Optimal Lot Streaming and Scheduling of Multiple Jobs in Two-Machine No-Wait Flow Shops

Ming Li

A Thesis

in

The Department

of

Mechanical and Industrial Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

March 2002

0-612-68456-3

ABSTRACT

Optimal Lot Streaming and Scheduling of Multiple Jobs in
Two-Machine No-Wait Flow Shops

Ming Li

In this thesis, a heuristic method is developed to optimize the sublots sizes of a number of jobs and to sequence these jobs simultaneously in two-machine no-wait flow shops. The objective is to minimize the production cost.

A no-wait manufacturing system is a production environment in which each job must be processed from start to finish, without any interruption either on or between machines. Such situations can be found in many manufacturing systems as in iron and steel production as well as in the process of anodizing products and components. In a manufacturing flow shop, there normally are several jobs with multiple identical items to be processed by a set of machines. Lot streaming is to create sublots so that machine operations can be overlapped. This thesis work devotes to the development of a heuristic method to find optimal discrete-sized sublots for each job and sequence multiple jobs efficiently in a two-machine no-wait flow shop.

Simulated annealing based heuristic search is used to search for a global optimal solution of the problem. The heuristic method is extended to solve three-machine flow shop problems. The efficiency and the effectiveness of this method are illustrated using several numerical examples.

ACKNOWLEDGMENTS

I would like to express my gratitude to my supervisor, Dr. Chen, for his continuous support and guidance during all stages of this thesis. I sincerely appreciate his valuable suggestions and thoughtful comments. His encouragement has enabled me to reach this education level. This thesis research has been supported by Natural Science and Engineering Research Council (NSERC) through Dr. Chen's grant and by Concordia Faculty of Engineering and Computer Science Graduate Support Fund.

I would like to thank my friends Shen Fan and Wen Qin for their continuous support and encouragement throughout this work.

I am profoundly thankful to my families, especially to my parents. Without their love and encouragement, this work would not have been possible. I also wish to thank my lovely daughter for the endless happiness she brought to me during this work.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

1.1.1 Production System and Production

A production system takes inputs and converts them into outputs. Here, inputs are raw materials, personnel, machines, buildings, technology, cash, information, and other resources; outputs are products and services. This conversion process is the heart of what is called production and is the predominant activity of a production system.

1.1.2 Different Types of Production System

There are many different viewpoints one may take in looking at a production system. Perhaps the most obvious characteristics of a production facility are the volume of items produced and the variety of different products made using the same resources. In this sense, according to Anderson (1994), production systems can be categorized into three types, i.e., mass production, batch production and one-off production.

Mass production involves producing a small number of different products in great quantity. An example might be the production of chocolate bars, where a dedicated production line will continue to produce the same product perhaps for years at a time. One characteristic of a mass production process is that operations are linked together in a line: when one operation is finished on a product it moves directly to the next operation.

Sometimes the product in question is a liquid or powder, for example, in the manufacture of a chemical fertilizer.

Batch production is used when there is a greater variety of products being produced, with correspondingly smaller volumes. In this situation it is usual to have machinery and equipment which can be used to carry out operations on a number of different products. A single machine will carry out an operation on a whole batch of items of one kind and then be set up to carry out a similar operation on a whole batch of items of another kind. An example of this type of production occurs in the manufacture of components for the aerospace industry. The number of items in a batch may depend partly on the expense associated with setting up machinery to process different parts, and partly on the size of individual customer orders.

One-off production is used when individual customers each requires an individual product, which is different from any product the company has made in the recent past. This implies low volumes but the greatest possible variety. With very large and complicated items the manufacturing process may be project based. This indicates that the manufacturing process is sufficiently complex, and over a long enough time-scale, that the major difficulties are associated with planning how the various different operations and activities will fit together. The most obvious examples of this type of manufacture occur in civil engineering projects.

Finally, it should be pointed out that large numbers of manufacturing facilities fail to fit neatly into any category. In reality, it can often be found that a manufacturing facility is a hybrid of more than one production style.

Since scheduling problems are often encountered in a batch production system, we assume that in this research batch production is the background of our study.

1.1.3 Production Planning and Scheduling

Now we know that production can be viewed as a transformation process in which raw materials are transformed into end products. However, these transformations are not possible without production resources and their planning.

Crucial to controlling production operations is the detailed scheduling of various aspects of the production function. We may view the production function in a company as a hierarchical process. First, the firm must forecast demand for aggregate sales over some predetermined planning horizon. These forecasts provide the input for determining the aggregate production and workforce levels for the planning horizon. The aggregate production plan then must be translated into the master production schedule (MPS). The MPS results in specific production goals by product and time period.

Materials requirements planning (MRP) is one method for meeting specific production goals of finished-goods inventory generated by the MPS. The MRP system "explodes" the production levels one obtains from the MPS analysis back in time to obtain production targets at each level of assembly by time period. The result of the MRP analysis is specific planned order releases for final products, subassemblies, and components.

Finally, the planned order releases must be translated into a set of tasks and the due dates associated with those tasks. This detailed planning results in the shop floor schedule. Because the MRP or other lot scheduling system usually recommends revisions in the planned order releases, shop floor schedules change frequently.

