 9 ***4*** 7 3 1 6 2 5]. In the PI neighborhood, the current starting job sequence solution has $n\times(n-1)/2$ neighbors.

In order to find a satisfactory solution of the due date problem, the fast polynomial heuristics are applied on the starting job sequence that gives the best schedule solution from the previous algorithms in Section 4.4 by investigating either the above SM algorithm as an improvement mechanism based on the idea that it will consider the jobs that are tardy in a left-to-right scan and move each of them left and right or the PI algorithm, where a tardy job is selected and swapped to different job positions left and right, and either to two randomly determined positions (denoted by the number “2”) or to all n–1 possible positions (denoted by the letter “A”). The best schedule among the generated neighbors is then taken as the result. The algorithm is as follows:

Algorithm 2: Improvement algorithm

Input: The best starting job sequence solution ω^{1*} from Algorithm 1; where ω^{1*} = [ω^{1*} [1] … ω^{1*} [n]]

Step 1: Set the dummy starting job sequence π to be ω^{1*}

Step 2: Check the first job j in the best starting job sequence ω^{1*}, if the job j is not tardy in the dummy starting job sequence π, go to Step 5.

Step 3: Interchange (*i.e.* apply 2-PI or A-PI) or shift (*i.e.* apply 2-SM or A-SM) the chosen job j in the dummy starting job sequence π and evaluate the objective function values by using ***the schedule construction approach*** (*see* Approach 1).

Step 4: Update the current dummy starting job sequence π, if the objective function values improve.

Step 5: Update the job list $\omega^* = \omega^* - \{j\}$.

Step 6: Go to Step 2 until the best starting job sequence ω^{1*} is empty.

Step 7: Return the best sequence solution π.

Since every tardy job in the job sequence is considered at most once, the complexity of the 2-PI and 2-SM procedures is $O(n)$, and the complexity of the A-PI and A-SM procedures is $O(n^2)$.

4.6 Iterative Algorithms

Those algorithms from both Section 4.4 and Section 4.5 tend to get trapped in a local optimal solution, but it can apply other general heuristics with the development of solution methodologies based on an artificial intelligence method (called an *iterative approach*) (Winston and Venkataramanan, 2003) (*see* Figure 4.5).

Hence, to obtain a near-optimal solution, *iterative* algorithms (or *metaheuristic* algorithms), have also been proposed by many researchers; for example, Gourgand *et al.* (1999) present several simulated annealing (SA)-based algorithms for the flexible flow shop problem. A specific neighborhood is used, and the authors apply the methods to a realistic industrial problem. Jin *et al.* (2006) consider the flexible flow shop with identical parallel machines. They propose two approaches to generate the initial job sequence and use an SA algorithm to improve it. It can be seen that an SA algorithm has been successfully applied to various combinatorial optimization problems. For an extensive survey of the theory and applications of the SA algorithm, *see* Koulamas, Antony, and Jaen (1994). Furthermore, Nowicki and Smutnicki (1998) propose a tabu search (TS) algorithm for the flexible flow shop makespan problem. A genetic algorithm (GA) has been widely used in many previous works for the flow shop makespan problem, *see e.g.* Werner (1984) and Reeves (1995). Cheng *et al.* (1995) address the earliness/tardiness scheduling problem with identical parallel machines, and they apply a GA to solve their problem. Ruiz *et al.* (2005) use a GA approach to deal with the permutation flow shop scheduling problem with sequence-dependent setup times. However, little research has been done for flexible flow shop scheduling problems, especially for the general case with unrelated parallel machines and setup times (*see* for instance the recent review on scheduling with setup times by Allahverdi *et al.* (2008)).

In this dissertation, three well-known iterative algorithms for the heuristic solution of the problem are considered, namely simulated annealing, tabu search, and genetic algorithms. All these iterative algorithms work with the starting job sequence for the first stage. If a new job sequence has been generated, the schedule construction procedure described in Section 4.3 is applied, and the objective function value of this schedule is used for evaluating the job sequence for the first stage.

4.6.1 Simulated Annealing Algorithm

A simulated annealing (SA) algorithm is an enhanced version of local optimization or an iterative search method, in which an initial solution is repeatedly improved by making small local alterations until no such alteration yields a better solution. It is developed by Kirkpatrick, Gelatt, and Vecchi (1983).

An SA algorithm is inspired by the annealing of metals, in which annealing refers to the process which occurs when physical substances, such as metals, are raised to a high energy level (melted) and then gradually cooled until some solid state is reached. The goal of this process is to reach the lowest energy state. In this process, physical substances usually move from higher energy states to lower ones if the cooling process is sufficiently slow, so a minimization naturally occurs. Due to natural variability, however, there is some probability at each stage of the cooling process that a transition to a higher energy state will occur. As the energy state naturally declines, the probability of moving to a higher energy state decreases.

In general, an SA algorithm is a stochastic optimization method for minimizing a function f over a discrete domain S. Starting from an initial solution $s \in S$, an SA algorithm generates a new solution $s' \in S$ in the neighborhood of the initial solution s by using a suitable operator. Concerning the neighborhood, both a shift move (SM) neighborhood (*i.e.* a job at an arbitrary position is selected and reinserted at some other position), and a pairwise interchange (PI) neighborhood (*i.e.* two arbitrary jobs are selected and interchanged) are considered (*see* Section 4.5).

The objective function value $f(s')$ of the new solution is then compared to the objective function value $f(s)$ of the initial solution (remember that the objective function value of the full schedule generated from the starting job sequence for the first stage is taken). The change in the objective function value, $\delta = f(s') - f(s)$, is calculated. If the objective function value decreases ($\delta < 0$), the new solution is automatically accepted and it becomes the point from which the search will continue. If the objective function value increases ($\delta \geq 0$), then a solution with a larger objective function may also be accepted with a probability, usually determined by a function, exp $(-\delta/T)$, where $T \in \Re$ is a control parameter of an SA algorithm called the *temperature*. The probability of acceptance the higher values decrease as T decreases. At high temperature, the search is almost random, whereas at low temperature the search becomes almost greedy. At zero temperature, the search becomes totally greedy, that is, only good moves are accepted. Consequently, the role of the temperature T is significant in the operation of an SA algorithm. This temperature, which is simply a positive number, is initialized to a value T_0 at the beginning of the procedure and is periodically reduced every NT iterations, where NT denotes the epoch length, so that it moves gradually from a relatively high value to near zero as the algorithm progresses according to a function referred to as the *cooling schedule*. An SA approach is shown in Algorithm 3.

Algorithm 3: Simulated annealing

Input: The initial temperature T_0, final temperature T_f, cooling temperature rate, and epoch length NT.

Step 1: Select randomly an initial solution $s_0 = [s_0[1] \ \ldots \ s_0[n]]$ and evaluate the objective function value $f(s_0)$ by setting a starting job sequence ω^1 for the first stage in Algorithm 1 to a solution s_0.

Step 2: Set a current solution s and a best solution s_{best} to an initial solution s_0.

Step 3: Set a temperature control parameter T to T_0.

Step 4: Generate a new solution s' in the neighborhood of the current solution s by using a suitable operator (*e.g.* PI and SM neighborhoods) (*see* Section 4.5).

Step 5: Evaluate the objective function value $f(s')$ by setting a starting job sequence ω^1 in ***the schedule construction approach*** (*see* Approach 1) to a solution s'.

Step 6: Update a best solution $s_{best} = s'$ if the objective function value $f(s') < f(s_{best})$.

Step 7: Update a current solution $s = s'$ and Go to Step 11 if $\delta < 0$, where $\delta = f(s') - f(s)$.

Step 8: Determine a probability function by using equation exp $(-\delta/T)$.

Step 9: Random a number uniformly distributed in the interval [0, 1].

Step 10: If RANDOM ~U[0,1] < a probability function exp $(-\delta/T)$, update a current solution $s = s'$.

Step 11: Go to Step 4, until number of iteration is equal to an epoch length *NT*.

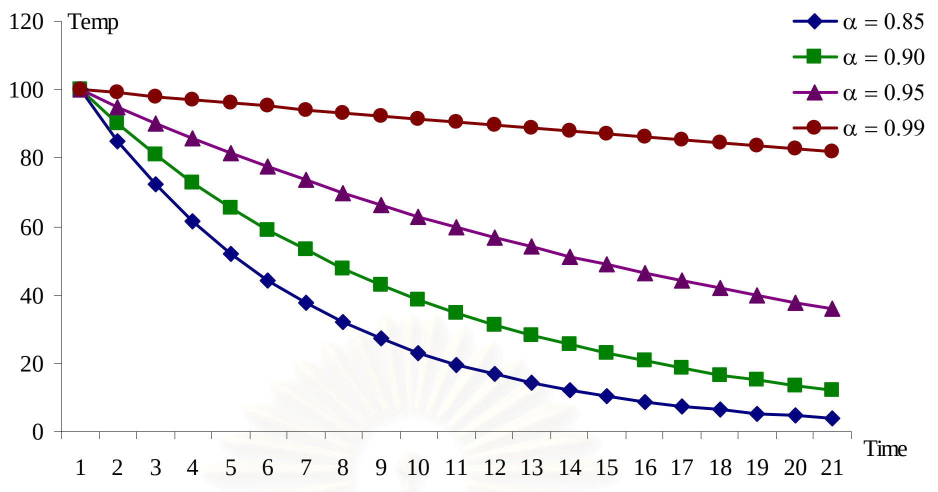
Step 12: Reduce the temperature by using cooling schedule (*see* Section 4.6.1.1) given in a specific cooling rate.

Step 13: Reset number of iteration processed, and go to Step 4 again with a new temperature control until the temperature drops bellow the final temperature T_f or other stopping criteria.

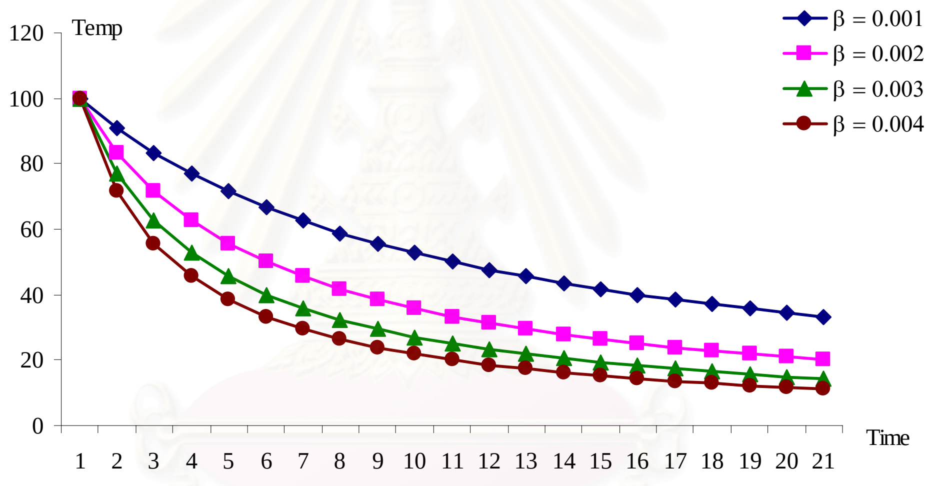
4.6.1.1 Cooling Schedule

The cooling schedule governs how likely the algorithm is to accept a bad transition as a function of the temperature T at each iteration. At the beginning of the search, the algorithm is eager to use randomness to explore the search space widely, so the probability of accepting a negative transition is high by using higher temperature. As the search progresses, the temperature is decreased, thus the probability of accepting will gradually decrease, converging to a simple iterative improvement algorithm.

$T_{new} = \alpha T_{old}$



$T_{new} = T_{old}/(1+ \beta T_{old})$



$T_{new} = T_{old} - \gamma T_0$

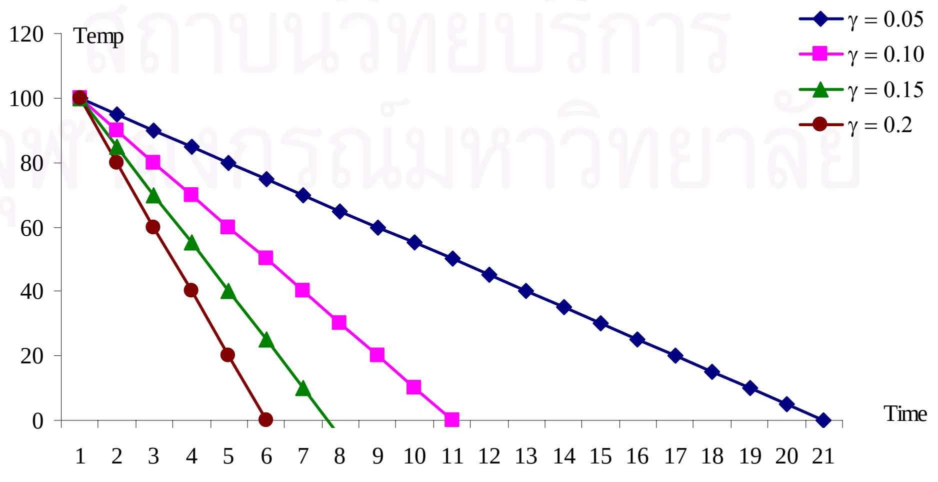


Figure 4.6 The patterns of cooling schedules

There are three most widely used cooling schedules (Bräsel *et al.*, 2006): (1) the geometric reduction schedule using the function $T_{new} = \alpha T_{old}$, (Kirkpatrick *et al.*, 1983); (2) the schedule suggested by Lundy and Mees (1986) using the relation $T_{new} = T_{old}/(1+ \beta\, T_{old})$; and (3) the linear reduction schedule using the function $T_{new} = T_{old} - \gamma T_0$ (Winston and Venkataramanan, 2003). Figure 4.6 shows the three cooling schedules assuming the initial temperature to be 100.

The scheme that follows a geometric law, which one of the most often used, corresponds to an exponential decay of the temperature. It is similar to the cooling schedule suggested by Lundy and Mees (1986) in which it opposes to a linear decay. However, the latter provides fast cooling in the early iterations and slower cooling at latter iterations. Consequently, at the beginning the search will explore the search space, while at the end the search will exploit to the local minimum. Nearchou (2004) shows that the performance of the latter is found superior to that of the first scheme. For the linear reduction, it is the basic of the cooling schedule, but it will drop in temperature to below zero faster than the others, that is, it probably entraps the local optimal as well.

4.6.1.2 Termination Condition

In an SA approach, the temperature is reduced to a smaller temperature when the best objective function value found so far is not updated for a predetermined number of iterations, and it is also reduced when *NT* iterations have been performed (*i.e.* at the end of an epoch length). The procedure is terminated when the temperature becomes equal to or less than zero or when other criteria reach.

4.6.2 Tabu Search Algorithm

A tabu search (TS) algorithm, initially developed by Glover (1986), is an iterative improvement approach designed to avoid terminating prematurely at a local optimum for solving combinatorial optimization problems. Similar to a simulating annealing algorithm, a TS algorithm is based on the idea of exploring the solution space of a problem by moving from one region of the search space to another in order to look for a better solution. The function transforming a solution into another solution is usually called *a move*. For any solution $s \in S$, a subset of moves applicable to it is defined. This subset of moves generates the *neighborhood* $\aleph(s)$ of s (*see* Section 4.5). Starting from an initial solution s, a TS algorithm iteratively moves from the current solution s to the best solution $s^* \in \aleph(s)$ even though s^* is worse than the current solution s, until some stopping criterion is satisfied (*see* Algorithm 4).

Selecting the best move s^* (which may or may not improve the current solution s) is based on the supposition that good moves are more likely to reach the optimal or near-optimal solutions. The set of admissible solutions attempted at a particular iteration forms a *candidate list*. A TS algorithm selects the best solution from the candidate list. Candidate list size is a trade-off between quality and performance.

However, to escape from a local optimum, an SA algorithm accepts an inferior solution, which may lead to better solutions later by using an acceptance probability. In contrast, a TS algorithm allows the search to move to the best solution s^* among a set of candidate moves $\aleph(s)$ as defined by the neighborhood structure, although it can move to a neighbor with a worse objective function value. Nevertheless, subsequent iterations may cause the search to move repeatedly back to the same local optimum. In order to prevent cycling back to recently visited solutions, it should be *forbidden* or declared *tabu* for a certain number of iterations. This is accomplished by keeping the attributes of the forbidden moves in a list, called the *tabu list*. The size of the tabu list, called the *tabu tenure*, must be large enough to prevent cycling, but small enough not to forbid too many moves.

Additionally, an aspiration criterion is defined to deal with the case in which a move leading to a new best solution is tabu. If a current tabu move satisfies the *aspiration criterion*, its tabu status is canceled, and it becomes an allowable move.

The use of the aspiration criterion allows the TS algorithm to lift the restrictions and intensify the search into a particular solution region.

Algorithm 4: Tabu search

Input: The candidate size (or size of neighborhoods) and tabu list size.

Step 1: Select randomly an initial solution $s_0 = [s_0[1] \ \ldots \ s_0[n]]$ and evaluate the objective function value $f(s_0)$ by setting a starting job sequence ω^1 for the first stage in Algorithm 1 to a solution s_0.

Step 2: Set a current solution s and a best solution s_{best} to an initial solution s_0.

Step 3: Generate a set of candidate solutions $\aleph(s)$ in the neighborhood of the current solution s by using a suitable operator (*e.g.* PI and SM neighborhoods) (*see* Section 4.5).

Step 4: Evaluate the objective function value $f(s')$ by setting a starting job sequence ω^1 for the first stage in ***the schedule construction approach*** (*see* Approach 1) to a solution s', where $s' \in \aleph(s) - \mathtt{L}$ and $\mathtt{L}$ is a set of tabu list.

Step 5: Select a best solution s^* among a set of candidate moves $\aleph(s)$.

Step 6: Update a best solution $s_{best} = s^*$ if the objective function value $f(s^*) < f(s_{best})$.

Step 7: Update a current solution $s = s^*$.

Step 8: Declare a current solution s in a tabu list $\mathtt{L}$ for a certain number of iterations.

Step 9: Go to Step 3, until the stopping criteria reach.

4.6.3 Genetic Algorithm

A genetic algorithm (GA) approach is an iterative heuristic based on Darwin’s evolutionary theory about “survival of the fittest and natural selection”. It belongs to the evolutionary class of artificial intelligent (AI) techniques. Holland

(1975) proposes some basic principles of natural evolution as a methodology to solve decision-making problems. For the classical flow shop problem, the first genetic algorithm has been given by Werner (1984).

The GA approach is characterized by a parallel search of the state space in contrast to a point-by-point search by conventional techniques. The parallel search is achieved by keeping a set of possible solutions, called a *population*. An individual in the population is a string of symbols. The GA starts with the initial generation of artificial individuals which are often created randomly (*see* Algorithm 5). Each symbol is called a *gene*, and each string of genes is termed as a *chromosome*. The individuals in the population are evaluated by a measure, called the *fitness*, to describe quantitatively how well the individual masters its task. The *initial population* is then evolved into different populations over a number of *generations* through the use of two types of *genetic operators*: (1) unary operators, *i.e.* mutation and inversion, which change the genetic structure of a single chromosome, and (2) a higher-order operator, referred to as crossover which consists of obtaining new individual(s) by combining the genetic material from two selected parent chromosomes. When applying crossover, two individuals (parents) are selected from the population, and new solution(s), called the *offspring*, is (are) created. Mutation creates a new solution by a random change on a selected individual. The genetic operators are applied to randomly selected parents to generate new offspring. Then the new population is selected out of the individuals of the current population and the new generated chromosomes (Gen and Cheng, 1997)

Algorithm 5: Genetic algorithm

Input: Number of population *pop_size*, Crossover rates p_c, and Mutation rates p_m.

Step 1: Generate randomly a number of solutions $P(s)$ and evaluate the objective function value $f(s)$ by setting a job list ω^0 in Algorithm 1 to a solution s, where $s \in P(s)$.

Step 2: Create offspring solutions $O^c(s)$ by using crossover operator (*see* Section 4.6.3.2).

Step 3: Create offspring solutions $O^m(s)$ by using mutation operator (*see* Section 4.6.3.3).

Step 4: Evaluate the objective function value offspring $O(s)$ by setting a job list ω^0 in ***the schedule construction approach*** (*see* Approach 1) to a solution s, where $s \in O(s)$ and $O(s) = O^m(s) \cup O^c(s)$.

Step 5: Update a best solution $s_{best} = s^*$ if the objective function value $f(s^*) < f(s_{best})$, where $s^* \in P(s) \cup O(s)$.

Step 6: Selection a new population from a current population $P(s)$ and offspring solution $O(s)$.

Step 7: Go to Step 2, until the stopping criteria reach.

The application of the GA approach requires the representation of a solution, the choice of genetic operators (crossover and mutation), an evaluation function, a selection mechanism, and the determination of genetic parameters (population size as well as crossover and mutation rates).

4.6.3.1 Encoding Scheme

For the representation, consideration of a job permutation is straightforward and widely used in many previous works on the GA approach for the flow shop problem, *see* Werner (1984). Thus, in this dissertation, a permutation-based code (or a *job code*) using integers as the chromosome coding scheme is applied. For instance, one chromosome of an example with nine jobs can be coded as the job sequence [9 3 7 8 2 6 5 1 4].

4.6.3.2 Crossover

A crossover operation is a mechanism for probabilistic inheritance of useful information from two fit individuals to offspring. The main idea is that the genetic information of a good solution is spread over the entire population. Thus, the best solution can be obtained by thoroughly combining the chromosomes in the population. Crossover operation achieves recombination of the genetic material. The recombination process includes domain specific knowledge to enforce the inheritance of desirable features from individuals of current population.

The PMX (partially mapped crossover) method may be the most popular crossover operator when operating with permutations. Firstly, choose two parents P1 and P2, *e.g.* P1 = [1 2 3 4 5 6 7 8 9] and P2 = [9 3 7 8 2 6 5 1 4], and two cutting sites along the string are randomly chosen, *e.g.* 3 and 7. The substrings defined by the two cutpoints are called the *mapping sections*. Secondly, exchange the two substrings between the parents to produce protochildren, which yields [1 2 3 |8 2 6 5| 8 9] and [9 3 7 |4 5 6 7| 1 4]. It is clear that protochildren will often lead to infeasible solutions. Then, one needs to determine the mapping relationship between the two mapping sections, and finally, it legalizes the offspring using this mapping relationship. In the first protochild, it can map the two infeasible genes 2 and 8 outside the mapping section, by using the mapping swaps, for instance, 2 in the first protochild's mapping section can be mapped to 5 in the second protochild's mapping section corresponding to the position. It does however not finish, because 5 is in the first protochild's mapping section as well. Again, 5 in the first protochild can be mapped to 7 in a similar way. At last, 2 in the first protochild can be swapped to 7. Similarly, 8 in the first protochild can be mapped to 4. Consequently, the first offspring is [1 ***7*** 3 | 8 2 6 5| ***4*** 9]. Then, the second offspring is analogously created as [9 3 ***2*** | 4 5 6 7| 1 ***8***].