Shop floor control means scheduling personnel and equipment in a work center to meet the due dates for a collection of jobs. Often, jobs must be processed through the machines in the work center in a unique order or sequence.

Both jobs and machines are treated as indivisible. Jobs must wait, or queue up, for processing when machines are busy. This is referred to as discrete processing. Production scheduling in continuous-process industries, such as sugar or oil refining, has a very different character.

Although there are many problems associated with operations scheduling, our concern in this thesis will be job sequencing. Given a collection of jobs remaining to be processed on a collection of machines, the problem is how to sequence these jobs to optimize some specified criterion. Properly choosing the sequencing rule can affect dramatic improvements in the throughput of the job shop.

1.1.4 Typical Production Scheduling Objectives

One of the difficulties of scheduling is that many objectives are present. More unfortunately they are often conflicting. The goals of different parts of the firm are not always the same. Some of the most common objectives are

- Meet due dates.
- Minimize work-in-process (WIP) inventory.
- Minimize the average flow time through the system.
- Provide for high machine/worker time utilization. (Minimize machine/worker idle time.)
- Provide for accurate job status information.

- Reduce setup times.
- Minimize production and worker costs.

1.1.5 Job Shop Scheduling and Flow Shop Scheduling

Normally, in a shop floor, there are a number of machines. Each of them is able to finish a unique operation. It is a common scene that, in line with the last level production planning, there are a number of jobs that have to be processed on these machines. Limited by the capacity of each machine, these jobs cannot be processed on these machines at the same time. To achieve the goal as optimal or approximately optimal as possible, we have to schedule these jobs through these machines under some criteria. The procedure of making the decisions about scheduling jobs through machines is called shop floor control, sometimes is called job shop scheduling.

1.1.5.1 Some Important Characteristics of Job Shop Scheduling Problems

In Nahmias (2001), significant issues for determining optimal or approximately optimal scheduling rules are the following:

The job arrival pattern: We often view the job shop problem as a static one in which we take a "snapshot" of the system at a point in time and proceed to solve the problem based on the value of the current state. Although many of the solution algorithms we consider view the problems as being static, most practical shop scheduling problems are dynamic in nature.

Number and variety of machines in the shop: A particular job shop may have unique features that could make implementing a solution obtained from a scheduling algorithm difficult. For example, it is generally assumed that all machines of a given type are identical. This is not always the case, however. The throughput rate of a particular machine could depend upon a variety of factors, such as the condition of the machine or the skill of the operator. Depending on the layout of the shop and the nature of the jobs, constraints might exist that would make solutions obtained from an "all purpose" procedure infeasible.

Particular flow patterns: The solutions obtained from the scheduling algorithms require that jobs be completed in a fixed order. Each sequence of jobs through machines results in a pattern of flow of materials through the system.

Evaluation of alternative rules: The choice of objective will determine the suitability and effectiveness of a sequencing rule. It is common for more than one objective to be important, so that it may be impossible to determine a unique optimal rule.

1.1.5.2 Terminologies on Job Shop Scheduling

Flow shop. In a flow shop each of the *n* jobs must be processed through the *m* machines in the same order, and each job is processed exactly once on each machine. This is what we typically think of as an assembly line.

Job shop. A general job shop differs from a flow shop in that not all jobs are assumed to require exactly *m* operations, and some jobs may require multiple operations on a single machine. Furthermore, in a job shop each job may have a different required sequencing of operations. General job shop problems are extremely complex. All-purpose solution algorithms for solving general job shop problems do not exist.

Sequential processing and parallel processing. Most of the problems that we will consider involve sequential processing. This means that the *m* machines are distinguishable, and different operations are performed by different machines. In parallel processing we assume that the machines are identical, and any job can be processed on any machine.

Flow time. The flow time of job i is the time that elapses from the initiation of the first job on the first machine to the completion of job i. Equivalently, it is the amount of time job i spends in the system.

Makespan. The makespan is the flow time of the job that is completed last. It is also the time required to complete all n jobs. Minimizing the makespan is a common objective in multiple-machine sequencing problems.

Tardiness and lateness. Tardiness is the positive difference between the completion time and the due date of a job. A tardy job is one that is completed after its due date. Lateness refers to the difference between the job completion time and its due date, and differs from tardiness in that lateness can be either positive or negative. Minimizing the average tardiness and the maximum tardiness is also a common scheduling objective.

1.1.5.3 Scheduling Jobs on Single Machine

Assume that n jobs are to be processed through one machine. As there is only one machine, every schedule can be represented by a permutation of the integers 1, 2, . . ., n. There are exactly $n!$ different permutation schedules.

- **Some Specific Sequencing Rules**

SPT (shortest processing time). Jobs are sequenced in increasing order of their processing times. The job with the shortest processing time is first, the job with the next shortest processing time is second, and so on.

EDD (earliest due date). Jobs are sequenced in increasing order of their due dates. The job with the earliest due date is first, the job with the next earliest due date is second, and so on.

CR (critical ratio). Critical ratio scheduling requires forming the ratio of the processing time of the job, divided by the remaining time until the due date, and scheduling the job with the largest ratio next.

➢ **A Comparison Among These Specific Sequencing Rules**

For sequencing jobs on a single machine, it is shown that SPT optimized several objectives, including mean flow time. However, SPT sequencing could be problematic. Long jobs would be constantly pushed to the rear of the job queue and might never be processed. For that reason, pure SPT scheduling is rarely used in practice. CR attempts to balance the importance placed on processing time and the remaining time until the due date. However, there is little evidence to suggest that critical ratio scheduling performs well relative to the common optimization criteria such as mean flow time. As one would expect, EDD scheduling performs best when the goal is to minimize the maximum tardiness.

1.1.5.4 Scheduling Jobs on Multi-Stage Machines

Assume that n jobs are to be processed through m machines. The number of possible schedules is staggering, even for moderate values of both n and m. For each machine, there are $n!$ different orderings of the jobs. If the jobs may be processed on the machines in any order, it follows that there are a total of $(n!)^m$ possible schedules. Obviously, according to French (1982), it is a Non-deterministic Polynomial (NP) hard problem to find an optimal sequence for this situation.

After identifying the difference between the job shop and the flow shop from their definitions, we can understand that scheduling problems concerned with a job shop are much more difficult than those in a flow shop. To make a scheduling problem tractable, we take a two-machine flow shop as the production environment in this study.

1.2 Important Considerations in Manufacturing Flow Shop

1.2.1 Setup, Batching, and Lot Streaming

In some manufacturing systems significant setups are required to change production from one type of products to another. Normally, setups depend on the technological characteristics of the machine. A setup may for instance involve the preparation, cleaning or heating of a machine before production can start; after the setup, the machine is in a certain setup state and a certain product can be produced. Normally, setups incur setup times and setup costs. They consume the limited capacity of the machine without contributing to production. So, we should do the best of our ability to avoid or reduce setups.

One obvious approach is to omit a setup if two groups of same product are produced in one run, i.e., both groups of product are arranged in one production batch. Like this, a production strategy called batching is devised. This can explain, from an economical viewpoint, why many products are manufactured in batches.

However, people noticed that batching strategy also brought a shortcoming – extending the production cycle time. In multi-stage manufacturing systems, such as flow shops, open shops or job shops, if people insist on the batching policy at each stage, a production lot can be transferred to the next machine only when it is completed on the current machine. Apparently, a significant part of time of some items completed on a machine is wasted on waiting for other items to be processed on the same machine. Naturally, people devised another production strategy. In this strategy, some items (a sublot) in a production lot can be transferred to the next machine and processed, while other items from the same lot, but of a different sublot, are processed on the current machine. We refer to this process of allowing overlaps through the creation of sublots as lot streaming. The creation of sublots permits the overlapping of different operations on the same production lot and may therefore reduce throughput time.

1.2.2 Terminologies on Lot Streaming

Lot streaming. Lot streaming is the process of splitting a production lot that consists of many identical items into sublots, and then scheduling those sublots in overlapping fashion, in order to accelerate the progress of an order in production.

Discrete version and continuous version. According to Trietsch and Baker (1993), discrete version of lot streaming means that there are discrete numbers of units in each sublot, i.e., the value of each sublot size always is an integer; and continuous version of lot streaming means that sublot sizes can be fractional.

Variable sublots, consistent sublots and equal sublots. According to Trietsch and Baker (1993), variable sublots means that the sublot size between machines i and $(i+1)$ may differ from that between machines $(i+1)$ and $(i+2)$, i.e., sublot sizes may change between machines; and consistent sublots means sublot sizes keep same between machines. A special case of consistent sublots is that all sublot sizes are equal, i.e., equal sublots.

Preemption and nonpreemtion. In this context, preemption means the sublots of a job are mingled with those of other jobs when they are processed on machines; on the contrary, nonpreemtion means the sublots of a job are processed successively on machines without mingling with sublots of other jobs.

1.2.3 Johnson's Algorithm

Assume that n products must be processed through two machines and that each product must be processed in the order machine 1 then machine 2. Furthermore, assume that the optimization criterion is to minimize the makespan. The problem of scheduling on two machines turns out to have a relatively simple solution.

A very efficient algorithm for solving the two-machine problem was discovered by Johnson (1954). Following Johnson's notation, denote the machines by A and B. It is assumed that the products must be processed first on machine A and then on machine B. suppose that products are labeled i, for $1 \leq i \leq n$, and define

A_i = processing time of product i on machine A.

B_i = processing time of product i on machine B.

Johnson's result is that the following rule is optimal for determining an order in which to process the products on the two machines.

Rule: product i precedes product $i + 1$ if min (A_i, B_{i-1}) < min (A_{i+1}, B_i).

An easy way to implement this rule is as follows:

- List the values of A_i and B_i in two columns.
- Find the smallest remaining element in the tow columns. If it appears in column A, then schedule that product next. If it appears in column B, then schedule that product last.
- Cross off the products as they are scheduled. Stop when all products have been scheduled.

Now, let us see an illustrative example of exerting Johnson's algorithm. There are eight products to be scheduled on two machines. The processing time of each product is given in Table 1.1.