The OPX (combined order and position-based crossover) method may be a good crossover choice, in which it creates feasible solutions like PMX and combines the characteristics of OX and PBX as well. It will create the first offspring based on OX, whereas the second offspring is characterized by PBX. Again two parents P1 and P2 are randomly selected, and consider the same example as for PMX above. Then, randomly select a substring from the first parent, *e.g.* [1 2 3 |4 5

6 7| 8 9]. Copy the substring into the first protochild corresponding to the first parent position, *e.g.* [_ _ _ |4 5 6 7| _ _]. Then, delete all the symbols from the second parent which are already in the substring and place its symbols into the unfixed positions in the first protochild from left to right according to the second parent order, *e.g.* [***9 3 8*** |4 5 6 7| ***2 1***]. To create the second offspring, the second protochild is created by copying the symbols from the second parent, where the jobs are the same as the symbols in the substring in the corresponding position, *e.g.* [_ _ 7 _ _ 6 5 _ 4]. Then, place the symbols from the first parent into the unfixed positions in the second protochild from left to right according to the order of the first parent regarding the substring symbols to produce the second offspring, [***1 2*** 7 ***3 8*** 6 5 ***9*** 4].

4.6.3.3 Mutation

The mutation operation is a means of introducing new information into the population. For this dissertation, the mutations are based again on either pairwise interchange move or shift move (*see* Section 4.5).

4.6.3.4 Evaluation Policy

During each generation, chromosomes are evaluated using some measure of fitness. In most optimization applications, the fitness function is constructed based on the original objective function. The fitness value of each chromosome is a key measure to guide the direction of search in the GA. Due to the minimization problem, the fitness value must be in inverse proportion to the objective function value so that a fitter chromosome has a larger fitness value.

$$fitness\,(v_z) \quad = \quad \frac{1}{f\,(v_z)} \qquad , z = 1,2,...,\ population\_size \tag{4.15}$$

where *fitness*(v_z) is the fitness value, and *f*(v_z) is the objective function value of the *z*-th chromosome for the complete schedule generated from the corresponding job sequence for the first stage using Approach 1 (*see* Section 4.3).

According to this research objective, the objective is to minimize a positively weighted convex sum of makespan and number of tardy jobs. Thus, the fitness value of a chromosome, *fitness*(v_z) is given by:

$$fitness\,(v_z) = \frac{1}{\lambda C_{\max}(v_z) + (1-\lambda)\sum_{j=1}^{n} U_j(v_z) + 1} \qquad (4.16)$$

$$, z = 1, 2, ..., population\_size$$

where $C_{max}(v_z)$ is the makespan of the *z*-th chromosome (resp. of the resulting complete schedule) ,$U_j(v_z)$ is a Boolean variable for job *j* of the *z*-th chromosome which is equal to 1 if job *j* is tardy, and 0 otherwise, and λ denotes the weight (or relative importance) given to makespan and number of tardy jobs. The largest value of the fitness function is the lowest value of the positively weighted convex sum of makespan and number of tardy jobs. In the denominator value one is added in order to prevent a division by zero when the weight λ and the number of tardy jobs are zero.

4.6.3.5 Selection Policy

An elitist policy and enlarged sampling space technique are used. Both parents and the offspring have the same chance of competing for survival. Figure 4.7 illustrates the selection based on an enlarged sampling space. Then Holland’s *proportionate selection* or *roulette wheel selection* is employed to reproduce the next generation based on the current enlarged population. The idea is to determine a selection probability (also called *survival probability*) for each chromosome proportional to its fitness value. For chromosome v_z with fitness *fitness*(v_z), its selection probability *prob*(v_z) is calculated as follows:

$$prob(v_z) = \frac{fitness(v_z)}{\sum_{z=1}^{population\_size + offspring\_size} fitness(v_z)} \tag{4.17}$$

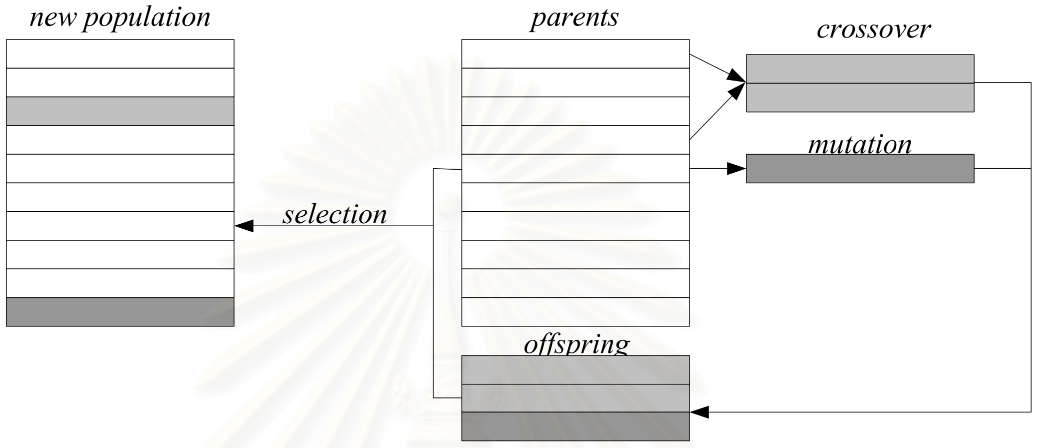


Figure 4.7 Illustration of the selection performed on an enlarged sampling space

4.6.3.6 Termination Condition

In the implementation of a GA approach, the search procedure is terminated when the best objective function value found so far is not updated for a predetermined number of generations. It can also be terminated when the number of generations exceeds the predetermined number of generations or when it reaches the other criteria.

4.7 Choice of an Initial Solution for the Iterative Algorithms

For the original iterative algorithms as stated in the previous section, their initial solution is generated by random. However, many researchers try to combine some local search with the iterative algorithms; for example, Reeves (1995), Wang and Zheng (2003), and Grabowski and Wodecki (2004) have used the NEH algorithm as the initial solution instead of a random initial solution for their iterative methods, namely a tabu search and a genetic algorithm. Janiak *et al.* (2007) have used the best solution among their constructive algorithms such ECT, EDD, and EDP as an initial solution for the simulated and tabu search algorithms. These imply that a better initial solution gives a better schedule solution as well. Hence, it is possible that the iterative algorithm can use a solution that is found by both constructive algorithms and improvement algorithms as a biased initial solution (or a part of the initial population for the genetic algorithm).

To improve the quality of the solution finally obtained, the influence of the choice of an appropriate initial solution for the SA and TS algorithms, and an initial population for the GA algorithm by using the heuristic constructive and improvement algorithms are also investigated. To this end, one or several constructive algorithm(s) SPT, LPT, ERD, EDD, MST, S/P, PAL, CDS, GUP, DAN, and NEH as well as the other selected polynomial improvement heuristics as initial solution(s), respectively (for the GA algorithm, the remaining initial solutions are still randomly generated) are employed. In addition, for the GA algorithm, all selected constructive algorithms in parallel as a part of the initial population are used.

4.8 Conclusion

In this chapter, the heuristic solution concepts for the flexible flow shop problem with unrelated parallel machines are presented. Three kinds of heuristics, namely *constructive*, *improvement*, and *iterative* algorithms, are developed. The constructive algorithms are adapted from the idea of Santos *et al.* (1996). Then, the improvement algorithms are proposed by using the neighborhood exchanges to improve the solution obtained from the constructive algorithms. The iterative algorithms based on the artificial algorithms are used to find the solution. The computational results of these heuristic algorithms will be shown in the next chapter.

Firstly, the constructive algorithms are developed on the starting job sequence for the first stage in Section 4.4. Such algorithms start with the generation of the representatives for each operation of each job and each stage. The representatives of the operating time are generated by using the combinations of the different relative machine speeds and setup times. After creating the representatives of operating time, some algorithms, namely dispatching rules and flow shop heuristic algorithms, are adapted to determine a starting job sequence for the first stage by using the nine combinations of the representative operating times. Next, use the greedy search approach to distribute jobs into machines for the first stage, and use both the FIFO and permutation rules to determine the new job sequence for other following stages, and follow by the greedy search again to distribute jobs into machine on that stage. With the nine combinations of the representatives of operating times, nine schedule outputs are generated, so the best schedule is selected from them.

Moreover, the improvement algorithms are proposed in Section 4.5. They start with an already built schedule from the constructive algorithm and try to improve the schedule by applying the ideas of the pairwise interchange and shift move approaches on the jobs that are tardy.

Moreover, the iterative algorithms, namely simulated annealing, tabu search, and genetic algorithms, are proposed on the starting job sequence in Section 4.6, and the hybrid iterative algorithms that use the constructive algorithms' solutions and/or improvement's solution as an initial solution (or a part of population) for the iterative algorithms are proposed in Section 4.7.

CHAPTER V

COMPUTATIONAL EXPERIMENTS

This chapter provides the computational experiments of the heuristic algorithms that are proposed in the previous chapter for the flexible flow shop problem with unrelated parallel machines. The computational experiments have been performed using a randomly generated set of test instances. From the heuristic solution concepts in the previous chapter, three main types of heuristics, which are constructive, improvement, and iterative algorithms, are proposed in this dissertation. All proposed heuristic algorithms are applied to determine a starting job sequence for the first stage of the flexible flow shop environment, and the greedy search approach is then used to distribute jobs into machines for the first production stage. Next, both the FIFO rule and the permutation rule are used to determine the new job sequence for other further stages, and the greedy search approach is again used to distribute jobs into machines for any stages. In this chapter, the results of the computational experiments will show the performance of the proposed heuristic algorithms.

This chapter is organized as follows: Firstly, the introduction is explained to give the definition and importance of the computational experiments. Secondly, the data generation of the test instances is given. Next, the performance compared to the best heuristics that are found in these tests on medium- and large-sized test problems of the heuristic algorithms that are proposed in the Chapter IV is presented. The performance on small-sized test problems that are compared to the optimal solution is shown in the next section. From the results of the computational experiments, the recommended heuristic solution approach is proposed. Finally, a conclusion will be drawn.

5.1 Introduction

Two fundamental goals in computer science are finding algorithms with provably good run times and with provably good or optimal solution quality. However, a heuristic is an algorithm that may abandon one or both of these goals; for example, it usually finds pretty good solutions, but there is no proof the solutions could not get arbitrarily bad; or it usually runs reasonably quickly, but there is no argument that this will always be the case (Pearl, 1984).

Consequently, the computational experiment is designed to evaluate the performance of the heuristics. In the computational testing of the algorithm, the experiment consists of solving a series of problem instances using a computer implementation. The most prevalent computational experiment concerns the relative effectiveness (in terms of stated performance measures such as computational effort or quality of solutions) of different heuristic methods in solving specific classes of problems (Barr *et al.*, 1995).

For the solution quality, there are many methods to evaluate the performance of the algorithms. Basically, the schedules generated are compared to the optimal solution by calculating the percentage deviation of the heuristic solution from the optimal solution as shown in the following equation:

$$\text{\% deviation from the optimal solution} = \frac{Heu_{sol} - Opt_{sol}}{Opt_{sol}} \times 100 \qquad (5.1)$$

where Heu_{sol} is the schedule solution obtained by a given algorithm, and Opt_{sol} is the optimal schedule solution obtaining by using an exact algorithm.

However, the determination of the optimal schedule solution may be practically impossible for the large-sized problems, or can be difficult for even medium-sized problem, so it is hard to evaluate the quality of the heuristic solution by comparing to the optimal schedule solution. For this reason, an effective tool for estimating the optimal solution, which is called the determination of a lower bound, is created to evaluate the quality solution instead of the optimal schedule solution. The quality of solution obtained is checked by calculating the percentage deviation of the heuristic solution from the lower bound (LB) as shown in the following equation:

$$\text{\% deviation from the lower bound } = \frac{Heu_{sol} - LB_{sol}}{LB_{sol}} \times 100 \qquad (5.2)$$

where Heu_{sol} is the schedule solution obtained by a given algorithm, and LB_{sol} is the lower bound of the solution.

Moreover, the benchmarks are available from the OR-Library (*i.e.* http://people.brunel.ac.uk/~mastjjb/jeb/info.html), which is a collection of test data sets for a variety of operations research (OR) problems. Consequently, the quality solution is compared to the best solution that is found in the OR-Library. However, for the new problem that the benchmark cannot find in the OR-Library, the quality solution can be compared to the best solution that is found in the tests by using the following equation:

$$\text{\% deviation from the best solution } = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \times 100 \qquad (5.3)$$

where Heu_{sol} is the schedule solution obtained by a given algorithm, and $Best_{sol}$ is the best solution found among the tests.

5.2 Data Generation of the Test Instances

The models are tested by using random generated data. For every experiment, random instances with unrelated parallel machines, the standard processing times, relative machine speeds, setup times, release dates, and due dates are generated as follows. The standard processing times are generated uniformly from the interval [10,100]. The relative speeds are distributed uniformly in the interval [0.7, 1.3]. The setup times, both sequence- and machine-dependent setup times, are generated uniformly from the interval [0, 50], whereas the release dates are generated uniformly from the interval between 0 and half of their total standard processing time mean. The due date of a job is set in a way that is similar to the approach presented by Rajendran and Ziegler (2003) and is as follows:

$$d_j = r_j + \sum_{t=1}^{k} ps_j^t + \text{total of mean setup time of a job on all stages} + (n-1) \times (\text{mean processing time of a job on one machine}) \times U(0,1) \quad (5.4)$$

The random generated data are tested on medium- and large-sized problems with 10 jobs × 5 stages, 30 jobs × 10 stages, and 50 jobs × 20 stages. For small-sized problems with a number of jobs ranging from three to seven were investigated. For all problem sizes, they were investigated with $\lambda \in \{0, 0.001, 0.005, 0.01\ 0.05, 0.1, 0.5, 1\}$ in the objective function. The idea behind the λ values is to balance both objectives. For a low value of λ, the tardy job problem will dominate the makespan problem, whereas for a large value of λ, the makespan problem will dominate the tardy job problem. Ten different instances for each problem size have been run.

All algorithms have been implemented in the C++ programming language on a PC with an Intel Pentium 4 2.00GHz CPU and 256 MB of RAM. The optimal solution obtained by means of the 0-1 mixed linear integer programming formulation given in Chapter III is found by a commercial mathematical programming software.

5.3 Performance of Algorithms on Medium- and Large-Sized Test Problems

The purpose of these experiments is to valuate the performance of each algorithm that are proposed in Chapter IV on the test problems whose sizes are medium or large (*see* Section 5.2) that cannot find the optimal solution in an acceptable time.

For the problem with $\lambda = 0$, the performance of each test for each algorithm is assessed by the absolute deviation of a particular algorithm from the best solution in such a test among the heuristic groups I and II as well as groups III and IV stated in Section 5.3.2 by using the following equation:

$$\text{absolute deviation from the best solution } = Heu_{sol} - Best_{sol} \qquad (5.5)$$

where Heu_{sol} is the schedule solution obtained by a given algorithm, and $Best_{sol}$ is the best solution found among the tests.

For the problem with $\lambda > 0$, the performance of each test for each algorithm is assessed by the percentage deviation of a particular algorithm from the best solution in such a test among the heuristic groups I and II as well as groups III and IV stated in Section 5.3.2 by using the following equation:

$$\text{percentage deviation from the best solution } = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \times 100 \qquad (5.6)$$

where Heu_{sol} is the schedule solution obtained by a given algorithm, and $Best_{sol}$ is the best solution found among the tests.

5.3.1 Performance of the Constructive Algorithms

The purpose of the first experiment is to test the computational efficiency of constructive algorithms – the simple dispatching rules (*see* Section 4.4.1) and flow shop makespan heuristics (*see* Section 4.4.2). The simple dispatching rules, namely SPT, LPT, ERD, EDD, MST, and S/P, are classified into the heuristic "Group I", whereas the flow shop makespan heuristics, namely PAL, CDS, GUP, DAN, and NEH, are classified into the heuristic "Group II".

Table 5.1 Average performance of the constructive algorithms of Group I and II

λ	Problem size	Group I						Group II				
		SPT	LPT	ERD	EDD	MST	S/P	PAL	CDS	GUP	DAN	NEH
0	10×5	3.000[a]	3.200	3.500	4.600	4.100	4.100	2.700	2.200	2.800	2.600	***0.700***
	30×10	*6.900*	7.900	7.700	7.900	8.400	7.900	7.700	6.100	7.300	7.800	*1.800*
	50×20	8.700	8.200	11.100	15.800	14.600	14.100	9.400	6.600	8.600	8.800	*1.000*
	Sum	18.600	19.300	22.300	28.300	27.100	26.100	19.800	14.900	18.700	19.200	***3.500***
0.001	10×5	90.920[b]	94.290	*87.880*	102.460	91.560	90.930	73.220	61.630	77.800	70.370	***10.470***
	30×10	89.090	104.510	94.730	90.250	100.510	91.170	101.410	77.990	94.510	98.410	*29.980*
	50×20	31.830	34.420	42.570	49.770	45.600	43.960	37.030	27.000	34.620	35.200	*10.320*
	Sum	211.840	233.220	225.180	242.480	237.670	226.060	211.660	166.620	206.930	203.980	*50.770*
0.005	10×5	45.290	*44.130*	44.710	58.240	52.180	52.520	38.870	31.010	41.540	36.150	***6.250***
	30×10	42.140	45.420	43.800	43.640	46.770	41.540	44.290	34.330	42.650	43.740	*11.500*
	50×20	18.812	*18.338*	23.140	28.685	26.281	25.233	20.046	15.791	18.867	18.902	*4.411*
	Sum	106.242	107.888	111.650	130.565	125.231	119.293	103.206	81.131	103.057	98.792	***22.161***
0.01	10×5	33.300	*30.430*	31.040	41.170	36.990	37.630	28.570	22.230	30.560	25.780	***4.710***
	30×10	30.633	30.954	30.780	31.752	33.282	*28.880*	29.870	23.506	29.744	29.561	*6.887*
	50×20	14.895	*14.199*	17.734	21.330	19.445	18.625	15.440	12.655	14.646	14.514	***3.160***
	Sum	78.828	*75.583*	79.554	94.252	89.717	85.135	73.880	58.391	74.950	69.855	***14.757***
0.05	10×5	22.154	*16.778*	17.176	21.889	20.413	19.662	18.069	12.421	19.019	15.633	***2.399***
	30×10	20.477	17.413	19.306	21.227	21.110	*16.986*	16.766	13.394	17.207	15.928	***2.386***
	50×20	10.406	*9.721*	11.872	12.748	11.476	10.838	10.355	8.400	9.898	9.457	***0.765***
	Sum	53.037	*43.912*	48.354	55.864	52.999	47.486	45.190	34.215	46.124	41.018	***5.550***
0.1	10×5	21.084	*15.177*	15.656	19.457	18.163	17.181	17.073	11.196	17.585	14.471	***2.097***
	30×10	18.691	15.309	17.482	19.453	19.007	*15.058*	14.637	11.722	15.071	13.784	***1.470***
	50×20	10.029	*9.384*	11.335	11.772	10.554	9.935	9.877	8.016	9.523	8.985	***0.479***
	Sum	49.804	*39.870*	44.473	50.682	47.724	42.174	41.587	30.934	42.179	37.240	***4.046***
0.5	10×5	21.203	*14.852*	15.456	18.446	17.310	16.114	17.373	11.176	17.221	14.448	***2.700***
	30×10	18.759	15.021	17.524	19.528	18.653	*14.916*	14.368	11.768	14.794	13.488	***0.869***
	50×20	9.985	*9.394*	11.181	11.244	10.068	9.446	9.754	7.933	9.489	8.866	***0.436***
	Sum	49.947	*39.267*	44.161	49.218	46.031	40.476	41.495	30.877	41.504	36.802	*4.005*
1.0	10×5	21.473	*15.061*	15.696	18.567	17.426	16.214	17.674	11.400	17.418	14.701	***2.885***
	30×10	18.793	15.018	17.551	19.567	18.630	*14.923*	14.367	11.785	14.780	13.477	***0.964***
	50×20	9.892	9.308	11.073	11.087	9.918	*9.296*	9.651	7.837	9.399	8.766	***0.428***
	Sum	50.158	*39.387*	44.320	49.221	45.974	40.433	41.692	31.022	41.597	36.944	***4.277***

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

The results of the first experiment are given in Table 5.1. The performance of algorithms is assessed by the average performance of the (absolute for $\lambda = 0$ resp. percentage for $\lambda > 0$) deviation of a particular algorithm from the best solution in these tests (among the heuristic groups I and II as well as groups III and IV stated in Section 5.3.2) for each three-problem size $n \times k$ (the best variant in each group is given in italic underlined whereas the overall best variant is given in bold face underlined).

The results of constructive algorithms show that among the simple dispatching rules (heuristic Group I), the SPT, LPT, and ERD rules are good dispatching rules. However, in general the SPT rule outperforms the other dispatching rules for $\lambda < 0.01$, and the LPT rule is better than the other rules otherwise. The results confirm earlier observations with the LPT algorithms, the advantage of which is to keep jobs with shortage processing times for later because these jobs are useful at the end for balancing the workload (Pinedo and Chao, 1999). It coincides with a previous study by Guinet *et al.* (1992) where they conclude that the LPT rule gives good results in a two-stage hybrid flow shop problem.

Among the adapted flow shop makespan heuristics in heuristic Group II, the NEH algorithm is clearly the best algorithm among all of the studied constructive heuristics (but, in fact, this algorithm takes the convex combination of both criteria into account when selecting partial sequences). This is in correspondence with Framinan, Gupta, and Leisten (2004) and Ruiz and Maroto (2005), who have found that, among the constructive methods, the NEH algorithm is regarded as the best one in practice. The CDS algorithm is certainly the algorithm on the second rank (but it is substantially worse than the NEH algorithm, even if the makespan portion in the objective function value is dominant, *i.e.* for large λ values). However, the main drawback of the NEH algorithm is that a total of $[n(n + 1)/2] - 1$ partial schedules need to be evaluated. The running time of the NEH algorithm, therefore, increases rapidly as the problem size increases.

5.3.2 Performance of the Improvement Algorithms

The purpose of the second experiment is to test the computational efficiency of the improvement algorithms that are proposed in Chapter IV (*see* Section 4.6). The fast improvement algorithms based on the four cases, namely, 2-random-position shift move, all shift move, 2-random-position pairwise interchange, and all pairwise interchange, that are proposed to improve the quality of the constructive algorithms in Chapter IV (*see* Section 4.5).

The average overall performance of the constructive algorithms (denoted by the letter "CA") and the fast improvement algorithms (denoted by the letter "2-SM", "A-SM", "2-PI", and "A-PI", respectively) are shown in Table 5.2.

The average overall performance of the CA group is calculated by finding the average performance of the average (absolute for $\lambda = 0$ resp. percentage for $\lambda > 0$) deviation of the constructive algorithms, namely SPT, LPT, ERD, EDD, MST, S/P, PAL, CDS, GUP, DAN, and NEH from Section 5.3.1.

For the average overall performance of the 2-SM group, which is applied the 2-random shift move on the constructive algorithms (*see* Section 4.5), the performance is calculated by finding the average performance of the average (absolute for $\lambda = 0$ resp. percentage for $\lambda > 0$) deviation of such algorithms from the best solution in these tests.