Table 1.1 The Illustrative Example of Exerting Johnson's Algorithm

Job	1	2	3	4	5	6	7	8
A	5	2	1	7	6	3	7	5
B	2	6	2	5	6	7	2	1

According to Johnson's algorithm, we obtain the optimal sequence:

3 – 2 – 6 – 5 – 4 – 7 – 1 – 8

The value of the makespan is 37. The Gantt chart for the optimal scheduling is pictured as Figure 1.1.

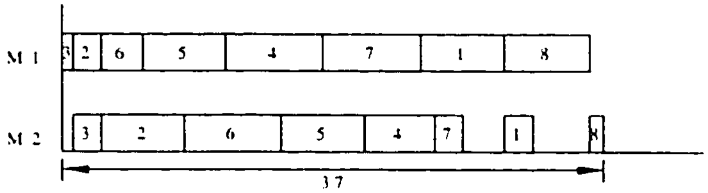

Figure 1.1 Gantt Chart By Johnson's Method

1.2.4 Interrelation Between Lot Streaming and Scheduling

It is well known that, when we deal with multiple single products, the length of the whole duration is concerned with not only the processing time of each of them but also the sequence of these products. For example, in a multi-product scheduling problem in a two-machine flow shop, any sequence which differs from the one obtained by using Johnson's method cannot be optimum. Similarly, when multiple single products are replaced by multiple lots (each lot comprises many identical items) and lot streaming is allowed for each lot, sequence still is a significant factor that affects the length of makespan. In addition, in this situation, any change on the size of sublots to any product may result in change on its ranking in a sequence. Thus, when the decomposition of the production lot is allowed, a solution procedure requires the creation of sublots through lot streaming, as well as the scheduling of sublots. Actually, in an environment of computer integrated manufacturing (CIM), lot streaming and scheduling decisions have to be taken concurrently, i.e., they are integrated and computer-controlled.

From the discussion above, it should be clear that lot streaming and scheduling decisions are strongly interrelated.

1.2.5 Inventories

When we consider inventories in the context of manufacturing, there is a natural classification scheme suggested by the value added from manufacturing or processing.

Raw materials. These are the resources required in the production or processing activity of the firm.

Components. Components correspond to items that have not yet reached completion in the production process. Components are sometimes referred to as subassemblies.

Work-in-process. Work-In-Process (WIP) is inventory either waiting in the system for processing or being processed. Work-in-process inventories include component inventories and may include some raw materials inventories as well. The level of WIP inventory is often used as a measure of the efficiency of a production scheduling system. The Just-In-Time (JIT) approach is aimed at reducing WIP to a minimum.

Finished goods. Also known as end items, these are the final products of the production process. During production, value is added to the inventory at each level of the manufacturing operation, culmination with finished goods.

1.2.6 Material Handling

In a manufacturing system, accompanying the procedure of transformation, all raw materials, components, and final products have to be moved from one place to another. This function is carried by a material handling system.

A material handling system is the entire network of transportation that receives materials, stores materials in inventories, moves them between processing points within and between buildings, and finally deposits the finished products into vehicles that will deliver them to customers. Popular material handling equipment includes: conveyors, cranes, elevators, forklifts, handcarts, and pallets. Automated guided vehicles (AGVs) can be found in FMS systems.

Since the subject in this study is job scheduling, we confine our attention on the material handling activities in the area of shop floor, i.e., the movement between machines.

1.2.7 Related Production Costs

- **Inventory Cost**

All inventory costs can be placed into one of three categories: holding cost, order cost, or penalty cost. Here, as a result of converging on WIP inventory, only holding cost is considered.

Holding cost. Holding costs are the costs that accrue as a result of having capital tied up in inventory. Holding costs can be assumed to be linear to the number of units being held for a particular time period.

- **Material Handling Cost**

In this context, material handling cost actually comprises two components: 1) material handler cost; 2) pallet and fixture cost.

Material handler cost. According to Langevin (1999), material handler costs are expenditures on operating and maintaining material handlers, e.g., AGVs, forklifts, etc. Material handlers are used to transfer parts among machines. Generally, material handler cost is a function of its running distance.

Pallet and fixture cost. According to Langevin (1999), this cost evaluates the expenses associated with the preparation and utilization of the pallets and fixtures. Normally, parts are positioned on pallets with fixtures for pick-up and drop-off. Similar to material handler cost, pallet and fixture cost is also a function of the distance traveled.

From now on, in this context, the total cost indicates the sum of holding cost and material handling cost.

1.2.8 No-Wait and Blocking Process

In Hall and Sriskandarajah (1996), the authors make a survey about scheduling problems with no-wait and blocking in process. A no-wait scheduling problem occurs in a production environment in which a job must be processed from start to finish, without any interruption either on or between machines. Blocking scheduling problems arise in serial manufacturing processes where no intermediate buffer storage is available. In such situations a job that has completed processing on a machine may remain there until a down-stream machine becomes available, but this prevents another job from being processed there.

The main reason for the occurrence of a no-wait or blocking production environment lies in the production technology itself. In some processes, for example, the temperature or other characteristics of the material require that each operation follow the previous one immediately.

Such situations arise in the production of steel, where molten steel undergoes a series of operations such as molding into ingots, unmolding, reheating, soaking, and preliminary rolling. Similarly, in the plastic molding production industries, a series of processes must follow one another immediately to prevent degradation. Further examples occur in the chemical and pharmaceutical industries. For similar reasons, the anodizing of metal products such as pipes, trim, and truck grilles has to be performed as a no-wait process. Modern manufacturing environments such as JIT, FMS, and robotic cells provide a highly coordinated manufacturing process which can frequently be modeled as a no-wait scheduling problem. Indeed, traditional scheduling models fail to address the need to meet demand requirements while simultaneously minimizing inventory. Finally, a no-wait environment may occur in service industries, where the customer has a prohibitively high cost of waiting in process.

The other reasons of a no-wait or blocking production environment is a lack of storage (or buffer) capacity between intermediate machines or work stations. In no-wait scheduling, a job must leave a machine immediately after processing is completed. The less restrictive case known as "blocking" permits a job to remain on a machine after processing if the next machine is busy, but no other job can be processed on that machine at that time. In-line robotic cells, in which robots are located along an in-line conveyor or other transport system, as used for spot-welding and assembling car bodies, may present a no-wait scheduling environment. Blocking may occur in JIT production lines maintaining a fixed limit on in-process inventory. A scheduling problem with blocking may be thought of as a relaxation of a similar scheduling problem with the no-wait assumption.

On the boundary of the no-wait/blocking scheduling environment are a number of interesting issues. For example, a flowshop problem with finite buffer capacity between operations is a very common scheduling environment. Much can be learned about this problem by studying the blocking environment, which is an extreme case of the finite capacity problem. As another example, the study of the blocking and no-wait environment has relevance to JIT systems and robotic cells. Furthermore, there are many industrial settings (for example, the steel industry) where the processing time of an operation is an increasing function of the wait because of material cooling and hardening. The no-wait scheduling environment is equivalent to a situation where this function increases very rapidly.

1.3 Different Approaches For Solving Flow Shop Scheduling Problems

Different methods have been developed and applied to solve various scheduling problems. Some of them are briefly discussed below.

- **Optimization**

In general, an ideal optimization approach means after a production system is described correctly and precisely by a mathematical model, operations research techniques are used and the optimality of the obtained solution is guaranteed.

However, in reality, this approach often encounters great difficulties. Those difficulties can be summarized below.

- It is difficult to formulate some manufacturing problems as optimization models.
- The solution to the optimization model, if it exists, may be difficult to obtain in a reasonable computation time, i.e., the time complexity of the problem is NP-hard.
- The solution to the model, if obtained, has to be interpreted before implementation. It may be difficult to use on-line.

➢ **Heuristics**

Heuristic techniques are efficient and practical methods that can be used to find good (but not necessarily optimal) solutions to a wide variety of difficult combinatorial problems. Such techniques are employed to find acceptable solutions to problems efficiently, while otherwise the optimal algorithms for finding optimal solutions take far too much computation time to be usable in practice.

➢ **Simulation**

Real-world scheduling problems are often too complex to be amenable to mathematical analysis. Computer-based simulation is a valuable tool for comparing various scheduling strategies and scenarios. A computer simulation is a computer program that accurately reflects a real-world situation. As with a mathematical model, variables are defined to represent real quantities and expressions developed to describe the relations among these variables. Although both deterministic and stochastic problems are amenable to simulation, simulation has been applied more in problems with some element of randomness.

1.4 Research In This Thesis

In the area of manufacturing management, it is important that to make production plans properly so as to keep production costs at a lower level and to deliver products in time. In this research, we study how to sequence multiple products in a two-machine no-wait flow shop in order to minimize the sum of holding cost and material handling cost.

1.4.1 Motivation

Along with the development of sophisticated manufacturing facilities and the progress of advanced production control approaches, people now take more care of holding cost and material handling cost than ever. It is well known that holding cost is a positive function of duration in which products have to be treated according to the production plan. Material handling cost is a positive function of the frequency of transferring products between machines. From the concept of batching and lot streaming, we can learn that two factors, duration and frequency, are conflicting in contributing to the production cost. In other words, to reduce holding cost would increase material handling cost; and vice versa.

In this research, we study how to partition a production lot into several sublots and sequencing all production lots simultaneously in order to minimize the sum of holding cost and material handling cost in a two-stage no-wait flow shop.

In manufacturing systems, a machine may represent a workstation, a line or a whole factory. If we consider a single facility, it is a single-machine case, otherwise it is a multi-machine case. A machine processes or produces different types of products. A production lot of each product is referred to as a job. Here, the word job has twofold implications. First, a job represents a type of product; second, a job is an aggregate of several pieces of identical units, called items. Setups are required in between these jobs.

1.4.2 Objectives and Main Contributions

For survival and development, manufacturers always do their best to keep production costs as low as possible. On the other hand, market demands for manufacturing quality, flexibility and fast product delivery are getting stronger. Job shop scheduling plays significant roles to meet this demand.

This thesis focuses on a static, deterministic, multi-job, two-machine flow shop where the objective is to minimize the total cost while satisfying the constraint of no-wait. While the attribute of this problem is NP-complete, a heuristic method was adopted to solve it. We focus our attention on minimizing the production cost as well as the makespan in a flowshop environment. In order to solve the NP-hard problem efficiently, a search method based on simulated annealing was chosen to search for an optimal result. Numerical examples were developed to test the effectiveness and efficiency of the developed method.