In the same method, the average overall performance of the A-SM, 2-PI, and A-PI group, each performance is also calculated by finding the average performance of the average (absolute for $\lambda = 0$ resp. percentage for $\lambda > 0$) deviation of each group of such algorithms from the best solution in these tests (the overall best variant is given in bold face).

Table 5.2 Average overall performance of the constructive and polynomial improvement heuristics

λ	Problem size	CA	2-SM	A-SM	2-PI	A-PI
0	10×5	3.045[a]	1.527	**1.173**	1.691	1.209
	30×10	7.036	3.927	2.982	4.264	**2.045**
	50×20	9.718	5.773	4.591	5.591	**2.236**
	Sum	19.799	11.227	8.746	11.546	**5.490**
0.001	10×5	77.410[b]	28.710	**19.530**	33.200	21.130
	30×10	88.410	37.020	24.030	37.820	**19.040**
	50×20	35.670	12.580	12.320	10.190	**5.730**
	Sum	201.490	78.310	55.880	81.210	**45.900**
0.005	10×5	40.990	14.870	**9.530**	18.150	11.500
	30×10	39.980	16.050	10.610	17.970	**8.590**
	50×20	19.864	8.612	8.170	8.658	**4.592**
	Sum	100.834	39.532	28.310	44.778	**24.682**
0.01	10×5	29.310	10.780	**6.870**	13.650	8.370
	30×10	27.804	12.240	7.727	13.843	**6.593**
	50×20	15.149	8.241	7.536	8.668	**5.415**
	Sum	72.263	31.261	22.133	36.161	**20.378**
0.05	10×5	16.874	6.029	**4.547**	8.306	5.231
	30×10	16.564	8.182	6.110	9.846	**4.891**
	50×20	9.631	5.643	5.312	6.518	**5.036**
	Sum	43.069	19.854	15.969	24.670	**15.158**
0.1	10×5	15.376	5.454	**3.997**	8.653	4.764
	30×10	14.699	6.685	4.707	8.702	**3.683**
	50×20	9.081	5.210	5.082	5.778	**4.730**
	Sum	39.156	17.349	13.786	23.133	**13.177**
0.5	10×5	15.118	5.486	**3.943**	7.884	4.457
	30×10	14.517	7.087	5.413	8.326	**4.304**
	50×20	8.891	5.147	4.946	6.554	**4.610**
	Sum	38.526	17.720	14.302	22.764	**13.371**
1.0	10×5	15.319	5.279	**4.144**	7.671	4.581
	30×10	14.532	6.901	5.294	9.186	**4.329**
	50×20	8.787	5.421	4.875	6.098	**4.567**
	Sum	38.638	17.601	14.313	22.955	**13.477**

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

From these results, it is obvious that the fast polynomial improvement heuristics 2-SM and 2-PT can improve the quality of the constructive algorithms by about 40 – 60 percent, whereas the A-SM and A-PI heuristics can improve the quality of the constructive algorithms even by 60 – 75 percent. In addition, it is noticed that for the problem size 10 jobs × 5 stages the all-shift-move (A-SM) heuristic is slightly better than the others, whereas the all-pairwise-interchange-based (A-PI) improvement heuristic is the best algorithm otherwise. However, in general the A-PI algorithm should be selected as the improvement algorithm. Consequently, in this dissertation only the A-PI-based improvement heuristic is used as an improvement algorithm. However, when comparing between the 2-SM and 2-PI algorithms whose CPU time is smaller than the CPU time of both the A-SM and A-PI algorithms, it is found that the 2-SM algorithm certainly behaves better than the 2-PI algorithm.

Next, the results for the A-PI improvement algorithms, heuristic "Group III" and heuristic "Group IV", are presented. They are generated from the first two groups of heuristics, where the solutions are improved by the selected polynomial improvement algorithm based on A-PI improvement heuristics, and they are denoted by the first letter "I" in front of the letters describing the heuristics of the first two groups.

The results for the fast polynomial improvement algorithms are given in Table 5.3 (the best variant in each group is given in italic underlined while the overall best variant is given in bold face underlined). From these results, it is obvious that the algorithms in the fourth heuristic group (namely, IPAL, ICDS, IGUP, IDAN, and INEH) improved the pure makespan heuristics from the second heuristic group (*i.e.* PAL, CDS, GUP, DAN, and NEH), and they are better than the dispatching rules in the first heuristic group (*i.e.* SPT, LPT, ERD, EDD, MST, and S/P) as well as the third heuristic group improved from them.

Table 5.3 Average performance of the fast polynomial improvement algorithms of Group III and IV

λ	Problem size	Group III						Group IV				
		ISPT	**ILPT**	**IERD**	**IEDD**	**IMST**	**IS/P**	**IPAL**	**ICDS**	**IGUP**	**IDAN**	**INEH**
0	10×5	1.400[a]	1.300	1.200	*1.000*	1.300	1.300	1.400	1.300	1.100	1.300	***0.700***
	30×10	2.100	2.500	2.400	1.500	***1.200***	1.800	2.500	2.300	2.100	2.300	*1.800*
	50×20	*1.300*	2.300	3.000	3.000	4.900	3.300	1.700	***0.700***	2.000	1.400	1.000
	Sum	*4.800*	6.100	6.600	5.500	7.400	6.400	5.600	4.300	5.200	5.000	***3.500***
0.001	10×5	16.220[b]	24.020	*12.550*	13.710	18.940	18.870	35.200	23.980	35.630	22.840	***10.470***
	30×10	20.310	18.910	18.970	***13.200***	15.710	13.910	*17.750*	18.110	21.450	21.140	29.980
	50×20	6.730	5.150	7.690	5.010	4.040	6.960	8.750	2.640	3.260	***2.460***	10.320
	Sum	43.260	48.080	39.210	***31.920***	38.690	39.740	61.700	*44.730*	60.340	46.440	50.770
0.005	10×5	10.000	13.410	*8.170*	8.650	11.990	11.740	13.980	14.060	15.260	12.970	***6.250***
	30×10	9.690	8.150	8.610	7.530	7.520	*6.610*	9.650	***6.050***	9.020	10.200	11.500
	50×20	3.924	5.228	5.514	6.051	*3.554*	5.631	5.453	***3.306***	3.734	3.703	4.411
	Sum	23.614	26.788	22.294	*22.231*	23.064	23.981	29.083	23.416	28.014	26.873	***22.161***
0.01	10×5	8.890	9.450	7.030	*6.640*	7.690	9.540	8.460	9.810	10.130	9.770	***4.710***
	30×10	6.373	9.709	5.676	6.753	4.762	***4.076***	7.585	*4.935*	8.988	6.782	6.887
	50×20	*4.699*	6.427	5.749	6.264	6.890	6.583	6.183	4.934	3.431	5.251	***3.160***
	Sum	19.962	25.586	*18.455*	19.657	19.342	20.199	22.228	19.679	22.549	21.803	***14.757***
0.05	10×5	5.476	5.281	*4.900*	6.629	5.643	5.620	6.322	4.229	5.675	5.365	***2.399***
	30×10	4.820	6.313	*2.768*	6.397	5.431	4.893	5.865	4.227	5.282	5.419	***2.386***
	50×20	*4.778*	5.247	5.438	7.221	6.010	6.711	5.486	3.139	5.538	5.064	***0.765***
	Sum	15.074	16.841	*13.106*	20.247	17.084	17.224	17.673	11.595	16.495	15.848	***5.550***
0.1	10×5	*4.546*	5.404	4.787	6.318	5.721	4.877	5.749	3.752	4.154	4.996	***2.097***
	30×10	3.255	4.743	*1.718*	5.523	4.957	5.033	4.193	2.241	3.848	3.537	***1.470***
	50×20	5.169	*4.241*	5.024	6.681	5.831	5.788	5.336	3.126	5.394	4.955	***0.479***
	Sum	12.970	14.388	*11.529*	18.522	16.509	15.698	15.278	9.119	13.396	13.488	***4.046***
0.5	10×5	4.969	4.932	5.195	5.707	6.287	*4.629*	4.327	2.790	4.147	3.346	***2.700***
	30×10	3.929	5.283	*2.404*	6.727	5.135	6.897	4.936	2.812	4.611	3.745	***0.869***
	50×20	5.453	*4.244*	5.090	6.215	5.900	4.725	5.163	3.450	5.125	4.906	***0.436***
	Sum	14.351	14.459	*12.689*	18.649	17.322	16.251	14.426	9.052	13.883	11.997	*4.005*
1.0	10×5	5.018	5.073	5.268	5.741	5.935	*4.840*	4.527	3.195	4.378	3.531	***2.885***
	30×10	4.155	4.838	*2.421*	6.843	5.932	6.940	4.910	2.405	4.666	3.543	***0.964***
	50×20	5.346	*4.147*	5.107	6.079	5.731	4.523	5.044	3.808	5.046	4.976	***0.428***
	Sum	14.519	14.058	*12.796*	18.663	17.598	16.303	14.481	9.408	14.090	12.050	***4.277***

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

When applying a fast pairwise interchange (A-PI) algorithm (denoted by the letter “I” first) to a dispatching rule and/or an adapted makespan heuristic, the results show that the quality of the solution can be improved by about 60 – 80 percent except for the NEH rule (for the problems with small positive λ value the percentage improvements are often larger than for the remaining problems, *e.g.* for $\lambda = 0.001$ the IEDD algorithm can improve the results of the EDD rule by about 87 percent on average). It can be noted that the NEH rule is not improved by using the improvement heuristics of algorithm INEH because both algorithms use a very similar strategy (it confirms the excellent solution quality by algorithm NEH). However, the improvement of the heuristics from the adapted pure makespan heuristics in the heuristic Group IV is better than the improvement of the heuristics derived from the dispatching rules in the heuristic Group III.

5.3.3 Performance of the Iterative Algorithms

Thirdly, the iterative algorithms, *i.e.* the SA, TS, and GA algorithms with a random initial solution (or population), have been studied (*see* Section 4.6). Before testing the performance of the iterative algorithms for the problem under consideration, it is necessary to find the favorable parameters of each iterative algorithm in order to reduce the effect of the parameters of each iterative algorithm on the solution quality.

Consequently, the purpose of this study is to determine the favorable parameters of the iterative algorithms for the problem under consideration. From the preliminary tests, the CPU time is limited to one second for the problems with ten jobs, ten seconds for the problems with 30 jobs, and 30 seconds for the problems with 50 jobs. Again, the algorithms are tested on the test instances with $\lambda \in \{0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1\}$. Based on the preliminary tests, the tested parameters are shown in Table 5.4.

Table 5.4 Tested parameters for the SA, TS, and GA algorithms

Algorithms	**Parameters**	**Levels**
SA	Initial temperature	2, 4, 6, 8, 10 through 100, in steps of 10
	Neighborhood structures	PI (= Pairwise Interchange), SM (= Shift Move)
	Cooling schedules	1 – 4 (geometric reduction with $\alpha \in \{0.8, 0.85, 0.9, 0.95\}$)
		5 – 13 (LM reduction with β: 0.01 through 0.09, in steps of 0.01)
		14 – 23 (LM reduction with β: 0.1 through 1.0, in steps of 0.1)
TS	Number of neighbors	10 through 50, in steps of 10
	Neighborhood structures	PI, SM
	Sizes of tabu list	5, 10, 15, and 20
GA	Population sizes	10, 30, 50, 70
	Crossover types	PMX, OPX
	Mutation types	PI, SM
	Crossover rates	0.1 through 0.9, in steps of 0.1
	Mutation rates	0.1 through 0.9, in steps of 0.1

Given the above three different problem sizes, the SA parameter values were tested. Table 5.5 through Table 5.7 present the effect of the initial temperatures, neighborhood structures and cooling schedules by using the average (absolute resp. relative) deviation from the best value as the performance measure.

From the full factorial experiment, the results are analyzed by means of a multi-factor *analysis of variance* (*ANOVA*) technique using a 5% significance level. The results give the average (absolute resp. percentage) deviation of a particular iterative algorithm from the best solution obtained by the iterative algorithms. For the SA algorithm, it is found that for the neighborhood structure and the cooling schedule, there are statistically significant differences, whereas there are slightly statistically significant differences in the initial temperature.

Table 5.5 The effect of various initial temperatures on the performance of the SA algorithm

	Problem	**Initial Temperature (T_0)**													
λ	**size**	**2**	**4**	**6**	**8**	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**
0	10×5	**0.017**[a]	0.019	0.024	0.019	0.024	0.024	0.030	0.022	0.028	0.027	0.028	0.029	0.023	0.032
	30×10	**0.483**	0.485	0.487	0.515	0.520	0.509	0.546	0.522	0.520	0.498	0.526	0.502	0.515	0.504
	50×20	**0.248**	0.252	0.278	0.270	0.278	0.296	0.291	0.309	0.307	0.298	0.311	0.289	0.257	0.294
	Sum	**0.748**	0.756	0.789	0.804	0.822	0.828	0.867	0.852	0.854	0.822	0.865	0.820	0.795	0.830
0.001	10×5	**0.772**[b]	0.963	0.987	1.003	1.168	1.181	0.894	1.243	1.154	1.058	1.193	1.315	1.076	1.161
	30×10	**9.480**	9.540	9.570	9.740	9.680	9.510	9.640	10.100	9.830	9.970	10.020	10.130	10.010	9.940
	50×20	**2.438**	2.582	2.617	2.583	2.655	2.670	2.769	2.711	2.756	2.745	2.723	2.810	2.825	2.794
	Sum	**12.690**	13.085	13.174	13.326	13.503	13.361	13.303	14.054	13.740	13.773	13.936	14.255	13.911	13.895
0.005	10×5	**0.649**	0.783	0.796	0.783	0.784	0.857	0.893	0.909	0.936	0.924	0.876	0.972	0.923	0.966
	30×10	**6.766**	7.164	6.989	7.203	7.079	7.038	7.132	7.135	7.068	7.294	7.326	7.280	7.309	7.437
	50×20	**2.863**	2.968	3.012	3.085	3.085	3.105	3.164	3.130	3.118	3.177	3.162	3.196	3.243	3.245
	Sum	**10.278**	10.915	10.797	11.071	10.948	11.000	11.189	11.174	11.122	11.395	11.364	11.448	11.475	11.648
0.01	10×5	**0.531**	0.657	0.654	0.710	0.729	0.736	0.778	0.753	0.749	0.735	0.741	0.831	0.783	0.814
	30×10	**6.111**	6.193	6.392	6.416	6.477	6.506	6.432	6.588	6.491	6.434	6.285	6.481	6.616	6.559
	50×20	**2.676**	2.793	2.808	2.819	2.851	2.918	2.798	2.921	2.900	2.887	2.911	2.928	3.001	2.964
	Sum	**9.318**	9.643	9.854	9.945	10.057	10.160	10.008	10.262	10.140	10.056	9.937	10.240	10.400	10.337
0.05	10×5	**0.178**	0.202	0.204	0.247	0.244	0.250	0.272	0.270	0.292	0.283	0.274	0.269	0.297	0.278
	30×10	4.644	**4.525**	4.761	4.705	4.883	4.560	4.855	4.916	4.879	4.774	4.776	4.910	4.901	4.805
	50×20	**1.931**	1.942	1.993	1.959	2.004	1.999	2.022	2.002	2.031	2.092	2.057	2.070	2.088	2.094
	Sum	6.753	**6.668**	6.958	6.911	7.131	6.809	7.149	7.188	7.201	7.149	7.107	7.250	7.286	7.176
0.1	10×5	**0.129**	0.162	0.155	0.176	0.167	0.164	0.180	0.189	0.188	0.182	0.205	0.196	0.208	0.186
	30×10	4.129	3.967	4.184	4.154	4.001	4.112	4.169	4.220	4.359	4.358	**3.929**	4.193	4.168	4.140
	50×20	1.648	**1.636**	1.654	1.699	1.721	1.709	1.743	1.752	1.726	1.820	1.743	1.769	1.785	1.796
	Sum	5.906	**5.764**	5.993	6.030	5.890	5.985	6.092	6.161	6.273	6.360	5.876	6.158	6.161	6.122
0.5	10×5	0.309	0.287	0.340	0.322	0.321	**0.275**	0.310	0.314	0.348	0.336	0.340	0.330	0.329	0.307
	30×10	4.645	4.425	**4.089**	4.322	4.598	4.422	4.366	4.179	4.484	4.557	4.481	4.690	4.690	4.514
	50×20	2.136	1.728	1.707	1.562	**1.551**	1.738	1.920	1.957	1.681	1.877	1.884	1.686	1.863	1.844
	Sum	7.090	6.440	**6.136**	6.206	6.470	6.435	6.596	6.450	6.513	6.770	6.705	6.706	6.882	6.665
1.0	10×5	0.511	0.479	0.451	0.425	0.417	**0.412**	0.456	0.430	0.479	0.477	0.462	0.458	0.469	0.467
	30×10	5.162	4.936	4.614	4.123	**4.009**	4.286	4.617	4.366	4.520	4.428	4.762	4.512	4.394	4.373
	50×20	3.535	2.291	**1.567**	1.924	1.681	1.677	2.019	1.761	2.350	1.904	2.412	2.505	2.283	2.126
	Sum	9.208	7.706	6.632	6.472	**6.107**	6.375	7.092	6.557	7.349	6.809	7.636	7.475	7.146	6.966

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

Table 5.6 The effect of various neighborhood structures on the performance of the SA algorithm

	Problem	**Neighborhood Structure**	
λ	**size**	**PI**	**SM**
0	10×5	0.029[a]	**0.020**
	30×10	0.525	**0.494**
	50×20	**0.248**	0.320
	Sum	**0.802**	0.834
0.001	10×5	1.279[b]	**0.888**
	30×10	10.040	**9.560**
	50×20	**2.640**	2.742
	Sum	13.959	**13.190**
0.005	10×5	0.973	**0.748**
	30×10	7.561	**6.757**
	50×20	3.196	**3.026**
	Sum	11.730	**10.531**
0.01	10×5	0.830	**0.628**
	30×10	6.900	**5.954**
	50×20	3.048	**2.691**
	Sum	10.778	**9.273**
0.05	10×5	0.377	**0.131**
	30×10	5.306	**4.250**
	50×20	2.299	**1.741**
	Sum	7.982	**6.123**
0.1	10×5	0.312	**0.043**
	30×10	4.591	**3.707**
	50×20	2.009	**1.449**
	Sum	6.912	**5.198**
0.5	10×5	0.549	**0.089**
	30×10	4.810	**4.113**
	50×20	1.974	**1.617**
	Sum	7.333	**5.819**
1.0	10×5	0.721	**0.193**
	30×10	4.750	**4.264**
	50×20	**2.143**	2.148
	Sum	7.614	**6.605**

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

Table 5.7 The effect of various cooling schedules on the performance of the SA algorithm

	Problem	Cooling Schedules											
λ	size	CS1	CS2	CS3	CS4	CS5	CS6	CS7	CS8	CS9	CS10	CS11	CS12
0	10×5	0.004[a]	0.001	0.011	0.043	0.221	0.087	0.060	0.035	0.029	0.019	0.021	0.005
	30×10	**0.207**	0.221	0.282	0.536	1.539	0.961	0.800	0.693	0.632	0.639	0.575	0.532
	50×20	0.043	0.046	0.079	0.400	2.107	1.011	0.686	0.439	0.311	0.229	0.207	0.196
	Sum	**0.254**	0.269	0.371	0.979	3.868	2.058	1.545	1.167	0.971	0.887	0.804	0.733
0.001	10×5	1.229[b]	1.244	1.229	2.314	6.868	3.036	1.751	1.284	1.035	0.712	0.631	0.577
	30×10	8.453	8.005	8.104	10.517	24.762	15.777	13.808	12.334	11.504	10.946	10.296	9.799
	50×20	1.067	1.131	1.272	2.850	9.646	5.948	4.541	3.811	3.354	3.186	3.095	2.827
	Sum	10.749	10.380	10.605	15.681	41.276	24.761	20.100	17.429	15.893	14.844	14.022	13.203
0.005	10×5	8.453	8.005	8.104	10.517	24.762	15.777	13.808	12.334	11.504	10.946	10.296	9.799
	30×10	6.061	5.774	6.188	7.414	16.505	12.073	10.335	9.466	8.694	8.587	8.052	7.753
	50×20	**0.981**	1.065	1.281	3.076	9.124	6.819	5.714	4.852	4.388	4.062	3.773	3.575
	Sum	15.495	14.844	15.573	21.007	50.391	34.669	29.857	26.652	24.586	23.595	22.121	21.127
0.01	10×5	8.453	8.005	8.104	10.517	24.762	15.777	13.808	12.334	11.504	10.946	10.296	9.799
	30×10	5.721	5.659	5.815	6.690	13.881	10.679	9.223	8.345	7.677	7.219	6.806	6.462
	50×20	**1.105**	1.146	1.293	2.841	8.468	6.220	4.839	4.136	3.713	3.409	3.244	3.057
	Sum	15.279	14.810	15.212	20.048	47.111	32.676	27.870	24.815	22.894	21.574	20.346	19.318
0.05	10×5	8.453	8.005	8.104	10.517	24.762	15.777	13.808	12.334	11.504	10.946	10.296	9.799
	30×10	5.599	5.655	5.013	5.303	7.394	5.896	5.416	5.307	4.975	4.772	4.901	4.699
	50×20	1.266	1.366	1.465	2.391	4.186	3.152	2.811	2.661	2.531	2.479	2.439	2.398
	Sum	15.318	15.026	14.582	18.211	36.342	24.825	22.035	20.302	19.010	18.197	17.636	16.896
0.1	10×5	8.453	8.005	8.104	10.517	24.762	15.777	13.808	12.334	11.504	10.946	10.296	9.799
	30×10	4.896	4.686	4.412	4.719	5.327	4.559	4.290	4.278	4.010	3.931	3.882	3.708
	50×20	1.190	1.197	1.323	2.055	2.952	2.473	2.334	2.279	2.218	2.165	2.150	2.103
	Sum	14.539	13.888	13.839	17.291	33.041	22.809	20.432	18.891	17.732	17.042	16.328	15.610
0.5	10×5	8.453	8.005	8.104	10.517	24.762	15.777	13.808	12.334	11.504	10.946	10.296	9.799
	30×10	4.718	4.384	4.132	4.186	4.035	3.723	3.621	**3.522**	3.673	3.861	3.711	4.021
	50×20	1.268	1.117	1.230	1.568	2.121	1.959	1.736	1.509	1.366	1.250	1.173	1.174
	Sum	14.439	13.506	**13.466**	16.271	30.918	21.459	19.165	17.365	16.543	16.057	15.180	14.994
1.0	10×5	0.344	0.272	0.206	0.144	**0.103**	0.112	0.117	0.152	0.193	0.203	0.208	0.235
	30×10	4.384	4.184	3.890	3.543	3.244	**3.141**	3.162	3.377	3.580	3.468	3.843	3.994
	50×20	1.413	1.331	1.386	1.627	2.077	1.640	1.432	1.372	**1.286**	1.293	1.425	1.419
	Sum	6.141	5.787	5.482	5.314	5.424	4.893	**4.711**	4.901	5.059	4.964	5.476	5.648

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

Table 5.7 The effect of various cooling schedules on the performance of the SA algorithm (cont.)

	Problem	Cooling Schedules										
λ	**size**	**CS13**	**CS14**	**CS15**	**CS16**	**CS17**	**CS18**	**CS19**	**CS20**	**CS21**	**CS22**	**CS23**
0	10×5	0.012[a]	0.010	0.002	0.001	**0.000**	0.004	0.001	0.001	**0.000**	0.002	**0.000**
	30×10	0.568	0.450	0.446	0.411	0.375	0.361	0.336	0.379	0.311	0.239	0.221
	50×20	0.186	0.150	0.107	0.061	0.043	**0.025**	0.057	0.036	0.039	0.036	0.039
	Sum	0.766	0.610	0.556	0.473	0.418	0.389	0.394	0.416	0.350	0.277	0.261
0.001	10×5	0.584[b]	0.431	0.334	0.191	0.323	0.208	0.265	0.105	**0.064**	0.351	0.152
	30×10	10.012	9.363	8.688	7.832	7.388	7.318	6.535	6.224	6.195	6.033	**5.475**
	50×20	2.797	2.754	2.287	1.973	1.743	1.589	1.326	1.299	1.236	1.118	**1.051**
	Sum	13.393	12.548	11.309	9.996	9.454	9.115	8.126	7.628	7.495	7.502	**6.678**
0.005	10×5	10.012	9.363	8.688	7.832	7.388	7.318	6.535	6.224	6.195	6.033	5.475
	30×10	7.627	7.315	5.749	5.399	5.253	4.653	4.530	4.369	4.384	4.239	**4.227**
	50×20	3.334	3.214	2.418	2.123	1.929	1.869	1.704	1.668	1.637	1.495	1.448
	Sum	20.973	19.892	16.855	15.354	14.570	13.840	12.769	12.261	12.216	11.767	**11.150**
0.01	10×5	10.012	9.363	8.688	7.832	7.388	7.318	6.535	6.224	6.195	6.033	5.475
	30×10	6.451	6.506	5.364	5.058	4.688	4.555	4.288	4.211	**3.946**	4.460	4.120
	50×20	2.905	2.789	2.327	2.157	2.077	1.946	1.816	1.775	1.664	1.584	1.494
	Sum	19.368	18.658	16.379	15.047	14.153	13.819	12.639	12.210	11.805	12.077	**11.089**
0.05	10×5	10.012	9.363	8.688	7.832	7.388	7.318	6.535	6.224	6.195	6.033	5.475
	30×10	4.437	4.362	4.094	3.964	3.977	4.067	3.941	3.853	**3.846**	4.301	4.127
	50×20	2.375	2.357	2.125	1.784	1.583	1.349	1.336	1.164	1.101	**1.066**	1.081
	Sum	16.824	16.082	14.907	13.580	12.948	12.734	11.812	11.241	11.142	11.400	**10.683**
0.1	10×5	10.012	9.363	8.688	7.832	7.388	7.318	6.535	6.224	6.195	6.033	5.475
	30×10	3.623	3.693	3.597	3.686	**3.517**	3.846	3.984	4.061	4.003	4.230	4.488
	50×20	2.079	1.996	1.552	1.281	1.166	1.091	**1.051**	1.097	1.232	1.376	1.403
	Sum	15.714	15.052	13.837	12.799	12.071	12.255	11.570	11.382	11.430	11.639	**11.366**
0.5	10×5	10.012	9.363	8.688	7.832	7.388	7.318	6.535	6.224	6.195	6.033	5.475
	30×10	4.013	4.114	4.291	4.520	4.747	5.277	5.340	5.490	5.609	5.743	5.886
	50×20	**1.043**	1.046	1.433	1.756	2.067	2.347	2.587	2.743	2.843	2.964	2.990
	Sum	15.068	14.523	14.412	14.108	14.202	14.942	14.462	14.457	14.647	14.740	14.351
1.0	10×5	0.249	0.261	0.462	0.542	0.706	0.778	0.885	1.013	1.052	1.129	1.136
	30×10	3.866	4.012	4.550	5.066	5.383	5.734	5.895	6.267	6.267	6.345	6.471
	50×20	1.458	1.617	2.490	2.728	2.978	3.241	3.260	3.359	3.461	3.517	3.536
	Sum	5.573	5.890	7.502	8.336	9.067	9.753	10.040	10.639	10.780	10.991	11.143

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

For the initial temperature, it is observed that a lower initial temperature is effective for the problem under consideration. However, not all the problem cases should use the same small initial temperature. For the problems with λ < 0.5, the use of an initial temperature of two and of ten for the other problems can be recommended.

For the neighborhood structure, it is clear that SMs are better than PI moves for $\lambda \geq 0.001$, especially for the problems whose makespan portion λC_{max} is dominant in comparison with the tardiness portion $(1-\lambda)\eta_T$, whereas the PI moves are slightly better than or nearly as good as the SMs for the other values. Consequently, the neighborhood structures should be based on PIs for $\lambda = 0$ and on shifts of jobs otherwise, or it can also be recommended to use SMs for the whole range of λ.

For the cooling schedule, the results show that the performance of a geometric reduction is as good as the performance of the Lundy and Mees reduction only for problems with $\lambda = 0$, whereas the Lundy and Mees reduction becomes better when the value λ increases. Nevertheless, the parameter of the Lundy and Mees reduction depends on the value λ. It is noted that for the problems with $\lambda < 0.5$, the cooling rate of the Lundy and Mees reduction is suitable at a range from 0.5 through 1.0 (1.0 is recommended), whereas for the other problems it is suitable at a range from 0.05 through 0.2 (0.1 is recommended).

Next, the TS algorithm with a random initial solution is studied. Given the above three different problem sizes, the effect of the number of neighbors, neighborhood structure, and size of tabu list by using the average (absolute resp. relative) deviation from the best value as the performance measure is shown in Table 5.8 through Table 5.10.

Table 5.8 The effect of the various numbers of neighbors on the performance of the TS algorithm

λ	**Problem size**	**The number of neighbors**				
		10	**20**	**30**	**40**	**50**
0	10×5	0.029[a]	**0.017**	0.033	0.050	0.083
	30×10	0.400	**0.242**	0.313	0.392	0.463
	50×20	**0.050**	0.146	0.346	0.533	0.625
	Sum	0.479	**0.404**	0.692	0.975	1.171
0.001	10×5	0.954[b]	0.989	**0.943**	1.477	3.281
	30×10	7.912	5.236	**4.940**	5.640	6.250
	50×20	0.981	**0.945**	2.040	3.409	4.250
	Sum	9.847	**7.170**	7.923	10.526	13.781
0.005	10×5	1.136	0.648	**0.618**	0.799	1.282
	30×10	6.057	3.942	**3.611**	4.134	4.195
	50×20	1.875	**1.663**	2.623	3.493	4.099
	Sum	9.068	**6.253**	6.852	8.426	9.576
0.01	10×5	0.781	**0.419**	0.474	0.730	1.099
	30×10	5.264	3.653	**3.549**	3.931	4.390
	50×20	2.171	**1.807**	2.783	3.744	4.161
	Sum	8.216	**5.879**	6.806	8.405	9.650
0.05	10×5	0.535	0.176	**0.166**	0.191	0.332
	30×10	4.585	3.727	**3.632**	3.777	4.119
	50×20	2.410	**1.734**	2.793	3.338	3.905
	Sum	7.530	**5.637**	6.591	7.306	8.356
0.1	10×5	0.381	**0.119**	0.158	0.154	0.344
	30×10	4.067	3.542	**3.458**	3.773	3.714
	50×20	2.174	**1.491**	2.313	2.925	3.555
	Sum	6.622	**5.152**	5.929	6.851	7.613
0.5	10×5	0.331	0.164	**0.108**	0.228	0.282
	30×10	3.705	**2.962**	3.168	3.182	3.569
	50×20	2.008	**1.304**	2.098	2.860	3.510
	Sum	6.044	**4.43**	5.374	6.269	7.36
1.0	10×5	0.358	**0.127**	0.152	0.218	0.327
	30×10	3.523	**2.805**	2.877	3.169	3.299
	50×20	2.129	**1.415**	2.321	2.861	3.551
	Sum	6.011	**4.347**	5.351	6.249	7.177

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

Table 5.9 The effect of the various neighborhood structures on the performance of the TS algorithm

λ	Problem size	Neighborhood Structures	
		PI	SM
0	10×5	**0.033**[a]	0.052
	30×10	**0.337**	0.387
	50×20	**0.163**	0.517
	Sum	**0.533**	0.955
0.001	10×5	**0.911**[b]	2.146
	30×10	**5.923**	6.068
	50×20	**2.050**	2.600
	Sum	**8.884**	10.814
0.005	10×5	**0.826**	0.967
	30×10	4.650	**4.125**
	50×20	**2.635**	2.867
	Sum	8.111	**7.959**
0.01	10×5	**0.667**	0.735
	30×10	4.413	**3.903**
	50×20	**2.759**	3.108
	Sum	7.839	**7.746**
0.05	10×5	0.379	**0.181**
	30×10	4.298	**3.638**
	50×20	2.839	**2.834**
	Sum	7.516	**6.653**
0.1	10×5	0.324	**0.138**
	30×10	4.004	**3.418**
	50×20	**2.407**	2.576
	Sum	6.735	**6.132**
0.5	10×5	0.283	**0.162**
	30×10	3.561	**3.073**
	50×20	**2.310**	2.402
	Sum	6.154	**5.637**
1.0	10×5	0.290	**0.183**
	30×10	3.341	**2.928**
	50×20	**2.345**	2.566
	Sum	5.976	**5.677**

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

Table 5.10 The effect of the various sizes of tabu list on the performance of the TS algorithm

	Problem	**Tabu list sizes**			
λ	**size**	**5**	**10**	**15**	**20**
0	10×5	0.057[a]	**0.030**	0.040	0.043
	30×10	0.380	0.367	**0.347**	0.353
	50×20	0.380	0.347	**0.287**	0.347
	Sum	0.817	0.743	**0.673**	0.743
0.001	10×5	2.254[b]	1.152	**0.924**	1.786
	30×10	6.046	5.912	6.412	**5.612**
	50×20	2.511	**2.221**	2.339	2.228
	Sum	10.811	**9.285**	9.675	9.626
0.005	10×5	1.036	**0.707**	0.894	0.949
	30×10	**4.325**	4.354	4.341	4.532
	50×20	2.837	2.746	2.711	**2.710**
	Sum	8.198	**7.807**	7.946	8.191
0.01	10×5	0.855	**0.534**	0.552	0.863
	30×10	**4.097**	4.165	4.238	4.131
	50×20	2.933	3.134	**2.687**	2.979
	Sum	7.885	7.833	**7.477**	7.973
0.05	10×5	0.261	**0.239**	0.257	0.364
	30×10	3.939	4.019	3.989	**3.926**
	50×20	**2.667**	2.905	2.903	2.869
	Sum	**6.867**	7.163	7.149	7.159
0.1	10×5	0.278	**0.150**	0.230	0.267
	30×10	3.684	3.769	**3.682**	3.708
	50×20	2.513	2.444	**2.427**	2.582
	Sum	6.475	6.363	**6.339**	6.557
0.5	10×5	0.219	**0.190**	0.217	0.264
	30×10	3.379	**3.219**	3.375	3.295
	50×20	2.369	2.340	2.431	**2.283**
	Sum	5.967	**5.749**	6.023	5.842
1.0	10×5	0.290	**0.167**	0.207	0.283
	30×10	3.171	3.105	**3.085**	3.177
	50×20	2.434	2.451	2.521	**2.416**
	Sum	5.895	**5.723**	5.813	5.876

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

For the number of neighbors, 20 and 30 non-tabu neighbors per iteration are good choices, but 20 nontabu neighbors are still slightly better.

For the neighborhood structure, it is clear that PI moves are better than SMs for $\lambda < 0.005$, whereas for $\lambda = 0.005$ and the problem sizes 10 jobs × 5 stages and 50 jobs × 20 stages, there are no statistically significant differences in both neighborhood structures, but they are statistically significant for the problem size 30 jobs × 10 stages. For the problem size 50 jobs × 20 stages and $\lambda \geq 0.1$, although the average main effect of PI moves is better than that of SMs, it is found that there is a statistically significant interaction between the neighborhood structure and the number of neighbors, that is, for 20 non-tabu neighbors the SMs become better than PI moves. Hence, in general SMs should be selected as the neighborhood structure for $\lambda \geq 0.005$.

For the size of the tabu list, it can be seen that sizes of 10 and 15 works good, but the size 10 of the tabu list is slightly superior.

Then, the GA approach with a random initial population is studied. The purpose of this study is to determine the favorable GA parameters, *i.e.* population size, crossover types, and mutation types, as well as crossover and mutation rates.

Table 5.11 through 5.13 present the effect of the population size, crossover types, and mutation types by using the average (absolute resp. relative) deviation from the best value as the performance measure.

Table 5.11 The effect of various population sizes on the performance of the genetic algorithm

λ	Problem size	Population size 10	30	50	70
0	10×5	0.168[a]	0.058	**0.045**	0.053
	30×10	1.366	0.786	**0.600**	0.948
	50×20	1.034	0.550	**0.514**	0.752
	Sum	2.568	1.394	**1.159**	1.753
0.001	10×5	1.366[b]	0.786	**0.600**	0.948
	30×10	21.280	13.640	**13.170**	17.530
	50×20	3.255	2.363	**2.301**	3.221
	Sum	25.901	16.789	**16.071**	21.699
0.005	10×5	2.629	1.168	**0.785**	1.108
	30×10	11.081	8.751	**8.708**	10.845
	50×20	3.282	**3.265**	3.282	4.084
	Sum	16.992	13.184	**12.775**	16.037
0.01	10×5	2.349	1.293	**0.937**	0.969
	30×10	8.943	8.133	**7.954**	9.383
	50×20	3.394	**3.249**	3.911	4.315
	Sum	14.686	**12.675**	12.802	14.667
0.05	10×5	1.375	0.734	**0.576**	0.714
	30×10	7.465	**6.827**	6.834	7.672
	50×20	2.492	**2.454**	2.755	3.469
	Sum	11.332	**10.015**	10.165	11.855
0.1	10×5	0.983	0.559	**0.467**	0.612
	30×10	6.636	**6.069**	6.244	6.991
	50×20	**2.081**	2.187	2.595	3.190
	Sum	9.700	**8.815**	9.306	10.793
0.5	10×5	1.187	0.836	**0.771**	0.887
	30×10	6.436	**5.689**	6.063	6.635
	50×20	**1.926**	2.010	2.476	2.986
	Sum	9.549	**8.535**	9.310	10.508
1.0	10×5	1.191	0.843	**0.834**	0.858
	30×10	6.335	**5.608**	5.893	6.578
	50×20	**1.977**	2.107	2.558	3.131
	Sum	9.503	**8.558**	9.285	10.567

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

Table 5.12 The effect of various crossover types on the performance of the genetic algorithm

	Problem	**Crossover types**	
λ	**size**	**PMX**	**OPX**
0	10×5	0.132[a]	**0.030**
	30×10	0.980	**0.871**
	50×20	0.757	**0.667**
	Sum	1.869	**1.568**
0.001	10×5	2.564[b]	**0.794**
	30×10	16.960	**15.850**
	50×20	2.994	**2.575**
	Sum	22.518	**19.219**
0.005	10×5	1.924	**0.922**
	30×10	10.051	**9.642**
	50×20	3.605	**3.351**
	Sum	15.580	**13.915**
0.01	10×5	1.698	**1.077**
	30×10	8.889	**8.318**
	50×20	3.840	**3.595**
	Sum	14.427	**12.990**
0.05	10×5	0.954	**0.745**
	30×10	7.411	**6.988**
	50×20	2.839	**2.746**
	Sum	11.204	**10.479**
0.1	10×5	0.748	**0.563**
	30×10	6.597	**6.373**
	50×20	2.590	**2.437**
	Sum	9.935	**9.373**
0.5	10×5	1.071	**0.770**
	30×10	6.353	**6.059**
	50×20	2.431	**2.268**
	Sum	9.855	**9.097**
1.0	10×5	1.104	**0.759**
	30×10	6.238	**5.969**
	50×20	2.514	**2.372**
	Sum	9.856	**9.100**

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

Table 5.13 The effect of various mutation types on the performance of the genetic algorithm

λ	Problem size	Mutation types: PI	Mutation types: SM
0	10×5	**0.035**[a]	0.127
	30×10	**0.906**	0.944
	50×20	**0.650**	0.775
	Sum	**1.591**	1.847
0.001	10×5	**0.828**[b]	2.530
	30×10	**16.290**	16.510
	50×20	**2.642**	2.928
	Sum	**19.760**	21.968
0.005	10×5	**0.921**	1.924
	30×10	9.970	**9.723**
	50×20	**3.417**	3.539
	Sum	**14.308**	15.186
0.01	10×5	1.666	**1.109**
	30×10	8.779	**8.428**
	50×20	**3.709**	3.726
	Sum	14.154	**13.263**
0.05	10×5	0.948	**0.751**
	30×10	7.451	**6.948**
	50×20	2.872	**2.713**
	Sum	11.271	**10.412**
0.1	10×5	0.798	**0.513**
	30×10	6.716	**6.254**
	50×20	2.631	**2.396**
	Sum	10.145	**9.163**
0.5	10×5	1.106	**0.734**
	30×10	6.356	**6.056**
	50×20	2.488	**2.211**
	Sum	9.950	**9.001**
1.0	10×5	1.102	**0.760**
	30×10	6.251	**5.956**
	50×20	2.563	**2.323**
	Sum	9.916	**9.039**

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

Table 5.14 The effect of various crossover rates on the performance of the genetic algorithm

λ	Problem size	Crossover rates 0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0	10×5	**0.000**[a]	0.028	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**
	30×10	0.556	0.639	0.583	0.639	0.750	0.722	0.694	**0.528**	0.806
	50×20	**0.361**	0.583	0.528	0.500	0.556	0.583	0.778	0.667	0.806
	Sum	**0.917**	1.250	1.111	1.139	1.306	1.306	1.472	1.195	1.611
0.001	10×5	**0.153**[b]	0.179	0.253	0.715	0.199	0.227	0.199	0.274	1.628
	30×10	15.320	12.570	12.400	13.310	11.400	10.090	14.570	10.460	**9.050**
	50×20	2.085	2.139	**1.925**	3.082	2.324	2.669	2.588	2.195	2.725
	Sum	17.558	14.888	14.578	17.107	13.923	12.986	17.357	**12.929**	13.403
0.005	10×5	0.607	0.446	0.423	0.812	0.951	0.897	**0.351**	0.941	1.248
	30×10	9.401	9.544	8.485	8.513	7.896	7.968	**7.379**	8.885	8.432
	50×20	3.513	3.193	3.542	3.336	3.618	3.341	3.341	**3.131**	3.771
	Sum	13.521	13.183	12.450	12.661	12.465	12.206	**11.071**	12.957	13.451
0.01	10×5	**0.639**	0.821	1.042	1.022	1.163	1.361	1.211	1.476	1.747
	30×10	8.457	7.774	7.775	7.711	8.401	7.730	7.691	8.623	**7.104**
	50×20	3.200	2.629	3.226	3.976	3.174	2.554	3.337	3.130	**2.090**
	Sum	12.296	11.224	12.043	12.709	12.738	11.645	12.239	13.229	**10.941**
0.05	10×5	0.744	0.636	0.937	0.921	**0.537**	0.839	0.557	0.816	0.876
	30×10	7.126	6.407	6.503	7.325	6.823	6.986	6.727	6.201	**6.063**
	50×20	2.175	2.798	**1.524**	2.249	2.271	2.383	2.516	2.375	2.111
	Sum	10.045	9.841	**8.964**	10.495	9.631	10.208	9.800	9.392	9.050
0.1	10×5	**0.327**	0.515	0.640	0.874	0.383	0.562	0.377	0.442	0.471
	30×10	6.302	5.832	**5.347**	5.387	6.127	5.813	5.533	6.000	6.174
	50×20	1.812	2.141	2.297	**1.696**	1.875	1.874	2.432	2.252	2.142
	Sum	8.441	8.488	8.284	**7.957**	8.385	8.249	8.342	8.694	8.787
0.5	10×5	0.549	0.700	0.587	0.785	0.725	0.718	0.613	**0.503**	0.706
	30×10	5.813	5.576	6.004	5.308	5.361	**4.808**	5.811	5.529	5.472
	50×20	1.786	1.443	**1.406**	1.767	2.038	2.238	1.931	1.834	2.127
	Sum	8.148	**7.719**	7.997	7.860	8.124	7.764	8.355	7.866	8.305
1.0	10×5	**0.434**	0.536	0.843	0.856	0.746	0.501	0.776	0.484	0.466
	30×10	5.543	5.162	5.679	**4.407**	6.076	5.421	5.583	5.916	5.359
	50×20	1.884	1.799	**1.492**	1.880	2.230	2.360	2.349	1.726	2.081
	Sum	7.861	7.497	8.014	**7.143**	9.052	8.282	8.708	8.126	7.906

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

Table 5.15 The effect of various mutation rates on the performance of the genetic algorithm

λ	Problem size	Mutation rates								
		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0	10×5	**0.000**[a]	**0.000**	**0.000**	**0.000**	**0.000**	0.028	**0.000**	**0.000**	**0.000**
	30×10	0.778	0.639	0.778	0.667	0.583	0.639	0.611	**0.583**	0.639
	50×20	0.806	0.806	0.528	0.472	0.583	0.500	**0.361**	0.528	0.778
	Sum	1.583	1.445	1.306	1.139	1.167	1.167	**0.972**	1.111	1.417
0.001	10×5	0.736[b]	1.642	0.253	0.290	0.208	**0.132**	0.200	0.214	0.154
	30×10	13.570	11.490	11.900	13.970	11.350	11.720	12.890	**10.910**	11.370
	50×20	4.050	2.770	1.844	2.487	2.077	2.709	2.507	**1.487**	1.801
	Sum	18.356	15.902	13.997	16.747	13.635	14.561	15.597	**12.611**	13.325
0.005	10×5	2.536	0.300	0.905	1.214	0.464	0.511	0.401	0.227	**0.115**
	30×10	9.698	7.879	8.136	8.555	7.694	**7.464**	9.042	9.220	8.815
	50×20	3.851	3.615	3.346	3.332	**2.593**	3.568	3.034	3.920	3.528
	Sum	16.085	11.794	12.387	13.101	**10.751**	11.543	12.477	13.367	12.458
0.01	10×5	1.994	1.045	1.558	1.276	0.891	1.153	**0.550**	0.867	1.149
	30×10	7.832	7.668	7.547	8.028	8.281	7.788	**7.313**	7.568	9.240
	50×20	2.825	3.584	**2.349**	2.948	3.354	3.447	2.805	2.910	3.094
	Sum	12.651	12.297	11.454	12.252	12.526	12.388	**10.668**	11.345	13.483
0.05	10×5	1.220	0.856	0.738	0.749	0.748	0.697	0.753	0.751	**0.350**
	30×10	6.706	7.166	6.778	**5.776**	6.729	7.162	6.221	6.952	6.673
	50×20	2.330	2.293	2.221	2.595	1.944	**1.841**	2.202	2.511	2.465
	Sum	10.256	10.315	9.737	**9.120**	9.421	9.700	9.176	10.214	9.488
0.1	10×5	0.887	0.667	0.434	0.440	0.822	0.345	0.313	**0.295**	0.387
	30×10	5.771	5.755	5.897	**5.178**	5.882	6.269	6.069	5.444	6.249
	50×20	2.223	2.164	**1.782**	2.295	2.239	2.042	1.922	1.938	1.916
	Sum	8.881	8.586	8.113	7.913	8.943	8.656	8.304	**7.677**	8.552
0.5	10×5	0.759	0.908	0.745	0.558	0.587	0.618	0.714	0.499	**0.499**
	30×10	5.638	5.861	5.138	5.303	5.305	5.719	6.112	5.483	**5.123**
	50×20	1.895	1.853	1.754	1.829	1.951	1.776	**1.667**	1.849	1.995
	Sum	8.292	8.622	7.637	7.690	7.843	8.113	8.493	7.831	**7.617**
1.0	10×5	0.657	0.684	0.629	**0.413**	0.586	0.565	0.650	0.751	0.708
	30×10	5.579	5.510	**4.990**	5.336	5.548	5.483	6.173	5.428	5.098
	50×20	1.913	2.162	2.079	**1.499**	2.434	1.823	1.723	2.288	1.882
	Sum	8.149	8.356	7.698	**7.248**	8.568	7.871	8.546	8.467	7.688

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

For the GA, it is found that for the population size, crossover types, and mutation types as well as crossover and mutation rates, there are statistically significant differences. The population sizes of 30 and 50 are quite not statistically different, but in general a population size of 30 is slightly better than a population size of 50.