1.4.3 Thesis Organization

Chapter 2 presents an overview of literature on scheduling problems with lot streaming and lot streaming methods. Chapter 3 presents a formulation of the problem studied in this research and its solution procedure. Simulated Annealing (SA) algorithm is used to search for a global optimal solution of the problem. This is also presented in

Chapter 3. In Chapter 4, numerical examples are presented to illustrate the heuristic method. In addition we take experimental analysis and some discussions concerned with the problem. In Chapter 5, with some numerical experiments, we explore the effect of the heuristic method used in a three-machine no-wait flow shop. In Chapter 6, we present our conclusions and suggest directions for future work in this area.

Chapter 2

Literature Review

2.1 α/β/γ Descriptor

When we discuss issues of lot streaming, there always are two or more machines in the manufacturing system because sublots of jobs have to be transferred between machines. If there is only one machine, transfer is meaningless. However, the number of jobs in the manufacturing system may vary from one job to multiple jobs in different studies. In this situation, each job contains several or many identical items. Different criteria can be used to measure job schedules. Minimizing the maximum makespan and minimizing the total cost are most popular optimal criteria.

To make the description to each problem more abbreviated, we adopt a three-field descriptor α/β/γ to describe each situation as shown in Table 2.1. For example, $1/Fm/C_{max}$ stands for a problem in which there is a single job, which comprises many of the same units, to be processed in a multi-stage flow shop. The technique of lot streaming is allowable. The goal of the problem is to minimize the makespan.

2.2 Single-Job Lot Streaming

Apparently, it is relatively simple when there is only one job to process in a manufacturing system. In fact, many research papers on lot streaming begin their studies from discussing one-job lot streaming. Various analytical and numerical results from these studies are produced.

Table 2.1 α/β/γ Descriptor

α: number of jobs	1: single job
	N: multi-job
β: number of machines and its configuration	2: two-machine
	3: three-machine
	m: m-machine
	F: flow shop
	J: job shop
	O: open shop
γ: problem objective	C_{max}: minimizing makespan
	TC: minimizing total cost

Szendrovits (1975) discusses a problem of $1/Fm/C_{max}$. A single job is processed in a multi-stage flow shop with only one setup at each stage. The inventory system assumes a constant and continuous demand of products. Since the output rate from production is greater than the demand rate, equal-sized sublots are produced periodically which results in a finished product inventory. Moreover, during the manufacturing cycle time a WIP inventory must be carried until products are finished at the last operation. The author develops a functional relationship between job size, manufacturing cycle time and average WIP inventory. By minimizing total manufacturing cost, economic production quantity (EPQ) can be calculated. Here, the total cost is the sum of the fixed cost per job and the holding cost of the average finished product and WIP inventories. A cost sensitivity analysis reveals the large potential savings from using the suggested EPQ model instead of the conventional economic production lot quantity (ELQ) model in multi-stage production systems.

Potts and Baker (1989) study a problem of $1/Fm/C_{max}$. The study shows that, to obtain an optimal makespan value (i.e., the minimum makespan), it is sufficient to consider sublot sizes which are the same on the second machine as they are on the first machine and are the same on machine m as they are on machine m-1. On this basis, the following important conclusion is drawn: When there are only two or three machines in a flow shop, consistent sublots are sufficient to minimize the makespan. The solutions remain valid when there is more than one job. Actually, it becomes a benchmark for many succeeding studies.

In a two-machine system, efficient direct optimal computations can be derived when the number of sublots is predetermined. The computations can be used to calculate both sublot sizes and makespan. Further, improvement bounds are derived. The bounds might be useful in determining the number of sublots.

Baker and Pyke (1990) study the same problem, i.e., $1\ /Fm/C_{max}$. In this study, the job is divided into only two sublots. On one hand, more sublots is better than fewer sublots to reduce manufacturing cycle time; on the other hand, the benefits due to increased number of sublots show diminishing marginal returns. In fact, conclusions in Potts and Baker (1989) show that the two-sublots solution can achieve up to 50% of the potential benefit attainable from multiple sublots.

In Trietsch and Baker (1993), the problem of $1/Fm/C_{max}$ is studied in detail. The authors provide an overview of the basic models and solution algorithms for the lot streaming problem. They also provide both continuous and discrete versions of the problem formulation. For a discrete version of the problem, it is pointed out that the problem can be formulated as an integer programming model. However, it may be difficult to solve. In addition, it is pointed out that the optimal makespan in the continuous version serves as a lower bound for the optimal makespan in the discrete version. The makespan produced by rounding the continuous solution serves as an upper bound.

In the same paper, the authors also study *2/Fm/C_{max}* problems. The two-machine problem (i.e., m = 2) and its solution are clearly building blocks for larger lot streaming problems. They show how to solve the continuous and discrete cases of the two-machine problems. They then proceed to the three-machine problem (i.e., m = 3) with consistent sublots, where they show how linear programming formulations (for the continuous version) and integer programming formulations (for the discrete version) could be applied to find optimal solutions.

All studies surveyed as above do not consider material handling. However, in a real manufacturing system, material handling cost or time is often unavoidable.