Considering the crossover types, the OPX crossover is obviously superior to the PMX crossover. For the mutation type, it is clear that SMs are better than PI moves for $\lambda \geq 0.01$, whereas the PI moves are mostly better than SMs for the other values. This means that the PI moves are suitable for problems whose tardiness portion dominates the makespan portion. Consequently, the neighborhood structures should be based on PIs for $\lambda < 0.01$, and on shifts of jobs otherwise.

Given the selected GA parameters, the crossover and mutation rates are analyzed again. The results are shown in Table 5.14 through Table 5.15. There are no significant differences for these parameters. However, in general a crossover rate of 0.6 and a mutation rate of 0.3 are recommended.

Then, the recommended SA, TS, and GA parameters are used to test the choice of an appropriate initial solution (or a part of the initial population for the GA approach) (*see* Section 4.7). The letters before the letters SA (or TS, or GA) denote the heuristic rule used for generating one initial solution for SA (or TS, or GA). For example, LPTSA means that the LPT rule is used as an initial solution for the SA algorithm, RNDTS means that an initial solution in TS is randomly generated, or NEHGA means that one initial solution of the initial population in the GA approach is generated by the NEH rule but the other initial solutions are still randomly generated.

Next, the results compared the LPT, ILPT, NEH, and INEH rules as well as the iterative algorithms with random initial solution or population denoted by RNDSA, RNDTS, and RNDGA).

Table 5.16 Average performance of the constructive and iterative algorithms

λ	Problem Size	LPT	ILPT	NEH	INEH	RNDSA	RNDTS	RNDGA
0	10×5	3.200[a]	1.300	0.700	0.700	**0.000**	0.040	0.100
	30×10	8.400	3.000	2.300	2.300	0.300	**0.260**	0.680
	50×20	8.400	2.500	1.200	1.200	**0.040**	**0.040**	0.400
	Sum	20.000	6.800	4.200	4.200	**0.340**	**0.340**	1.180
0.001	10×5	99.380[b]	26.900	13.320	13.320	**0.030**	0.060	3.430
	30×10	115.570	24.880	36.810	36.810	6.320	**5.880**	9.460
	50×20	35.302	5.650	10.921	10.921	**0.748**	0.917	1.694
	Sum	250.252	57.430	61.051	61.051	7.098	**6.857**	14.584
0.005	10×5	46.353	15.036	7.862	7.862	**0.031**	0.946	1.067
	30×10	56.309	16.126	19.711	19.711	**3.899**	4.167	7.785
	50×20	19.539	6.237	5.427	5.427	**0.946**	1.617	2.512
	Sum	122.201	37.399	33.000	33.000	**4.876**	6.730	11.364
0.01	10×5	31.949	10.704	5.944	5.944	**0.160**	0.565	1.359
	30×10	40.655	17.738	14.730	14.730	**3.497**	4.214	6.731
	50×20	15.380	7.504	4.199	4.199	**0.923**	1.669	2.079
	Sum	87.984	35.946	24.873	24.873	**4.580**	6.448	10.169
0.05	10×5	17.358	5.831	2.936	2.936	**0.015**	0.039	0.701
	30×10	24.011	12.349	8.167	8.167	**3.054**	4.059	6.164
	50×20	11.311	6.766	2.218	2.218	**0.806**	2.187	2.650
	Sum	52.680	24.946	13.321	13.321	**3.875**	6.285	9.515
0.1	10×5	15.885	6.054	2.720	2.720	0.082	**0.077**	0.747
	30×10	21.987	10.890	7.440	7.440	4.098	**3.705**	6.333
	50×20	10.779	5.565	1.755	1.755	**1.099**	1.880	2.266
	Sum	48.651	22.509	11.915	11.915	**5.279**	5.662	9.346
0.5	10×5	15.493	5.523	3.284	3.284	**0.013**	0.052	0.630
	30×10	21.179	10.949	6.291	6.291	**3.089**	3.682	5.769
	50×20	10.547	5.337	1.492	1.492	**0.888**	2.076	2.432
	Sum	47.219	21.809	11.067	11.067	**3.990**	5.810	8.831
1.0	10×5	15.486	5.474	3.279	3.279	**0.041**	0.075	0.607
	30×10	21.079	10.419	6.324	6.324	**3.965**	3.365	5.692
	50×20	10.591	5.365	1.605	1.605	**1.021**	2.012	2.666
	Sum	47.156	21.258	11.208	11.208	**5.027**	5.452	8.965

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

In Table 5.16, the results give the average (absolute resp. percentage) deviation of a particular algorithm from the best solution in these tests. It is found that iterative metaheuristic algorithms can improve the quality of the LPT rule by about 90 percent or even more. Among the iterative algorithms, the results show that for the RNDSA and RNDTS algorithms there are no statistically significant differences, but in general, the RNDSA algorithm is slightly better than the RNDTS algorithm.

5.3.4 Performance of a Choice of an Initial Solution for the Iterative Algorithms

Fourthly, to improve the quality of the solution finally obtained by the iterative algorithms, the initial solution of the iterative algorithms has used the biased-initial solution instead of the random initial solution (*see* Section 4.7). Consequently, the purpose of this experiment is to investigate the influence of the choice of an appropriate initial solution for the SA and TS algorithms and an initial population for the GA algorithm by using the heuristic constructive and improvement algorithms. In addition, for the GA approach, all selected constructive algorithms in parallel as a part of the initial population are used to investigate the influence of the biased parallel initial solutions in the population.

In Table 5.17, the SPT, LPT, S/P, ISPT, ILPT, IS/P, PAL, CDS, NEH, IPAL, ICDS, and INEH rules are selected as the initial solution under consideration. The letter CA denotes the whole group of constructive algorithms considered. The letter C before the letters SA, TS, and GA denote the SA, TS, and GA algorithms using the best of the constructive algorithms as an initial solution. In addition, for the GA approach, some selected algorithms in parallel as a part of the initial population are used. Based on each heuristic group, all solutions in each heuristic group stated above as a part of the initial population (the other initial solutions are still randomly generated) are used. Consequently, there are four new choices of initial populations tested (denoted by MIX1GA, MIX2GA, MIX3GA, and MIX4GA, respectively).

Table 5.17 Average performance of the iterative algorithms with biased initial solutions

λ	Problem size	CA	C-SA	C-TS	C-GA	MIX1GA	MIX2GA	MIX3GA	MIX4GA
0	10×5	1.943[a]	**0.001**	0.006	0.089	0.120	**0.100**	0.040	0.060
	30×10	4.829	0.274	**0.237**	0.794	0.520	0.360	0.780	0.720
	50×20	5.050	**0.014**	0.056	0.474	0.460	0.200	0.160	0.080
	Sum	11.821	**0.290**	0.299	1.357	1.100	0.660	0.980	0.860
0.001	10×5	51.554[b]	**0.096**	0.421	2.487	3.600	2.560	2.630	3.010
	30×10	59.438	5.719	4.909	9.640	8.270	**3.080**	10.540	8.710
	50×20	19.131	0.854	0.923	2.410	2.481	0.488	1.286	**0.344**
	Sum	130.123	6.668	6.253	14.537	14.351	**6.128**	14.456	12.064
0.005	10×5	26.798	**0.206**	0.791	1.433	1.778	1.271	1.303	1.431
	30×10	32.128	4.198	**3.834**	7.380	6.433	4.945	6.990	6.122
	50×20	12.033	0.894	1.463	3.070	3.400	**0.564**	1.557	0.973
	Sum	70.958	**5.298**	6.088	11.883	11.611	6.780	9.850	8.526
0.01	10×5	19.264	**0.205**	0.502	1.044	1.445	0.904	1.233	1.595
	30×10	24.923	**3.781**	4.030	7.693	6.412	5.147	8.697	6.042
	50×20	10.475	**0.924**	1.532	3.362	2.431	1.382	1.515	1.013
	Sum	54.662	**4.910**	6.064	12.099	10.288	7.433	11.445	8.650
0.05	10×5	11.247	0.051	**0.047**	0.844	0.756	0.277	0.547	0.696
	30×10	16.360	**3.745**	4.100	6.521	6.499	5.042	4.872	4.525
	50×20	8.117	**0.891**	2.064	3.207	3.037	2.686	1.484	1.131
	Sum	35.725	**4.687**	6.210	10.572	10.292	8.005	6.903	6.352
0.1	10×5	10.408	0.049	**0.025**	0.608	0.855	0.659	0.476	0.294
	30×10	14.940	4.107	**3.868**	5.762	6.160	4.905	4.527	4.647
	50×20	7.573	1.107	1.963	3.019	2.827	2.983	1.105	**1.103**
	Sum	32.921	**5.263**	5.856	9.389	9.842	8.547	6.108	6.044
0.5	10×5	10.134	**0.020**	0.034	0.640	0.646	0.688	0.418	0.449
	30×10	14.625	**3.316**	3.636	5.808	5.941	4.942	3.647	3.724
	50×20	7.249	**0.757**	1.820	2.764	2.331	2.724	0.961	0.811
	Sum	32.008	**4.093**	5.490	9.212	8.918	8.354	5.026	4.984
1.0	10×5	10.181	**0.036**	0.051	0.582	0.691	0.566	0.492	0.408
	30×10	14.565	3.764	3.554	5.696	5.292	4.530	3.732	**3.255**
	50×20	7.312	0.993	1.809	2.987	2.366	2.861	0.950	**0.875**
	Sum	32.058	4.792	5.414	9.264	8.349	7.957	5.174	**4.538**

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

Table 5.18 Average performance of iterative algorithms with biased initial solutions

λ	Problem size	SA-Based Algorithms						TS-Based Algorithms						GA-Based Algorithms (1)						GA-Based Algorithms (2)			
		LPTSA	ILPTSA	CDSSA	NEHSA	ICDSSA	INEHSA	LPTTS	ILPTTS	CDSTS	NEHTS	ICDSTS	INEHTS	LPTGA	ILPTGA	CDSGA	NEHGA	ICDSGA	INEHGA	MIX1GA	MIX2GA	MIX3GA	MIX4GA
0	10×5	***0.000***[a]	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***	***0.000***	0.020	***0.000***	0.100	0.120	0.120	*0.060*	0.120	*0.060*	0.120	*0.040*	0.100	0.060
	30×10	*0.260*	0.280	*0.260*	0.300	0.300	0.300	0.220	0.220	0.340	0.180	0.300	***0.160***	*0.760*	0.900	0.780	0.880	0.880	0.880	0.520	0.780	*0.360*	0.720
	50×20	***0.000***	***0.000***	***0.000***	***0.000***	0.040	***0.000***	0.060	0.120	0.020	***0.000***	0.040	***0.000***	0.580	0.960	0.580	*0.240*	0.340	*0.240*	0.460	0.160	0.200	*0.080*
	Sum	*0.260*	0.280	*0.260*	0.300	0.340	0.300	0.280	0.340	0.360	0.180	0.360	***0.160***	1.440	1.980	1.480	*1.180*	1.340	*1.180*	1.100	0.980	*0.660*	0.860
0.001	10×5	0.520[b]	***0.010***	0.030	0.020	0.510	0.030	*0.090*	0.570	0.100	0.170	1.060	0.530	2.160	2.990	2.480	*2.070*	3.660	*2.070*	3.600	2.630	*2.560*	3.010
	30×10	5.380	6.050	5.600	5.870	*5.220*	5.880	5.470	4.640	4.520	*4.220*	5.760	*4.220*	10.680	9.310	*8.230*	10.340	8.430	10.340	8.270	10.540	***3.080***	8.710
	50×20	0.954	*0.778*	0.831	0.844	0.802	0.940	1.211	1.287	0.817	*0.615*	0.652	0.622	1.839	3.610	1.933	*1.323*	2.114	1.259	2.481	1.286	0.488	***0.344***
	Sum	6.854	6.838	*6.461*	6.734	6.532	6.850	6.771	6.497	5.437	***5.005***	7.472	5.372	14.679	15.910	*12.643*	13.733	14.204	13.669	14.351	14.456	*6.128*	12.064
0.005	10×5	0.115	0.236	0.531	***0.041***	0.060	0.259	*0.984*	1.279	1.179	1.076	1.103	0.721	*1.134*	1.275	1.437	1.374	1.899	1.374	1.778	1.303	*1.271*	1.431
	30×10	*3.999*	4.203	4.249	4.264	4.727	4.320	3.897	3.785	***3.687***	4.075	4.051	4.183	8.038	7.726	7.481	7.055	*6.482*	7.055	6.433	6.990	*4.945*	6.122
	50×20	0.969	0.866	*0.799*	0.888	0.923	0.917	1.524	2.046	1.175	0.838	0.975	*0.842*	2.464	4.560	2.157	1.426	3.183	*1.426*	3.400	1.557	***0.564***	0.973
	Sum	***5.083***	5.305	5.579	5.193	5.710	5.496	6.405	7.110	6.041	5.989	6.129	*5.746*	11.636	13.561	11.075	9.855	11.564	*9.855*	11.611	9.850	*6.780*	8.526
0.01	10×5	***0.084***	0.163	0.212	0.382	0.086	0.324	0.551	0.481	0.779	0.664	*0.387*	0.706	1.283	1.111	*0.859*	0.873	1.273	0.873	1.445	1.233	*0.904*	1.595
	30×10	3.701	3.957	***3.450***	4.103	4.064	4.104	4.389	4.172	*3.677*	3.892	3.934	3.819	*7.098*	8.423	7.383	8.586	7.142	8.586	6.412	8.697	*5.147*	6.042
	50×20	0.965	0.969	***0.895***	0.936	0.933	0.957	1.245	2.171	1.767	*0.931*	1.517	0.931	2.706	5.873	2.239	*1.498*	4.247	*1.498*	2.431	1.515	1.382	*1.013*
	Sum	4.750	5.089	***4.557***	5.421	5.083	5.385	6.185	6.824	6.223	5.487	5.838	*5.456*	11.087	15.407	*10.481*	10.957	12.662	10.957	10.288	11.445	*7.433*	8.650
0.05	10×5	0.038	*0.032*	0.054	0.049	0.040	0.049	0.033	0.052	***0.029***	0.042	0.054	0.029	*0.567*	0.793	0.926	0.625	0.859	0.625	0.756	0.547	*0.277*	0.696
	30×10	4.176	3.369	3.903	4.016	*3.301*	4.005	4.654	3.823	4.165	***3.250***	4.378	3.356	6.440	8.122	6.803	*4.730*	7.754	*4.730*	6.499	4.872	5.042	*4.525*
	50×20	0.896	0.852	0.918	0.748	0.881	***0.740***	1.963	2.152	2.430	*1.491*	1.894	*1.491*	3.133	5.216	3.199	*1.382*	3.472	*1.382*	3.037	1.484	2.686	*1.131*
	Sum	5.110	4.253	4.875	4.813	***4.222***	4.794	6.650	6.027	6.624	*4.783*	6.326	4.876	10.140	14.131	10.928	*6.737*	12.085	*6.737*	10.292	6.903	8.005	*6.352*
0.1	10×5	*0.033*	0.064	0.038	0.070	0.047	0.070	0.057	0.035	0.025	0.030	0.021	***0.012***	0.688	0.523	0.669	*0.504*	0.540	*0.504*	0.855	0.476	0.659	*0.294*
	30×10	4.794	*3.305*	4.647	3.581	3.705	3.581	4.067	4.061	3.724	3.260	3.786	***3.254***	5.945	6.834	5.444	*4.219*	6.190	*4.219*	6.160	*4.527*	4.905	4.647
	50×20	1.067	1.105	1.168	***0.813***	0.861	0.837	2.142	2.264	2.141	*1.369*	1.994	1.370	2.484	4.394	2.892	1.091	3.441	*1.086*	2.827	1.105	2.983	*1.103*
	Sum	5.894	4.474	5.853	***4.464***	4.613	4.488	6.266	6.360	5.890	4.659	5.801	*4.636*	9.117	11.751	9.005	5.814	10.171	*5.809*	9.842	6.108	8.547	*6.044*
0.5	10×5	0.016	0.014	***0.009***	0.013	0.014	0.013	*0.015*	0.035	0.039	0.039	0.041	0.039	0.787	0.887	0.720	*0.537*	0.671	*0.537*	0.646	*0.418*	0.688	0.449
	30×10	3.228	3.500	3.571	*3.140*	4.027	3.153	3.864	3.721	3.894	***2.854***	3.753	2.904	5.853	7.253	5.490	*3.775*	5.813	*3.775*	5.941	3.647	4.942	*3.724*
	50×20	0.776	0.794	0.826	0.715	0.689	***0.687***	1.895	2.099	2.157	1.170	1.723	*1.157*	2.150	4.194	2.401	*0.982*	3.585	*0.982*	2.331	0.961	2.724	*0.811*
	Sum	4.020	4.308	4.406	3.868	4.730	***3.853***	5.774	5.855	6.090	4.063	5.517	*4.100*	8.790	12.334	8.611	*5.294*	10.069	*5.294*	8.918	5.026	8.354	*4.984*
1.0	10×5	0.020	***0.006***	0.059	0.038	0.035	0.038	0.059	0.065	0.069	0.047	0.024	*0.035*	0.626	0.591	0.604	0.458	*0.387*	0.458	0.691	0.492	0.566	*0.408*
	30×10	4.467	***2.588***	4.071	3.183	3.709	3.148	3.813	3.726	3.763	*3.024*	3.471	3.030	5.461	7.428	5.580	*3.533*	5.579	3.547	5.292	3.732	4.530	*3.255*
	50×20	0.858	1.066	1.184	***0.588***	1.014	***0.588***	1.821	2.068	2.199	*1.198*	1.874	*1.198*	2.367	4.351	2.523	*1.040*	4.303	*1.040*	2.366	0.950	2.861	*0.875*
	Sum	5.345	***3.660***	5.314	3.809	4.758	3.774	5.693	5.859	6.031	4.269	5.369	*4.263*	8.454	12.370	8.707	*5.031*	10.269	5.045	8.349	5.174	7.957	*4.538*

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

It is found that the SA-based algorithms are still particularly recommendable. In addition, the GA-based algorithms can be improved by using a group of biased initial solutions as a part of the initial population instead of all randomly generated or only one biased initial solution.

Concerning the choice of an initial solution, this test only shows the results for the recommended SA, TS, and GA parameters to test an appropriate selection of an initial solution in Table 5.18. It is found that for the SA algorithm, there are no statistically significant differences when using different initial solutions. It is however found that the ILPTSA, NEHSA and INEHSA rules are slightly better than the others in general. Consequently, the ILPTSA, NEHSA and INEHSA algorithms are good choices for the SA algorithm with using a biased initial solution. However, the experiments have shown that there are slightly statistically significant differences for different initial solutions. The NEHTS, INEHTS, NEHGA and INEHGA rules are, however, still good solutions when compared within each group.

5.4 Performance of Algorithms on Small-Sized Test Problems

The purpose of these experiments is to evaluate the performance of some selected algorithms that are proposed in Chapter IV on the small-sized test problems (*see* Section 5.2) that can find the optimal solution in an acceptable time.

For the problem with $\lambda = 0$, the performance of each test for each algorithm is assessed by the absolute deviation of a particular algorithm from the optimal solution by using the following equation:

$$\text{absolute deviation from the optimal solution} = Heu_{sol} - Opt_{sol} \qquad (5.7)$$

where Heu_{sol} is the schedule solution obtained by a given algorithm, and Opt_{sol} is the optimal schedule solution.

For the problem with $\lambda > 0$, the performance of each test for each algorithm is assessed by the percentage deviation of a particular algorithm from the optimal solution by using the following equation:

$$\text{percentage deviation from the best solution } = \frac{Heu_{sol} - Opt_{sol}}{Opt_{sol}} \times 100 \qquad (5.8)$$

where Heu_{sol} is the schedule solution obtained by a given algorithm, and Opt_{sol} is the optimal schedule solution.

In this section, the results of some algorithms for small-sized problems with a number of jobs ranging from three to seven are presented. They give the average deviation from the optimal solution obtained by means of the 0-1 mixed linear integer programming formulation given in Chapter III using a commercial mathematical programming software, CPLEX 8.0.0 and AMPL, with an Intel Pentium 4 2.00GHz CPU with 256 MB of RAM. However, the CPU time is limited to at most 2 hours (if the time limited is exceed, it uses the best solution found instead of an optimal one for the evaluation of the heuristic solution).

In the test, the problems with the same generation of the data (*see* Section 5.2) and $\lambda \in \{0, 0.001, 0.005, 0.01, 0.05, 0.1, 0.5, 1\}$ in the objective function are tested. The constructive algorithms LPT and NEH, the fast improvement algorithms ILPT and INEH, and the iterative metaheuristic algorithms RNDSA, NEHSA, RNDTS, NEHTS, RNDGA, and NEHGA are selected to present the performance of each group of the algorithms.

Table 5.19 Average performance of the constructive, fast improvement, and metaheuristic algorithms for small-sized problems

λ	Problem size	LPT	ILPT	NEH	INEH	RNDSA	NEHSA	RNDTS	NEHTS	RNDGA	NEHGA
0	3 jobs	0.300[a]	**0.050**	0.100	0.100	**0.050**	**0.050**	**0.050**	**0.050**	**0.050**	**0.050**
	4 jobs	0.750	0.250	0.250	0.250	**0.150**	**0.150**	**0.150**	**0.150**	**0.150**	**0.150**
	5 jobs	0.700	0.250	**0.150**	**0.150**	**0.150**	**0.150**	0.190	0.200	**0.150**	**0.150**
	6 jobs	1.200	0.400	0.450	0.450	**0.150**	**0.150**	**0.150**	**0.150**	**0.150**	**0.150**
	7 jobs	1.150	0.500	0.450	0.450	**0.200**	**0.200**	**0.200**	**0.200**	**0.200**	**0.200**
	Sum	4.100	1.450	1.400	1.400	**0.700**	**0.700**	0.740	0.750	**0.700**	**0.700**
0.001	3 jobs	45.270[b]	5.180	8.610	8.610	**4.950**	**4.950**	**4.950**	**4.950**	**4.950**	**4.950**
	4 jobs	88.390	32.670	37.460	37.460	**7.350**	**7.350**	7.500	7.480	**7.350**	**7.350**
	5 jobs	23.570	2.320	5.700	5.700	**1.870**	**1.870**	2.270	2.320	**1.870**	**1.870**
	6 jobs	97.960	33.860	27.320	27.320	**13.370**	**13.370**	**13.370**	**13.370**	**13.370**	**13.370**
	7 jobs	23.960	7.140	7.200	7.200	**0.970**	**0.970**	**0.970**	**0.970**	**0.970**	**0.970**
	Sum	279.150	81.170	86.290	86.290	**28.510**	**28.510**	29.060	29.090	**28.510**	**28.510**
0.005	3 jobs	14.218	3.797	5.367	5.367	**3.569**	**3.569**	3.771	4.111	**3.569**	**3.569**
	4 jobs	29.860	9.370	12.480	12.480	**4.400**	**4.400**	4.410	4.420	**4.400**	**4.400**
	5 jobs	15.182	2.383	3.756	3.756	**1.627**	**1.627**	**1.627**	**1.627**	**1.627**	**1.627**
	6 jobs	31.400	8.990	9.070	9.070	**3.440**	**3.440**	3.480	3.480	**3.440**	**3.440**
	7 jobs	14.180	4.960	4.970	4.970	**2.240**	**2.240**	**2.240**	**2.240**	**2.240**	**2.240**
	Sum	104.840	29.500	35.643	35.643	**15.276**	**15.276**	15.528	15.878	**15.276**	**15.276**
0.01	3 jobs	9.820	3.371	4.379	4.379	**3.143**	**3.143**	3.550	3.550	**3.143**	**3.143**
	4 jobs	20.475	5.812	8.182	8.182	**3.350**	**3.350**	3.423	3.370	**3.350**	**3.350**
	5 jobs	11.884	2.334	2.899	2.899	**1.450**	**1.450**	**1.450**	**1.450**	**1.450**	**1.450**
	6 jobs	20.018	5.542	5.677	5.677	**2.154**	**2.154**	2.165	**2.154**	2.164	2.155
	7 jobs	10.528	2.917	2.903	2.903	**1.096**	**1.096**	**1.096**	**1.096**	1.102	**1.096**
	Sum	72.725	19.976	24.040	24.040	**11.193**	**11.193**	11.684	11.620	11.209	11.194
0.05	3 jobs	7.048	3.388	3.267	3.267	**2.887**	**2.887**	3.046	2.934	**2.887**	**2.887**
	4 jobs	12.651	3.366	4.521	4.521	**2.084**	**2.084**	**2.084**	**2.084**	**2.084**	**2.084**
	5 jobs	8.846	2.576	1.778	1.778	**0.899**	**0.899**	**0.899**	**0.899**	**0.899**	**0.899**
	6 jobs	9.344	2.516	3.256	3.256	**1.042**	**1.042**	**1.042**	**1.042**	1.047	1.047
	7 jobs	6.291	0.960	0.650	0.650	**0.192**	**0.192**	**0.192**	**0.192**	0.229	**0.192**
	Sum	44.180	12.806	13.472	13.472	**7.104**	**7.104**	7.263	7.151	7.146	7.109
0.1	3 jobs	6.828	3.343	3.149	3.149	**2.929**	**2.929**	3.011	3.011	**2.929**	**2.929**
	4 jobs	12.301	3.529	4.315	4.315	**2.059**	**2.059**	**2.059**	**2.059**	**2.059**	**2.059**
	5 jobs	8.705	2.417	1.574	1.574	**0.889**	**0.889**	**0.889**	**0.889**	**0.889**	**0.889**
	6 jobs	9.172	2.656	2.186	2.186	**0.949**	**0.949**	**0.949**	**0.949**	**0.949**	**0.949**
	7 jobs	5.906	0.615	0.335	0.335	**0.146**	**0.146**	**0.146**	**0.146**	0.184	0.155
	Sum	42.912	12.560	11.559	11.559	**6.972**	**6.972**	7.054	7.054	7.010	6.981
0.5	3 jobs	6.760	3.475	3.237	3.237	**3.123**	**3.123**	3.295	3.360	**3.123**	**3.123**
	4 jobs	12.063	2.776	4.137	4.137	**1.921**	**1.921**	1.945	**1.921**	**1.921**	**1.921**
	5 jobs	8.531	2.215	1.548	1.548	**0.745**	**0.745**	**0.745**	**0.745**	**0.745**	**0.745**
	6 jobs	8.607	2.620	2.514	2.514	**1.238**	**1.238**	**1.238**	**1.238**	1.269	**1.238**
	7 jobs	6.840	1.581	0.391	0.391	**0.113**	**0.113**	**0.113**	**0.113**	0.149	**0.113**
	Sum	42.801	12.667	11.827	11.827	**7.140**	**7.140**	7.336	7.377	7.207	**7.140**
1.0	3 jobs	6.750	3.492	3.247	3.247	**3.142**	**3.142**	3.182	3.191	**3.142**	**3.142**
	4 jobs	12.038	2.745	4.107	4.107	**1.877**	**1.877**	**1.877**	**1.877**	**1.877**	**1.877**
	5 jobs	9.458	3.192	2.441	2.441	**1.712**	**1.712**	**1.712**	**1.712**	**1.712**	**1.712**
	6 jobs	9.368	2.811	2.747	2.747	**1.518**	**1.518**	**1.518**	**1.518**	1.545	1.531
	7 jobs	7.834	2.139	0.843	0.843	**0.297**	**0.297**	0.332	**0.297**	0.315	0.306
	Sum	45.448	14.379	13.385	13.385	**8.546**	**8.546**	8.621	8.595	8.591	8.568

[a] average absolute deviation for $\lambda = 0$, [b] average percentage deviation for $\lambda > 0$

From the results in Table 5.19, it can be observed that the ILPT improvement algorithm can improve the constructive LPT rule by about 70 percent in terms of the deviation from the optimal value. The performance of the NEH and INEH rules as well as the ILPT rule are not significantly different. As stated above for the large-sized problems, the NEH rule has already embedded the improvement routine as used by the ILPT algorithm. However, in contrast to large-sized problems it can be observed that the ILPT algorithm is slightly better than NEH and INEH for problems with $0.001 \leq \lambda \leq 0.05$.

For the iterative algorithms such as RNDSA, RNDTS, and RNDGA, it is found that they can improve the quality of the solution of the constructive and fast improvement algorithms such as ILPT, NEH, and INEH by about 40 – 70 percent. Similar to the large-sized problems, the SA-based algorithms certainly outperform the other algorithms.

In particular, it can be observed that the average percentage deviation of algorithm RNDSA from the optimal solution for the problems with at least five jobs and $\lambda \geq 0.01$ is usually less than 2% (for the problems with seven jobs and $\lambda \geq 0.05$ even less than 0.3%). The higher percentage deviations for small positive λ values result from the peculiarities of the objective function, namely that, if the number of tardy jobs in the heuristic solution is only greater by one in comparison with an optimal solution, this may lead to a rather large percentage deviation (*see e.g.* problems with six jobs and $\lambda = 0.001$). In addition, it is found that for small-sized problems, a biased initial solution for the iterative algorithm is slightly better than a random initial solution.

5.5 The Recommended Heuristic Solution Algorithm

From the results of the performance of the algorithms stated in the previous sections, in this section a recommended algorithm is concluded. According to the computational results, the INEHSA algorithm is recommended to find the schedule solution for the flexible flow shop scheduling problem with unrelated parallel machines. It consists of four parts, namely the determination of the representatives of the operating time, the modified NEH algorithm, the fast improvement algorithm, and the SA algorithm. The algorithm process flow for the algorithm is shown in Figure 5.1.

The first part is the determination of the representatives of the operating time. Since the actual job operating times are unknown, until an assignment of jobs to machines for the corresponding stage has been done, the representatives of the operating time is necessary to determine. The representative of operating time of job *j* at stage *t* is the sum of the processing time $ps_j^t \big/ v_{ij}^{/t}$ plus the representative of the setup time $s_{lj}^{/t}$, where the representatives of relative machine speed $v_{ij}^{/t}$ and setup time $s_{lj}^{/t}$ for stage *t*, $t = 1, \ldots, k$, use the minimum, maximum and average values of the data. Thus, nine combinations of relative machine speeds and setup times will be used to find the best solution (*see* Section 4.4).

The second part is the construction of the initial schedule solution, which is referred to the constructive algorithm. The modified NEH algorithm is recommended to construct a set of the schedules that is corresponding to the using of the nine combinations of the representatives of operating times. The starting job sequence that gives the best schedule solution is selected from the nine possible starting job sequences (*see* Section 4.4.2.5).

The third part is the improvement algorithm that is applied on the selected starting job sequence from the second part. In this part, the all pairwise interchange approach is applied on the starting job sequence for the jobs that are tardy. The new starting job sequence that gives the best schedule solution is kept (*see* Section 4.5).

The final part is the iterative algorithm, where the staring job sequence that generated from the third part is used as the initial solution for the SA algorithm (*see* Section 4.6.1).

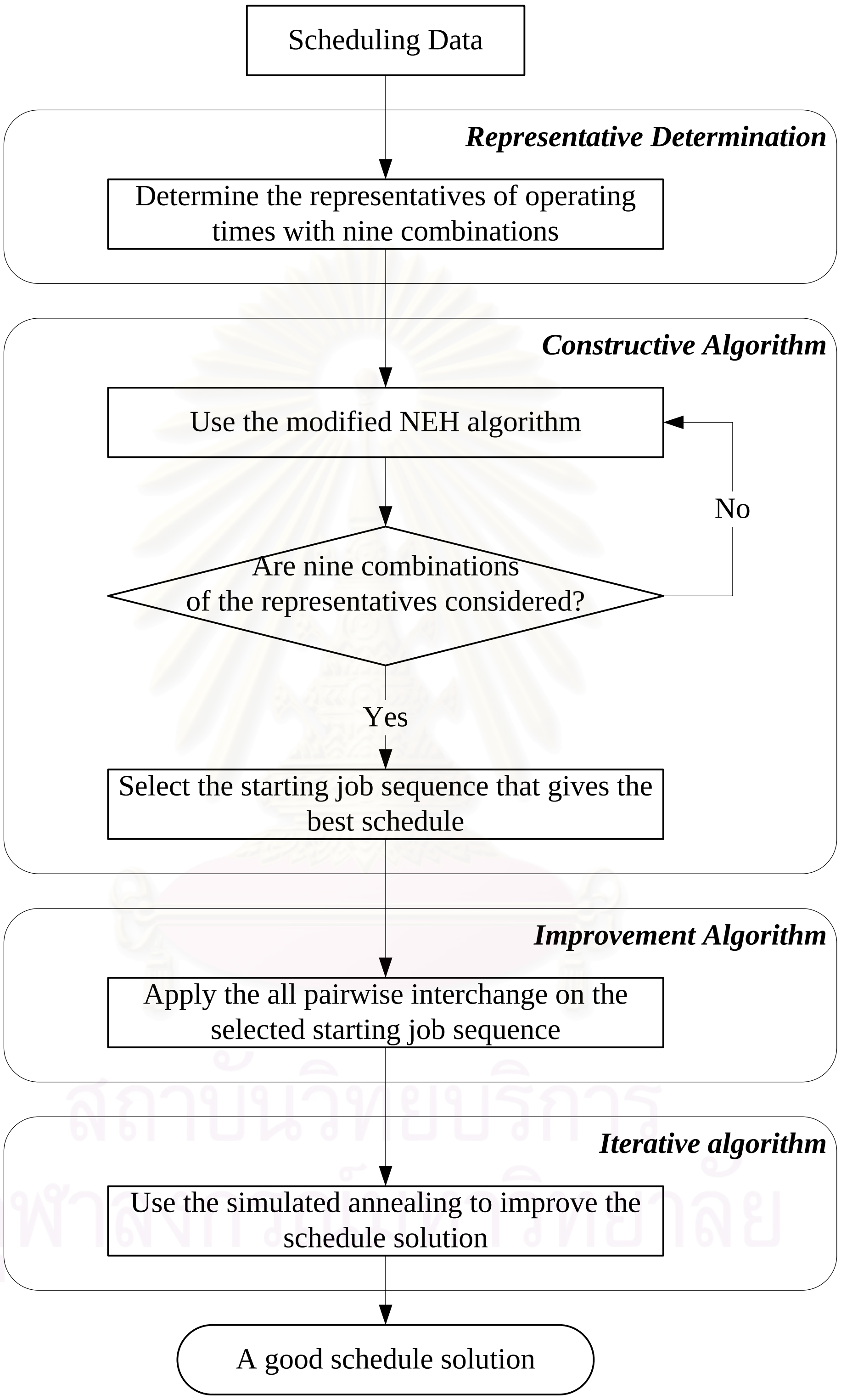

Scheduling Data
Representative Determination
Determine the representatives of operating times with nine combinations
Constructive Algorithm
Use the modified NEH algorithm
No
Are nine combinations of the representatives considered?
Yes
Select the starting job sequence that gives the best schedule
Improvement Algorithm
Apply the all pairwise interchange on the selected starting job sequence
Iterative algorithm
Use the simulated annealing to improve the schedule solution
A good schedule solution

Figure 5.1 Flow chart of the recommended heuristic solution algorithm

5.6 Conclusion

In this chapter, the results of computational experiments of the heuristic algorithms are shown. Firstly, for the constructive algorithms, their performances are compared to each other. It is shown that arranging the jobs in the starting job sequence for the first stage by using the modified NEH algorithm is an excellent constructive algorithm for minimizing the objective function considered. Consequently, the NEH algorithm is most superior to the other constructive algorithms.

To improve the solutions obtained by the constructive algorithms, polynomial heuristic improvement algorithms based on shift moves and pairwise interchanges of jobs are applied. When applying a fast polynomial improvement algorithm, it is found that the all-pairwise-interchange approach is a good improvement algorithm. In addition, the INEH algorithm also provides the best solution.

Next, the performance of the parameters of the iterative algorithms is presented. For the SA parameters, *i.e.* initial temperatures, neighborhood structures, and cooling schedules, it is found that a low initial temperature is slightly preferable (two for $\lambda < 0.5$ and ten otherwise are recommended). The neighborhood structures are based on pairwise interchanges for $\lambda = 0$ and on shifts of jobs otherwise. The Lundy and Mees cooling scheme, $T_{new} = T_{old}/(1+\beta\times T_{old})$, is recommended.

For the TS parameters, *i.e.* neighborhood structures, size of the tabu list, and the number of generated neighbors per iteration, the pairwise interchange neighborhood is better for $\lambda < 0.005$, whereas the shift neighborhood becomes better otherwise. A tabu list size of 10 and the generation of a constant number of 20 nontabu neighbors in each iteration are good choices.

For the GA parameters, it is found that the OPX crossover is clearly superior to the PMX crossover, whereas the SM neighborhood is selected as the mutation operator for problems with $\lambda \geq 0.01$, and the PI neighborhood otherwise is recommended. The algorithm is fixed the crossover and mutation rates at 0.6 and 0.3, respectively.

For the recommended SA, TS, and GA parameters, the performance of these algorithms RNDSA, RNDTA, and RNDGA is investigated. It is found that the RNDSA algorithm outperforms the other algorithms. Then, the influence of the initial solution on these algorithms is studied. The results have shown that the SA-based algorithms are still good algorithms. However, among the SA-, TS- and GA- based algorithms, the NEH and INEH rules are a good choice of an initial solution in general.

Finally, the INEHSA algorithm is recommended as the heuristic solution algorithm for the problem under consideration, where it consists of four parts, namely the creation of the representatives of the operating time, the modified NEH algorithm, the fast improvement algorithm, and the SA algorithm.

CHAPTER VI

CONCLUSION AND FUTURE RESEARCH

Scheduling is the allocation of available and limited production resources to perform a number of jobs over time and to meet certain objectives while respecting a set of constraints (Baker, 1974; Pinedo, 1995; Pinedo and Chao, 1999). The aims of this dissertation is to formulate a mathematical model and to develop approximate algorithms to solve the scheduling problem in the flexible flow shop environment with unrelated parallel machines and sequence-dependent setup times in order to minimize the makespan and the number of tardy jobs. In this chapter, conclusion is presented in the first section. Then, future research is discussed in the next section.

6.1 Conclusion

The dissertation problem is the job scheduling problem in a flexible flow shop environment (*see* Chapter I). The flexible flow shop environment consists of many production stages in series, where jobs have to undergo multiple operations in the same order. However, at least one stage consists of a number of machines in parallel, so jobs have to be processed on any one of the parallel machines in the stage. In addition, processing times to perform a job on each machine in the stage are different. They depend on the jobs and the machines (referred to as *unrelated parallel machines*). Moreover, setup times are considered in the problem. The *sequence-dependent setup time* is the setup duration that depends on the current job and the immediately proceeding job, whereas the *machine-dependent setup time* is the setup duration that depends on the current job and the machine that the job is processed. The machine-dependent setup is the sequence-dependent setup that occurs between the last job in the previous planning period and the first job in the current planning period on the same machine. Moreover, due to the unfinished jobs of the previous planning period, the machines are reserved for processing the unfinished jobs without rescheduling. All data are assumed to be known and constant. The scheduling problem has dual objectives, namely minimizing the makespan and minimizing the number of tardy jobs.

The brief review of machine environments for scheduling problem is presented in Chapter II. The related survey of the problem under consideration is provided. It was found that most literature has studied the flexible flow shop problem with identical parallel machines since 1973 to 2006. Few researchers consider the flexible flow shop with non-identical parallel machines; for example, Soewandi and Elmaghraby (2003) consider the two-stage flexible flow shop with uniform parallel machines, and Low (2005) and Kyparisis and Koulamas (2006) consider the multi stage flexible flow shop with unrelated parallel machines. For the literature in 2007 and 2008, researchers tend to study the flexible flow shop with unrelated parallel machines, *see* Jenabi *et al.* (2007), Low *et al.* (2008), and Ruiz *et al.* (2008).

A 0-1 mixed integer program formulation is provided in Chapter III. It considers three main conditions, namely unrelated parallel machines, sequence-dependent setup time between jobs, and machine-dependent setup time of a job. The optimal solution can be obtained by running the commercial mathematical programming software, CPLEX 8.0.0 and AMPL. It is found that the mathematical model can be used for solving the problems with up to six jobs and four stages in acceptable time. This observation is similar to the recommendations from previous research by Lee and Asllani (2004) in that they have recommended that the 0-1 mathematical programming is practical for the scheduling problem where the number of jobs is lesser than five jobs. In addition, the total number of possible sequence combinations that is generated by using a complete enumeration is estimated. It is concluded that the total number of possible sequence combinations is so large when the number of jobs and/or number of machines per stage and/or number of stages increases. Attempts to find all solutions are unsuccessful as they require too much CPU time. Thus, it is hard to find the optimal solution or even best solution by using either a complete enumeration or a mathematical model.

The main limitation of the exact solution approach is the high memory consumption of finding the optimal solution, so the heuristic approach to find the good solution is suggested as the preferred approach. The proposed heuristic solution concepts consist of three main kinds of the heuristic algorithms, which are the *constructive*, *improvement*, and *iterative* algorithms (*see* Chapter IV).

The schedule construction approach is firstly proposed to evaluate the performance of the schedule and to evaluate the fitness of the solution in each iteration of the iterative algorithms. It is based on the idea of Santos *et al.* (1996). It starts with a *starting job sequence*. The stages are scheduled separately. Considering the jobs in the order of the starting job sequence, each job is loaded on the machine with the minimum completion time in the first considered production stage (referred to as a *greedy search approach*). Then the approach uses the particular rules (*i.e.* the First-In-First-Out rule and the permutation rule) to generate a *new job sequence* for the next production stage. Again, considering the jobs in the new job sequence, each job is loaded on the machine with the minimum completion time in the next considered production stage. Repeat the steps of the approach until all production stages are considered (*see* Section 4.3).

The constructive algorithms described in Section 4.4 are proposed for determining the starting job sequence for the first stage for the problem. They start with finding the representatives of the operation time for each operation. Then, they use the representatives of the operation times to find a starting job sequence for the first stage by using some algorithms and follow with the proposed schedule construction approach to find the schedule result. Both dispatching rules (*i.e.* the SPT, LPT, ERD, EDD, MST, and S/P rules) and flow shop makespan heuristics (*i.e.* the PAL, CDS, GUP, DAN, and NEH) are used to find a starting job sequence for the first stage.

The computational performance of each constructive algorithm is compared to each other. It is shown that arranging the jobs in the starting job sequence for the first stage by using the modified NEH algorithm is an excellent constructive algorithm for minimizing the objective function considered.

Next, to improve the solutions obtained by the constructive algorithms, polynomial heuristic improvement algorithms based on shift moves and pairwise interchanges of jobs are applied (*see* Section 4.5).

When applying a fast polynomial improvement algorithm, it is found that the all-pairwise-interchange approach is good for the neighborhood exchanges in the improvement algorithms. It can improve the quality of a particular constructive solution by about 60 – 80 percent.

Next, the iterative algorithms, namely simulated annealing, tabu search, and genetic algorithms, are developed in Section 4.6. The parameter testing for the iterative algorithms is first conducted. For, the SA parameters, *i.e.* initial temperatures, neighborhood structures, and cooling schedules, it is found that a low initial temperature is slightly preferable (it recommends two for $\lambda < 0.5$, and ten otherwise). The neighborhood structures should be based on pairwise interchanges for $\lambda = 0$, and on shifts of jobs otherwise. The Lundy and Mees cooling scheme, $T_{new} = T_{old}/(1+\beta\times T_{old})$, is recommended.

For the TS algorithm, the TS parameters, *i.e.* neighborhood structures, size of the tabu list, and the number of generated neighbors per iteration, are tested. Similar to the SA algorithm, the pairwise interchange neighborhood is better for $\lambda < 0.005$, whereas the shift neighborhood becomes better otherwise. It can recommend a tabu list size of 10 and the generation of a constant number of 20 nontabu neighbors in each iteration.