On the basis of the model developed by Szendrovits (1975), Goyal (1976) further points out that manufacturing cycle time is a function of not only the job size but also the number of sublots. By adding fixed costs of moving sublots through all machines as a new cost component into the original total cost, the author obtains a modified EPQ function. Finally, the optimal sublot sizes can be found by an iterative search procedure.

Egbelu (1990) addresses a problem of *1/Fm/TC*. Cost factors considered in the model are labor/machining cost, tool cost, material handling cost, and overhead cost. Two main time consuming activities encountered in batch manufacturing are operations at the workstations and material handling. Both activities are influenced mostly by the machining parameters at each workstation and the material flow plan respectively. Based on this idea, the author developed a methodology to integrate the selection of machining rates and the number of parts to transport between workstations. This is to create an overall production plan to minimize production costs. This procedure can be adapted to describe any production system that uses stationary and mobile material handling system. However, the model only considers production systems for which there is only one unit of the material handling equipment between adjacent workstations in the product route. A dynamic programming algorithm embedded in a univariate search algorithm is used in seeking solutions to the model.

Egbelu (1991) studies a problem of $1/Fm/C_{max}$. Differing from those production systems discussed in above papers, a multi-stage manufacturing system discussed here is a flexible flow shop. There are multiple, not necessarily identical, machines at each stage of production in the system. In this paper, the machines in any given stage are assumed to be spatially separated; thus inter-machine handling time is considered explicitly for every pair of machines in adjacent stages. A model is proposed for planning and controlling production in a batch manufacturing system. Here, a single job exists as equal-sized sublots. Besides the time of moving sublots between machines, machine setup times and unit-load preparatory time on each machine are also considered. A heuristic algorithm based on dynamic programming is developed for seeking a solution to the model. Sensitivity analysis is performed. The conclusions point out that both larger sublot sizes and longer material handling time can cause longer makespan.

As an extension of the work in Szendrovits (1975), Steiner and Truscott (1993) study a situation of $1/Om/TC$. In this study, an open shop replaces the original flow shop. Since complete flexibility in operation sequencing is permitted, a study on the effect of sequencing decisions becomes necessary. In addition, a new measure of performance replaces the original one. Here, the total cost includes not only the WIP inventory carrying cost but also the sublots transportation cost.

2.3 Multi-Job Lot Streaming and Scheduling

On the basis of the studies concerning one-job lot streaming, researchers obtained many insights about lot streaming and extended studies from one job to multiple jobs. The latter situations can be found in many manufacturing systems.

It is well known that the sequence of multiple jobs to be processed can affect the makespan greatly. When lot streaming is allowed, the effect of sequencing still exists. However, the problem of sequencing in a lot streaming environment is more difficult than that in a classical production environment.

For $N/Fm/C_{max}$, a reasonable approach might be to sequence the jobs without lot streaming and then simply to split each job into optimal sublots. However, a counter-example in Potts and Baker (1989) shows that it is not possible to solve the N-job problem simply by the hierarchical scheme.

Differing from classical permutation scheduling problems, when lot streaming of each job is permitted, scheduling problems of N jobs through a two-machine or three-machine flow shop are studied in Vickson and Alfredsson (1992). The objective is to minimize the makespan. Under the assumption of 1) all transfer batches incur zero cost, 2) there are unlimited buffers between machines, 3) there are no setup times between different jobs on any of the machines, and 4) preemption of either items or sublots is not permitted, the optimality of unit-sized transfer batches (i.e., one item per transfer batch) is proven for general flow shop problems with regular performance measure. If preemption among all sublots is permitted, then some important results in classical scheduling theory hold as well. For instance, with consistent sublots, to obtain an optimal permutation schedule of *N* jobs, it is sufficient to only consider certain schedules. In these schedules, sublots order is same on the first pair and the last pair of machines. Under the same condition, an algorithm similar to Johnson's is derived to optimize two-machine problem and certain special three-machine problems. Examples are presented to show that, although all setup times are equal to zero, lot streaming can yield a better value of a regular performance measure than that without splitting jobs.

As we have seen from the example in Potts and Baker (1989), if preemption among sublots is permitted, there is a complex relationship between scheduling and lot streaming even when there are only two jobs. On the other hand, the lot streaming scheme of unit-sized sublots can rarely be found in real manufacturing systems. Two main reasons are: 1) when material handling cost is considered, this scheme will cause high material handling cost, 2) too many sublots are hardly being tracked and controlled. Hence, many studies have been conducted under the condition of non-preemptive and batch availability (i.e., any sublot is a batch rather than an item).

Vickson (1995) studies the problem of $N/F2/C_{max}$. This paper extends the optimal lot streaming analysis of Potts and Baker (1989) to the case of multiple jobs with job setup and sublot transfer times in a two-machine flow shop. It shows that lot streaming problem can be decoupled from job sequencing problem when performance measure is regular and jobs are not preempted. For this reason, the makespan minimization problem with lot streaming is reduced to an equivalent two-machine makespan problem without transfer batches or setup/transfer times. It then can be solved by Johnson's algorithm.

Closed-form solutions are given for the continuous lot sizing problem, and a fast algorithm is presented to solve its integer version. When setups are detached from the transfer lots, the linear programming solution is similar to that in Potts and Baker (1989) for the problem without setup times. When the setups are attached to the first transfer lot, a more complex situation is present.