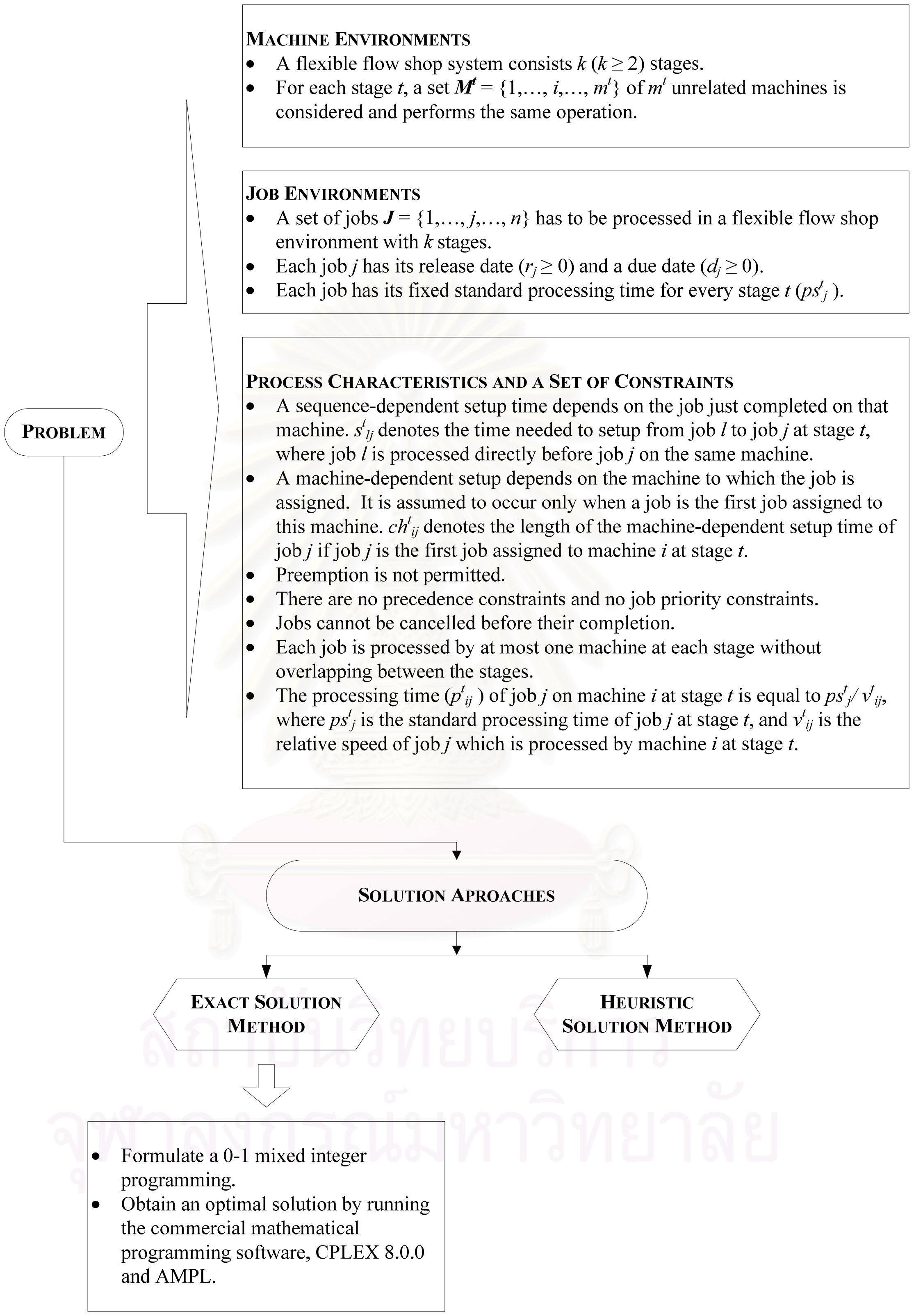
For the GA algorithm, it is found that the OPX crossover is clearly superior to the PMX crossover, whereas it recommends that the SM neighborhood should be selected as the mutation operator for problems with $\lambda \geq 0.01$, and the PI neighborhood otherwise. The crossover and mutation rates at 0.6 and 0.3 are better.

For the recommended SA, TS, and GA parameters, the performance of these algorithms RNDSA, RNDTA, and RNDGA, where an initial solution is randomly generated, are tested. It is found that the RNDSA algorithm outperforms the other algorithms.

Then, the influence of using the initial solution on these algorithms is studied (*see* Section 4.7). The results have shown that the SA-based algorithms are still good algorithms. However, among the SA-, TS- and GA- based algorithms, the NEH and INEH rules are a good choice of an initial solution for such algorithms in general.

Finally, the INEHSA algorithm is recommended as the heuristic solution algorithm for the problem under consideration, where it consists of four parts, namely the determination of the representatives of the operating time, the modified NEH algorithm, the fast improvement algorithm, and the SA algorithm. The first part is to determine the representatives of the operating times, since the actual job operating times are unknown until an assignment of jobs to machines for the corresponding stage has been done. The second part is the construction of the initial schedule solution, which is referred to the modified NEH approach. The modified NEH approach is applied for arranging the jobs in the starting job sequence and uses the schedule construction approach to construct the initial schedule output. The third part is the improvement algorithm that is applied on the selected starting job sequence from the second part. The all-pairwise-interchange approach is applied on the starting job sequence for the jobs that are tardy. The new starting job sequence that gives the best schedule solution is kept. The final part applies the simulated annealing algorithm, where the staring job sequence that is generated from the third part is used as the initial solution for the SA algorithm.

The research summary of this dissertation is depicted in Figure 6.1.



PROBLEM

MACHINE ENVIRONMENTS

- A flexible flow shop system consists k ($k \geq 2$) stages.
- For each stage t, a set $\boldsymbol{M}^t = \{1,\ldots, i,\ldots, m^t\}$ of m^t unrelated machines is considered and performs the same operation.

JOB ENVIRONMENTS

- A set of jobs $\boldsymbol{J} = \{1,\ldots, j,\ldots, n\}$ has to be processed in a flexible flow shop environment with k stages.
- Each job j has its release date ($r_j \geq 0$) and a due date ($d_j \geq 0$).
- Each job has its fixed standard processing time for every stage t (ps^t_j).

PROCESS CHARACTERISTICS AND A SET OF CONSTRAINTS

- A sequence-dependent setup time depends on the job just completed on that machine. s^t_{lj} denotes the time needed to setup from job l to job j at stage t, where job l is processed directly before job j on the same machine.
- A machine-dependent setup depends on the machine to which the job is assigned. It is assumed to occur only when a job is the first job assigned to this machine. ch^t_{ij} denotes the length of the machine-dependent setup time of job j if job j is the first job assigned to machine i at stage t.
- Preemption is not permitted.
- There are no precedence constraints and no job priority constraints.
- Jobs cannot be cancelled before their completion.
- Each job is processed by at most one machine at each stage without overlapping between the stages.
- The processing time (p^t_{ij}) of job j on machine i at stage t is equal to ps^t_j / v^t_{ij}, where ps^t_j is the standard processing time of job j at stage t, and v^t_{ij} is the relative speed of job j which is processed by machine i at stage t.

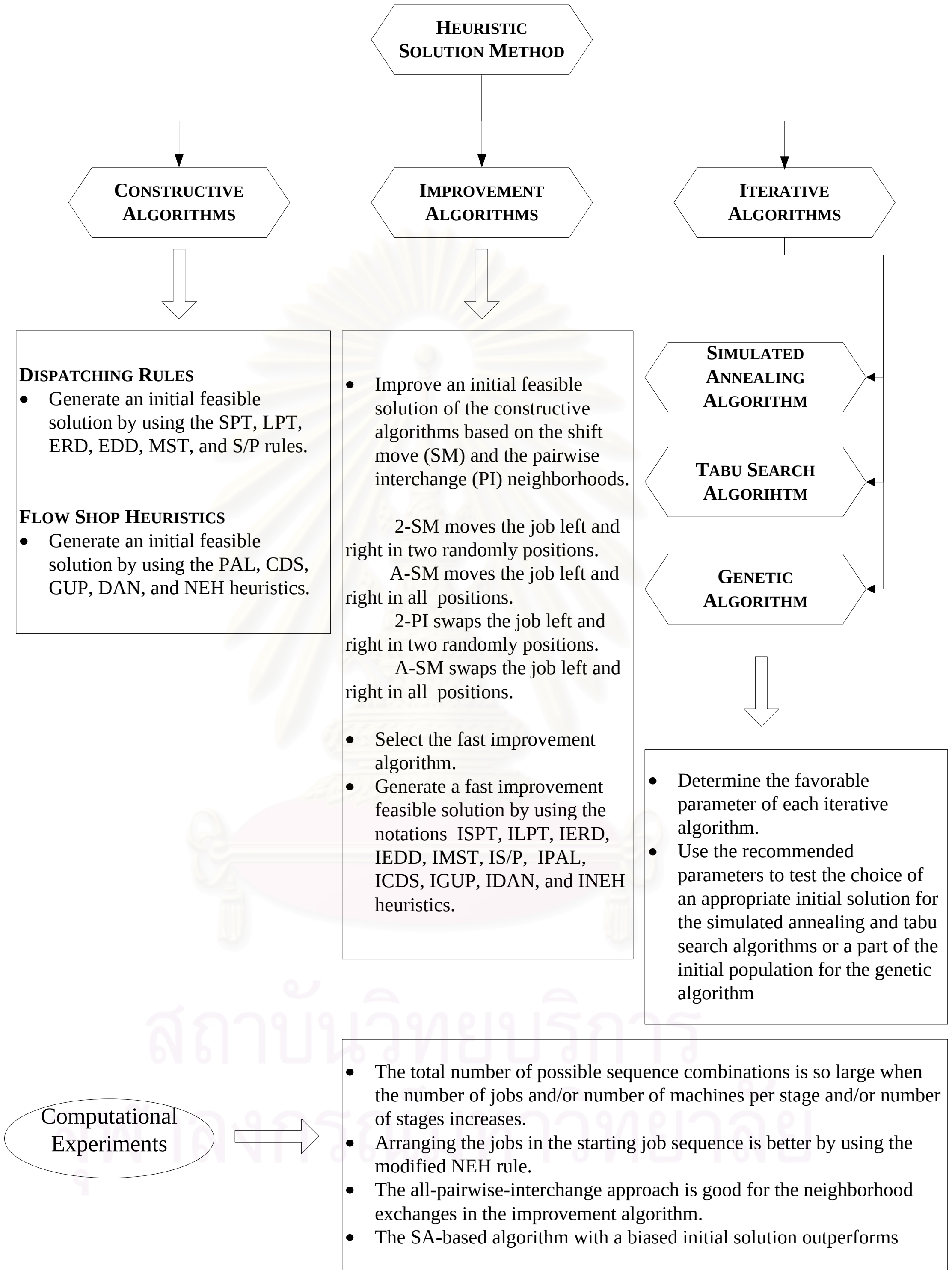
SOLUTION APROACHES

EXACT SOLUTION METHOD

HEURISTIC SOLUTION METHOD

- Formulate a 0-1 mixed integer programming.
- Obtain an optimal solution by running the commercial mathematical programming software, CPLEX 8.0.0 and AMPL.

Figure 6.1 Research summary



DISPATCHING RULES
- Generate an initial feasible solution by using the SPT, LPT, ERD, EDD, MST, and S/P rules.

FLOW SHOP HEURISTICS
- Generate an initial feasible solution by using the PAL, CDS, GUP, DAN, and NEH heuristics.

- Improve an initial feasible solution of the constructive algorithms based on the shift move (SM) and the pairwise interchange (PI) neighborhoods.

2-SM moves the job left and right in two randomly positions.
A-SM moves the job left and right in all positions.
2-PI swaps the job left and right in two randomly positions.
A-SM swaps the job left and right in all positions.

- Select the fast improvement algorithm.
- Generate a fast improvement feasible solution by using the notations ISPT, ILPT, IERD, IEDD, IMST, IS/P, IPAL, ICDS, IGUP, IDAN, and INEH heuristics.

- Determine the favorable parameter of each iterative algorithm.
- Use the recommended parameters to test the choice of an appropriate initial solution for the simulated annealing and tabu search algorithms or a part of the initial population for the genetic algorithm

- The total number of possible sequence combinations is so large when the number of jobs and/or number of machines per stage and/or number of stages increases.
- Arranging the jobs in the starting job sequence is better by using the modified NEH rule.
- The all-pairwise-interchange approach is good for the neighborhood exchanges in the improvement algorithm.
- The SA-based algorithm with a biased initial solution outperforms

Figure 6.1 Research summary (cont.)

6.2 Future Research

This dissertation has studied a flexible flow shop problem with unrelated parallel machines and sequence-dependent setup times. The primary contributions of this dissertation are as follows:

1. To develop a mathematical model that extends the flexible flow shop problem with identical parallel machines to cope with the flexible flow shop problem with unrelated parallel machines which is common to encounter in the real-world situation, and
2. To investigate heuristic algorithms for the flexible flow shop problem with unrelated parallel machines. These algorithms generalize existing procedures from the literature. Moreover, while in the literature for flow shop problems only one optimization criterion is used, this dissertation uses a bicriteria problem. New results are *e.g.* the combination of makespan heuristics with fast polynomial iterative algorithms so that for the generation of the initial solution both criteria are used. Within the individual algorithms new components have been suggested (*e.g.* crossover operators for the genetic algorithms, the schedule construction algorithm, the suggested treatment of the operating times, etc.).

However, the proposed algorithms in this dissertation only determine a starting job sequence as an initial seed for the first stage and use the fixed rules (*i.e.* either FIFO or permutation rules) to find the new job sequence for the next stage. Then, they use the particular approach to assign the jobs to the machines in each stage. Consequently, heuristics that use the different stage sequencing orders will be studied to increase the solution quality further. In addition, the procedure for fixing the processing and setup times as well as the relative speeds in the constructive algorithms can be refined further.

Even though most studies including this dissertation have developed the model to solve the flexible flow shop problem, such a problem remains largely unsolved. The progress in scheduling theory, while advancing at a rapid pace, is not appreciable to solve practical flexible flowshop scheduling problems optimally and efficiently. Moreover, it lacks of integrative and interactive decision making in the business practices. Consequently, it is suggested that the gap between the development of theory and practical applications of theory intend to be bridged. Three areas of future research are suggested: theoretical, computational, and empirical research.

6.2.1 Theoretical Research

The mixed integer programming (MIP) optimization has limited operational capability. However, although the size of most industrial problems exceeds the capability of the proposed model, it can be beneficial to researchers for testing the performance of heuristics designed for multi-criteria problems. Further research in this area is required to develop more efficient computer codes so that the model is able to be applied to slightly or even much larger scheduling problems.

Although the mathematical model can easily provide an optimal solution, it becomes too complex to be used for large scheduling problems. Solving larger instances of the scheduling problems using a commercial solver would be a challenge. Techniques such as branch-and-price (branch-and-bound plus column generation) and branch-and-cut (branch-and-bound plus row generation) may be developed to solve larger instances, that is, theoretical research in a flexible flow shop scheduling problem should attempt to develop a polynomial bounded algorithm to reject quickly a large number of inefficient partial schedules to curtail enumeration scheme.

Moreover, from the results of the numerical examples it is seen that there are usually different optimal solutions for each criterion. Thus, to select an appropriate criterion is necessary for making the decision. Perhaps, it is worth using a multi-objective function to choose an optimal schedule considering different goals.

Simultaneously, more quick but reliable heuristic algorithms should be developed. For example, it can be done to use other iterative algorithms such as ant colony algorithms. The choice of good parameters for them should be tested. In addition, the influence of the starting solution should be investigated.

Consideration of hybrid algorithms for the flexible flow shop problem provides other fruitful areas for future theoretical research; for instance, hybrid algorithms should be developed by using a local search algorithm within a GA approach. This means that, after generating an offspring, this solution should be improved by applying for instance tabu search or simulated annealing before applying the selection criterion of the GA approach.

6.2.2 Computational Research

It is difficult to select an algorithm in solving a given flexible flow shop problem. The future computational research should consider such aspects as comparative efficiency of a wide range of algorithms for a specified problem with given data. Consequently, new measures of computational effort should be developed. In addition, it is possible to use artificial intelligence techniques, such as neural networks, to select a good specific heuristic to be used for a given flexible flow shop problem (Gupta, Sexton, and Tunc, 2000).

6.2.3 Empirical Research

The mathematical theory of flexible flow shop scheduling suffers from too much abstraction and too little application. Consequently, it seems to be motivated by how the practical research of flexible flow shop can be used. Despite a few decade of research, the practical use of flexible flow shop is rare. Perhaps, future research in flexible flow shop scheduling should be inspired more by real life problems rather than problems encountered in mathematical abstractions.

For a realistic problem formulation, empirical research is necessary to understand the practical situations. The flexible flowshop scheduling is only one of a few areas where no case histories are available. Empirical research should answer such questions as (Gupta and Stafford, 2006):

1. What is the maximum problem size encountered in practice?
2. What specific situations give rise to flowshop scheduling problems?
3. What are the desired objectives of scheduling?
4. What is the nature of processing times?
5. How rigid (or flexible) are the operating policies?

Moreover, for heuristic algorithms, although the scheduling problems are tested on a wide range of heuristic algorithms, it is necessary to obtain the industrial data to further validate performance.

REFERENCES

Adler, L., Fraiman, N., Kobacker, E., Pinedo, M., Plotnicoff, J. C., and Wu, T. P. 1993. BPSS: A scheduling support system for the packaging industry. **Operations Research** 41(4): 641-648.

Alisantoso, D., Khoo, L. P., and Jiang, P. Y. 2003. An immune algorithm approach to the scheduling of a flexible PCB flow shop. **The International Journal of Advanced Manufacturing Technology** 22(11-12): 819-827.

Allahverdi, A., Gupta, J. N. D., and Aldowaisan, T. 1999. A review of scheduling research involving setup considerations. **Omega**. 27(2): 219-239.

Allahverdi, A., Ng, C. T., Cheng, T. C. E., and Kovalyov, M. Y. 2008. A survey of scheduling problems with setup times or costs. **European Journal of Operational Research** 187(3): 985-1032.

Allaoui, H., and Artiba, A. 2004. Integrating simulation and optimization to schedule a hybrid flow shop with maintenance constraints. **Computers & Industrial Engineering** 47(4): 431-450.

Azizoğlu, M., Cakmak, E., and Kondakci, S. 2001. A flexible flowshop problem with total flow time minimization. European Journal of Operational Research 132(3): 528-538.

Azizoğlu, M., and Kirca, O. 1999. On the minimization of total weighted flow time with identical and uniform parallel machines. **European Journal of Operational Research** 113(1): 91-100.

Baker, K. R. 1974. **Introduction to Sequencing and Scheduling**. 1st ed. Canada: John Wiley & Sons.

Barr, R., Golden, B., Kelly, J., Rescende, M., and Stewart, W. 1995. Designing and Reporting on Computational Experiments with Heuristic Methods. **Journal of Heuristics** 1(1): 9-32.

Belouadah, H., and Potts, C. N. 1994. Scheduling identical parallel machines to minimize total weighted completion time. **Discrete Applied Mathematics** 48(3): 201-218.

Bernardo, J. J., and Lin, K. S. 1994. An interactive procedure for bi-criteria production scheduling. Computers & Operations Research 21(6): 677-688.

Bertel, S., and Billaut, J. C. 2004. A genetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. **European Journal of Operational Research** 159(3): 651-662.

Bhattacharya, S., and Bose, S. K. 2007. Mathematical model for scheduling operations in cascaded continuous processing units. **European Journal of Operational Research** 182(1): 1-14.

Blum, C., and Roli, A. 2003 Metaheuristics in combinatorial optimization: Overview and conceptual comparison. **ACM Computing Surveys.** 35(3): 268-308.

Bolat, A., Al-Harkan, I., and Al-Harbi, B. 2005. Flow-shop scheduling for three serial stations with the last two duplicate. **Computers & Operations Research** 32(3): 647-667.

Botta-Genoulaz, V. 2000. Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. **International Journal of Production Economics** 64(1-3): 101-111.

Brah, S.A., and Hunsucker, J. L. 1991. Branch and bound algorithm for the flow shop with multiple processors. **European Journal of Operational Research** 51(1): 88–99.

Brah, S. A., and Loo, L. L. 1999. Heuristics for scheduling in a flow shop with multiple processors. **European Journal of Operational Research** 113(1): 113-122.

Bräsel, H., Herms, A., Mörig, M., Tautenhahn, T., Tusch, J., Werner, F., and Willenius, P. 2006. A Comparison of Heuristics for Mean Flow Time Open Shop Scheduling, In A. Dolgui, G. Morel, C. E. Pereira (eds.), **Proceedings of the 12th IFAC Symposium on Control Problems in Manufacturing**, pp. 119-124. Vol. 3, St. Etienne/France.

Bratley, P., Florian, M., and Robillard, P. 1975. Scheduling with earliest start and due date constraints on multiple machines. **Naval Research Logistics Quarterly** 22(1): 165-173.

Burns, L. D., and Daganzo, C. F. 1987. Assembly line job sequencing principles. **International Journal of Production Research**. 25(1): 71-99.

Campbell, H. G., Dudek, R. A., and Smith, M. L. 1970. A heuristic algorithm for the *n* Job *m* Machine sequencing problem. **Management Science** 16(10): B630-B637.

Chen, B. 1995. Analysis of classes of heuristics for scheduling a two-stage flow shop with parallel machines at one stage. **Journal of the Operational Research Society** 46(2): 234-244.

Cheng, R., Gen, M., and Tozawa, M. 1995. Minmax earliness/tardiness scheduling in identical parallel machine system using genetic algorithms. **Computers & Industrial Engineering** 29(1-4):513-517.

Chung, C. S., Flynn, J., and Kirca, O. 2002. A branch and bound algorithm to minimize the total flow time for *m*-machine permutation flowshop problem. **International Journal of Production Economics** 79(3): 185-196.

Conway, R. W., Maxwell, W. L., and Miller, L. W. 1967. **Theory of Scheduling** Reading, Massachusetts: Addison-Wesley.

Cook, W. J., Cunningham, W. H., Pulleyblank, W. R, and Schrijver, A. 1998. **Combinatorial Optimization**. 1st ed. New York: John Wiley & Sons.

Damodaran, P., and Srihari, K. 2004. Mixed Integer Formulation to Minimize Makespan in a Flow Shop with Batch Processing Machines. **Mathematical and Computer Modelling** 40(13): 1465-1472.

Dannenbring, D. G. 1977. An evaluation of flow shop sequencing heuristics. **Management Science** 23(11):1174-1182.

Day, J. E., and Hottenstein, M. P. 1970. Review of Sequencing Research. **Naval Research Logistics Quarterly**. 17(1): 11-39.

Deal, D. E., and Hunsucker, J. L. 1991. The two-stage flowshop scheduling problem with m machines at each stage. **Journal of Information and Optimization Sciences** 12 (3): 407-417.

Dessouky, M. M., Dessouky, M. I., and Verma, S. K. 1998. Flowshop scheduling with identical jobs and uniform parallel machines. **European Journal of Operational Research** 109(3): 620-631.

Elmaghraby, S. E., and Park, S. H. 1974. Scheduling jobs on a number of identical machines. **AIIE Transactions** 6(1): 1-13.

Engin, O., and Döyen, A. 2004. A new approach to solve hybrid flow shop scheduling problems by artificial immune system. **Future Generation Computer Systems** 20(6): 1083-1095.

Eren, T. 2007. A multicriteria flowshop scheduling problem with setup times. **Journal of Materials Processing Technology** 186(1-3): 60-65.

Finke, D. A., and Medeiros, D. J. 2002. Shop scheduling using tabu search and simulation. **Proceedings of the 2002 Winter Simulation Conference**: 1013-1017.

Framinan, J. M., Gupta J. N. D., and Leisten, R. 2004. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. **Journal of the Operational Research Society** 55(12): 1243-1255.

Gagliardi, F. 2007. Some Issues About Cognitive Modelling and Functionalism. In R. Basili and M.T. Pazienza (eds.), **AI*IA 2007: Artificial Intelligence and Human-Oriented Computing**, pp. 60-71. Berlin: Springer-Verlag.

Garey, M. R., and Johnson, D. S. 1979. **Computers and Intractability: A Guide to the Theory of NP-Completeness.** San Francisco: W. H. Freeman and Company.

Gen, M., and Cheng, R. 1997. **Genetic Algorithms and Engineering Design**. 1st ed. New York: John Wiley & Sons.

Glover, F. 1986. Future paths for integer programming and links to artificial intelligence. **Computers & Operations Research** 13(5): 533-549.

Glover, F., and Laguna, M. 1993. Tabu search. In C. R. Reeves (ed.), **Modern heuristic techniques for combinatorial problems**, pp. 70-150. chapter 3. Oxford: Blackwell Scientific Publications.

Gourgand, M., Grangeon, N., and Norre, S. 1999. Metaheuristics for the deterministic hybrid flowshop problem. **Proceeding of the international conference on industrial engineering and production management (IEPM'99)**, Glasgow, United Kingdom, July 12-15 1999: 136-145.

Grabowski, J., and Wodecki, M. 2004. A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. **Computers & Operations Research.** 31(11): 1891-1909.

Grangeon, N., Tanguy, A., and Tchernev, N. 1999. Generic simulation model for hybrid flow-shop. **Computers & Industrial Engineering** 37(1-2): 207-210.

Guinet A., Echalier, F., and Dussauchoy, A. 1992. Scheduling jobs on parallel machines: a survey. **EURO XII/TIMS XXXI Joint International Conference**, Helsinki, Finland

Guinet, A. G. P., and Solomon, M. M. 1996. Scheduling hybrid flowshops to minimize maximum tardiness or maximum completion time. **International Journal of Production Research** 34(6): 1643-1654.

Guinet, A., Solomon, M. M., Kedia, P. K., and Dussauchoy, A. 1996. A Computational study of heuristics for two-stage flexible flowshops. **International Journal of Production Research** 34(5): 1399-1415.

Guirchoun, S., Martineau, P., and Billaut, J. C. 2005. Total completion time minimization in a computer system with a server and two parallel processors. **Computers & Operations Research** 32(3): 599-611.

Gupta, J. N. D. 1971. A functional heuristic algorithm for the flowshop scheduling problem. **Operations Research Quarterly** 22(1): 39-47.

Gupta, J. N. D. 1988. Two-stage, hybrid flow shop scheduling problem. **Journal of the Operational Research Society** 39(4): 359-364.

Gupta, J. N. D., Hariri, A. M. A., and Potts, C. N. 1997. Scheduling a two-stage hybrid flow shop with parallel machines at the first stage. **Annals of Operations Research** 69: 171-191.

Gupta, J. N. D., Krüger, K., Lauff, V., Werner, F., and Sotskov, Y. N. 2002. Heuristics for hybrid flow shops with controllable processing times and assignable due dates. **Computers & Operations Research** 29(10): 1417-1439.

Gupta, J. N. D., and Ruiz-Torres, A. J. 2005. Generating efficient schedules for identical parallel machines involving flow-time and tardy jobs. **European Journal of Operational Research**. 167(3): 679-695.

Gupta, J. N. D., Sexton, R. S., and Tunc, E. A. 2000. Selecting scheduling heuristics using neural networks. **INFORMS Journal on Computing**. 12(2): 150-162.

Gupta, J. N. D., and Stafford Jr, E. F. 2006. Flowshop scheduling research after five decades. **European Journal of Operational Research** 169(3): 699-711.

Gupta, J. N. D., and Tunc, E. A. 1991. Schedules for a two-stage hybrid flowshop with parallel machines at the second stage. **International Journal of Production Research** 29(7): 1489-1502.

Gupta, J. N. D., and Tunc, E. A., 1994. Scheduling a two-stage hybrid flowshop with separable setup and removal times. **European Journal of Operational Research** 77(3): 415-428.

Gupta, J. N. D., and Tunc, E. A. 1998. Minimizing tardy jobs in a two-stage hybrid flowshop. **International Journal of Production Researc**h 36(9): 2397-2417.

Haouari, M., Hidri, L., and Gharbi, A. 2006. Optimal scheduling of a two-stage hybrid flow shop. **Mathematical Methods of Operations Research**. 64(1): 107-124.

Haouari, M., and M'Hallah, R. 1997. Heuristic algorithms for the two-stage hybrid flowshop problem. **Operations Research Letters** 21(1): 43-53.

Harjunkoski, I., and Grossmann, I. E. 2002. Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. **Computers & Chemical Engineering** 26(11): 1533-1552.

Heizer, J., and Render, B. 2001. **Operations Management**. 6th ed. New Jersey. Prentice Hall.

Hejazi, S. R., and Saghafian, S. 2005. Flowshop-scheduling problems with makespan criterion: a review. **International Journal of Production Research** 43(14): 2895-2929.

Hino, C. M., Ronconi, D. P., and Mendes, A. B. 2005. Minimizing earliness and tardiness penalties in a single-machine problem with a common due date. **European Journal of Operational Research** 160(1): 190-201.

Ho, J. C., and Chang, Y. L. 1995. Minimizing the number of tardy jobs for m parallel machines. **European Journal of Operational Research** 84(2): 343-355.

Holland, J. H. 1975. **Adaptation in natural and artificial systems**. Ann Arbor. University of Michigan.

Holthaus, O., and Rajendran, C. 1997. New dispatching rules for scheduling in a job shop — An experimental study. **The International Journal of Advanced Manufacturing Technology** 13(2): 148-153.

Hoogeveen, J. A., Lenstra, J. K., and Veltman, B. 1996. Preemptive scheduling in a two-stage multiprocessor flow shop is NP-hard. **European Journal of Operational Research** 89 (1): 172-175.

Hsieh, J. C., Chang, P. C., and Hsu, L. C. 2003. Scheduling of drilling operations in printed circuit board factory. **Computers & Industrial Engineering**. 44(3): 461-473.

Iyer, S. K., and Saxena, B. 2004. Improved genetic algorithm for the permutation flowshop scheduling problem. **Computers & Operations Research** 31(4): 593-606.

Jackson, J. R. 1955. Scheduling a production line to minimize maximum tardiness. **Management Science Research Project** Research Report 43. Los Angeles. CA: University of California.

Janiak, A., Kozan, E., Lichtenstein, M., and Oğuz, C. 2007. Metaheuristic approaches to the hybrid flow shop scheduling problem with a cost-related criterion. **International Journal of Production Economics** 105(2): 407-424.

Jenabi, M., Fatemi Ghomi, S. M. T., Torabi, S. A., and Karimi, B. 2007. Two hybrid meta-heuristics for the finite horizon ELSP in flexible flow lines with unrelated parallel machines. **Applied Mathematics and Computation** 186(1): 230-245.

Jin, Z., Yang, Z., and Ito, T. 2006. Metaheuristic algorithms for the multistage hybrid flowshop scheduling problem. **International Journal of Production Economics** 100(2): 322-334.

Johnson, S. M. 1954. Optimal two- and three-stage production schedules with setup times included. **Naval Research Logistics Quarterly** 1(1): 61-68.

Jones, D. F., Mirrazavi, S. K., and Tamiz, M. 2002. Multi-objective meta-heuristics: An overview of the current state-of-art. **European Journal of Operational Research** 137(1): 1-9.

Jungwattanakit, J., Reodecha, M., Chaovalitwongse, P., and Werner, F. 2005. An evaluation of sequencing heuristics for flexible flowshop scheduling problems with unrelated parallel machines and dual criteria. **Otto-von-Guericke-Universitat Magdeburg Preprint** 05(28): 1-23.

Karacapilidis, N. I., and Pappis, C. P. 1996. Production planning and control in textile industry: A case study. **Computers in Industry** 30(2): 127-144.

Kim, J. S., Kang, S. H., and Lee, S. M. 1997. Transfer batch scheduling for a two-stage flowshop with identical parallel machines at each stage. **Omega** 25(5): 547-555.

Kirkpatrick, S., Gelatt, Jr. C. D., and Vecchi, M. P. 1983. Optimization by simulated annealing. **Science** 220(4598): 671-680.

Koulamas, C., Antony, S. R., and Jaen, R. 1994. A survey of simulated annealing applications to operations research problems. **Omega International Journal of Management Science** 22(1):41-56.

Kumar, N. S. H., and Srinivasan, G. 1996. A genetic algorithm for job shop scheduling – a case study. **Computers in Industry** 31(2): 155-160.

Kurz, M. E., and Askin, R. G. 2003. Comparing scheduling rules for flexible flow lines. **International Journal of Production Economics** 85(3): 371-388.

Kurz, M. E., and Askin, R. G. 2004. Scheduling flexible flow lines with sequence-dependent setup times. **European Journal of Operational Research** 159(1): 66-82.

Kyparisis, G. J., and Koulamas, C. 2006. Flexible flow shop scheduling with uniform parallel machines. **European Journal of Operational Research** 168(3): 985-997.

Lee, C. Y., and Vairaktarakis, G. L. 1994. Minimizing makespan in hybrid flowshops. **Operations Research Letters** 16(3): 149-158.

Lee, S. M., and Asllani, A. A. 2004. Job scheduling with dual criteria and sequence-dependent setups: Mathematical versus genetic programming. **Omega** 32(2): 145-153.

Lin, B. M. T., and Jeng, A. A. K. 2004. Parallel-machine batch scheduling to minimize the maximum lateness and the number of tardy jobs. **International Journal of Production Economics** 91(2): 121-134.

Lin, Hung-Tso., and Liao, Ching-Jong. 2003. A case study in a two-stage hybrid flow shop with setup time and dedicated machines. **International Journal of Production Economics** 86(2): 133-143.

Linn, R., and Zhang, W. 1999. Hybrid flow shop scheduling: A survey. **Computers & Industrial Engineering** 37(1-2): 57-61.

Liu, C. Y., and Chang, S. C. 2000. Scheduling flexible flow shops with sequence-dependent setup effects. **IEEE Transactions on Robotics and Automation** 16(4): 408-419.

Logendran, R., deSzoeke, P., and Barnard, F. 2006. Sequence-dependent group scheduling problems in flexible flow shops. **International Journal of Production Economics** 102(1): 66-86.

Loukil, T., Teghem, J., and Tuyttens, D. 2005. Solving multi-objective production scheduling problems using metaheuristics. **European Journal of Operational Research**. 161(1): 42-61.

Low, C. 2005. Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. **Computers & Operations Research** 32(8): 2013-2025.

Low, C., Hsu, C. J., and Su, C. T. 2008. A two-stage hybrid flowshop scheduling problem with a function constraint and unrelated alternative machines. **Computers & Operations Research** 35(3): 845-853.

Lundy, M., and Mees, A. 1986. Convergence of an annealing algorithm. **Mathematical Programming** 34(1): 111–124.

Mansouri, S. A. 2006. A simulated annealing approach to a bi-criteria sequencing problem in a two-stage supply chain. **Computers & Industrial Engineering** 50(1-2): 105-119.

Markland, R. E., Darby-Dowman, K. H., and Minor, E. D. 1990. Coordinated production scheduling for make-to-order manufacturing. **European Journal of Operational Research** 45(2-3): 155-176.

Moore, J. M. 1968. An *n* job, one machine sequence algorithm for minimizing the number of late jobs. **Management Science** 15(1): 102-109.

Moursli, O., and Pochet, Y. 2000. A Branch-and-bound algorithm for the hybrid flowshop. **International Journal of Production Economics** 64(1-3): 113-125.

Murata, T., Ishibuchi, H., and Tanaka, H. 1996. Multi-objective genetic algorithm and its applications to flowshop scheduling. **Computers & Industrial Engineering** 30(4): 957-968.

Murtadi, A., and Taboun, S. M. 2001. A genetic algorithm for scheduling sequence dependent jobs on identical parallel machines to minimize the bi-criteria of makespan and number of tardy jobs. **The 29th International Conference Computers and Industrial Engineering**. Montreal, Canada. (2001, November 01): 508-513.

Narasimhan, S. L., and Mangiameli, P. M. 1987. A comparison of sequencing rules for a two-stage hybrid flow shop. **Decision Sciences** 18(2): 250-265.

Nawaz, M., Enscore, Jr, E. E., and Ham, I. 1983. A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem. **Omega** 11(1): 91-95.

Nearchou, A. C. 2004. The effect of various operators on the genetic search for large scheduling problems. **International Journal of Production Economics** 88(2): 191-203.

Negenman, E. G. 2001. Local search algorithms for the multiprocessor flow shop scheduling problem. **European Journal of Operational Research** 128 (1): 147-158.

Nemhauser, L. G., and Wolsey, L. A. 1999. Integer and combinatorial optimization. New York: John Wiley.

Néron, E., Baptiste, P., and Gupta, J. N. D. 2001. Solving hybrid flow shop problem using energetic reasoning and global operations. **Omega – The International Journal of Management Science** 29(6): 501-511.

Nowicki, E., and Smutnicki, C. 1998. The flow shop with parallel machines: A tabu search approach. **European Journal of Operational Research** 106(2-3): 226-253.

Oguz, C., Lin, B. M. T., and Cheng, T. C. E. 1997. Two-stage flowshop scheduling with a common second-stage machine. **Computers and Operations Research** 24(12): 1169-1174.

Palmer, D. S. 1965. Sequencing jobs through a multi-stage process in the minimum total time--A quick method of obtaining a near optimum. **Operational Research Quarterly** 16(1):101-107.

Panwalkar, S. S., Dudek, R. A., and Smith, M. L. 1973. Sequencing research and the industrial scheduling problem. In S. E. Elmaghraby (ed.), **Proceedings of Symposium on the Theory of Scheduling and its Applications**, pp. 29-38. New York: Springer-Verlag.

Park, B. J., Choi, H. R., and Kim, H. S. 2003. A hybrid genetic algorithm for the job shop scheduling problems. **Computers & Industrial Engineering** 45(4): 597-613.

Paul, R. J. 1979. A production scheduling problem in the glass-container industry. **Operations Research** 27(2): 290-302.

Pearl, J. 1984. **Heuristics: Intelligent Search Strategies for Computer Problem Solving**. 1st ed. Massachusetts: Addison-Wesley.

Pinedo, M. 1995. **Scheduling: theory, algorithms, and systems**. 1st ed. New Jersey: Prentice-Hall.

Pinedo, M., and Chao, X. 1999. **Operations scheduling with applications in manufacturing and services**. 1st ed. Boston: McGraw-Hill.

Portmann, M. C., Vignier, A., Dardilhac D., and Dezalay, D. 1998. Branch and bound crossed with GA to solve hybrid flowshops. **European Journal of Operational Research** 107(2): 389-400.

Quadt, D., and Kuhn, H. 2007. A taxonomy of flexible flow line scheduling procedures. **European Journal of Operational Research** 178(3): 686-698.

Rajendran, C., and Chaudhuri, D. 1992. A multi-stage parallel-processor flowshop problem with minimum flowtime. **European Journal of Operational Research** 57(1): 111-122.

Rajendran, C., and Ziegler, H. 2003. Scheduling to minimize the sum of weighted flowtime and weighted tardiness of jobs in a flowshop with sequence-dependent setup times. **European Journal of Operational Research**. 149(3): 513-522.

Riane, F., Artiba, A., and Elmaghraby, S. E. 1998. A hybrid three-stage flowshop problem: Efficient heuristics to minimize makespan. **European Journal of Operational Research** 109(2): 321-329.

Reeves, C. R. 1995. A genetic algorithm for flowshop sequencing. **Computers & Operations Research** 22(1): 5-13.

Ruiz, R., and Maroto, C. 2005. A comprehensive review and evaluation of permutation flowshop heuristics. **European Journal of Operational Research** 165(2): 479-494.

Ruiz, R., and Maroto, C. 2006. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. **European Journal of Operational Research** 169(3): 781-800.

Ruiz, R., Maroto, C., and Alcaraz, J. 2005. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. **European Journal of Operational Research** 165(1):34-54.

Ruiz, R., Şerifoğlu, F., and Urlings, T. 2008. Modeling realistic hybrid flexible flowshop scheduling problems. **Computers & Operations Research** 35(4): 1151-1175.

Salvador, M. S. 1973. A solution to a special case of flow shop scheduling problems, In S. E. Elmaghraby (ed.) **Symposium on the Theory of Scheduling and Applications,** Springer-Verlag, New York: 83-91.

Santos, D. L., Hunsucker, J. L., and Deal, D. E. 1995. Flowmult: permutation sequences for flow shops with multiple processors. **Journal of Information & Optimization Sciences** 16(2): 351-366.

Santos, D. L., Hunsucker, J. L., and Deal, D. E. 1996. An evaluation of sequencing heuristics in flow shops with multiple processors. **Computers & Industrial Engineering** 30(4): 681-691.

Sarin, S. C., Ahn, S., and Bishop, A. B. 1988. An improved branching scheme for the branch and bound procedure of scheduling n jobs on m machines to minimize total weighted flowtime. **International Journal of Production Research** 26(7): 1183-1191.

Sawik, T. 2000. Mixed integer programming for scheduling flexible flow lines with limited intermediate buffers. **Mathematical and Computer Modelling** 31(13): 39-52.

Sawik, T. 2002. An exact approach for batch scheduling in flexible flow lines with limited intermediate buffers. **Mathematical and Computer Modelling** 36(4-5): 461-471.

Sawik, T. 2007. A lexicographic approach to bi-objective scheduling of single-period orders in make-to-order manufacturing. **European Journal of Operational Research** 180(3): 1060-1075.

Schuurman, P., and Woeginger, G. J. 2000. A polynomial time approximation scheme for the two-stage multiprocessor flow shop problem. **Theoretical Computer Science** 237(1-2): 105-122.

Shen, V. Y., and Chen, Y. E. 1972. A scheduling strategy for the flow-shop problem in a system with two classes of processors. **Proceedings of the 6th Princeton Conference on Information and System Science** 645-649.

Smith, W. E. 1956. Various optimizers for single-stage production. **Naval Research Logistics Quarterly** 3(1): 59-66.

Soewandi, H., and Elmaghraby, S. E. 2001. Sequencing three-stage flexible flowshops with identical machines to minimize makespan. **IIE Transactions** 33(11): 985-993.

Soewandi, H., and Elmaghraby, S. E. 2003. Sequencing on two-stage hybrid flowshops with uniform machines to minimize makespan. **IIE Transactions** 35(5): 467-477.

Sriskandarajah, C., and Sethi, S. P. 1989. Scheduling algorithms for flexible flowshops: Worst case and average case performance. **European Journal of Operational Research** 43(2): 143-160.

Sule, D. R., and Vijayasundaram, K. 1998. A heuristic procedure for makespan minimization in job shops with multiple identical processors. **Computers & Industrial Engineering** 35(3-4): 399-402.

Sundararaghavan, P. S., Kunnathur, A. S., and Viswanathan, I. 1997. Minimizing makespan in parallel flowshops. **Journal of the Operational Research Society** 48(8): 834-842.

Tang, L., and Liu, G. 2007. A mathematical programming model and solution for scheduling production orders in Shanghai Baoshan Iron and Steel Complex. **European Journal of Operational Research** 182(3): 1453-1468.

Tsubone, H., Ohba, M., Takamuki, H., and Miyake, Y. 1993. A production scheduling system in the hybrid flow shop. **Omega**. 21(2): 205-214.

Uetake, T., Tsubone, H., and Ohba, M. 1995. A production scheduling system in a hybrid flow shop. **International Journal of Production Economics** 41(1-3): 395-398.

Valente, J. M. S., and Alves, R. A. F. S. 2005. An exact approach to early/tardy scheduling with release dates. **Computers & Operations Research** 32(11): 2905-2917.

Ventura, J. A., and Radhakrishnan, S. 2003. Single machine scheduling with symmetric earliness and tardiness penalties. **European Journal of Operational Research** 144(3): 598-612.

Wang, H. 2005. Flexible flow shop scheduling: Optimum, heuristics, and artificial intelligence solutions. **Expert Systems**. 22(2): 78-85.

Wang, L., Zheng, D.-Z. 2003. An effective hybrid heuristic for flow shop scheduling. **The International Journal of Advanced Manufacturing Technology** 21(1): 38-44.

Wang, W., and Hunsucker, J. L. 2003. An evaluation of the CDS heuristic in flow shops with multiple processors. **Journal of the Chinese Institute of Industrial Engineers** 20(3): 295-304.

Wardono, B., and Fathi, Y. 2004. A Tabu search algorithm for the multi-stage parallel machine problem with limited buffer capacities. **European Journal of Operational Research** 155(2): 380-401.

Werner, F. 1984. **On the solution of special sequencing problems.** PhD Thesis, TU Magdeburg.

Wilbrecht, J. K., and Prescott, W. B. 1969. The influence of setup time on job shop performance. **Management Science**. 16(4): B274-B280.

Wilkerson, L. J., and Irwin, J. D. 1971. An improved algorithm for scheduling independent tasks. **AIIE Transaction** 3(3): 239-245.

Winston, W. L. 2004. **Operations Research: Applications and Algorithms**. 4th ed. California: Thomson-Brooks.

Winston, W. L., and Venkataramanan, M. 2003. **Introduction to Mathematical Programming - Operations Research: Volume ONE**. 4th ed. California: Thomson-Brooks.

Wittrock, R. J. 1985. Scheduling algorithms for flexible flow lines. **IBM Journal of Research and Development** 29(4): 401-412.

Wittrock, R. J. 1988. An adaptable scheduling algorithm for flexible flow lines, **Operations Research** 36(3): 445-453.

Yanney, J. D., and Kuo, W. 1989. A practical approach to scheduling a multistage, multiprocessor flow-shop problem. **International Journal of Production Research** 27(10): 1733-1742.

Zandieh, M., Fatemi Ghomi, S. M. T., and Moattar Husseini, S. M. 2006. An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times. **Applied Mathematics and Computation** 180(1): 111-127.

VITA

Mr.Jitti Jungwattanakit was born on May 20th, 1975 in Bangkok province, Thailand. He graduated from King Mongkut’s Institute of Technology Ladkrabang, Thailand in academic year 1995 with a bachelor’s degree in Electronic Engineering. He started his work in 1996 as an electrical engineer at Siam Yamato Steel Co., Ltd. During his work, he also earned a Master’s degree in Industrial Management from King Mongkut’s Institute of Technology Ladkrabang, Thailand in academic year 2000. He decided to quit his job to pursue his doctorate in 2003 in the field of Industrial Engineering at Chulalongkorn University, Thailand.