The results reveal that a limited capacity to transfer parts between machines can be utilized effectively: significant benefits accrue even each unit of a job is not transferred when it becomes available.

Sriskandarajah and Hall (1996) review scheduling literature in no-wait and blocking production environment. They point out that no-wait and blocking scheduling problems provide an important direction of research. This importance is likely to increase because of the variety of no-wait and blocking scheduling applications arising in modern manufacturing. They discuss many applications in detail and present the first detailed summary of computational complexity of these problems. They also describe optimal and heuristic algorithms for solving certain deterministic flow shop, job shop, and open shop problem.

Langevin et al (1999) study how to optimize transfer batch size in order to minimize the total cost. The problem belongs to *2/F2/TC*. The two-machine FMS system studied in the paper is actually a blocking manufacturing system. Cost components include material handling cost, pallet cost, holding cost, and machine cost. Without considering preemption, the optimal sequence between jobs is determined by Johnson's algorithm with respect to maximize machine utilizations. Mathematical programs are developed to minimize total cost.

Sriskandarajah and Wagneur (1999) develope an algorithm to simultaneously deal with lot streaming and scheduling problem of multiple jobs in two-machine no-wait flow shops to minimize makespan. The problem is solved for a fixed number of sublots for each job. Closed-form solutions are given for the lot streaming problem in continuous version, then an efficient polynomial heuristic procedure is used to solve its discrete version. By exerting the algorithm of Gilmore and Gomory (1964), it is shown that the problem of sequencing and lot streaming of multiple products with continuous-sized sublots can be solved in O(NlogN) time. Computational results indicate that this heuristic consistently delivers close to optimal solutions. Afterwards, the solution procedures are extended to a traditional and more general lot streaming model.

2.4 Concluding Remarks

Lot streaming problems have been studied for several decades by many researchers. As we have seen, there are many studies exploiting the effectiveness of lot streaming to shorten manufacturing cycle time and cut down WIP inventory.

Lot streaming can reduce makespan effectively. In Trietsch and Baker (1993), an example of $1/F3/C_{max}$ reveals that, by splitting a job into two sublots and accepting the intermittent idling of machines, the value of makespan can be dropped to 360 minutes from the original 504 minutes. Saved time takes as much as 28.6% of the original flow time.

Lot streaming can also be implemented to cut down WIP inventory. The example in Szendrovits (1975) illustrates that a marginal saving of 25% in total cost can be obtained by using the EPQ model. In fact, the core of EPQ is lot streaming.

When we deal with multiple jobs while lot streaming is allowed, the issue of scheduling is unavoidable. As illustrated by a simple example in Potts and Baker (1989), preemption among jobs may bring in a complex circumstance. When the problem size is large, an optimal schedule will be difficult to generate. Hence, most studies assume non-preemption.

Many research papers consider manufacturing systems with unlimited intermediate buffers between machines. Studies on manufacturing systems without intermediate buffers are less. However, as shown in Hall and Sriskandarajah (1996), this type of manufacturing system exists in many industrial enterprises. Moreover, results from the former studies cannot be directly applied to the latter. For this reason, we focus our study on manufacturing systems with no-wait process, and use production cost as the performance measure.

2.5 Model and Solution Method Developed in This Thesis

In this thesis the problem of finding economic transfer batch size of each job as well as sequencing total N jobs simultaneously in a two-machine no-wait flow shop is studied. The goal is to minimize the production cost of manufacturing these N jobs. Here production cost is the sum of holding cost and material handling cost. It is a typical combinatorial optimization problem.

Most combinatorial optimization problems are NP-hard. For an NP-hard problem, computational effort rises exponentially with the problem size, or the number of variables in the problem. Although there exist optimal solutions, we may not be able to determine one in a reasonable computational time. Therefore, we encounter the computational difficulty in solving this problem.

To resolve NP-hard problems, many heuristic methods have been invented. A heuristic method is able to obtain satisfactory solutions in reasonable computational times but the obtained solutions are not guaranteed to be optimal. In this thesis, we use a heuristic method based on that in Sriskandarajah and Wagneur (1999). With this method, when a scheme of number of sublots assigned to each job is made, results like discrete-sized sublots, optimal sequence of jobs to be processed, and makespan can be obtained simultaneously. Having the initial scheme of number of sublots and the makespan corresponding to it, we can obtain an initial *TC*. With these initial results as starting point, we then use simulated annealing (SA) to search for the optimal scheme of number of sublots. The solution is to minimize production cost associated with lot streaming and job sequence. The efficiency and effectiveness of this method are investigated using several numerical examples presented in Chapter 4.

Chapter 3

Model Development and Solution Procedure

3.1 Problem Description

3.1.1 Problem Features

Assume that in a manufacturing flow shop, N different jobs are ready to be processed on two machines. Every job comprises many identical items, and each job has its own processing time on each machine. Since each job contains many identical items, lot streaming can be used to accelerate the machining process and to reduce work-in-process (WIP) inventory level. Also assume that each job should be processed on the machines with the same sequence, i.e., from first machine (M_1) to second machine (M_2) due to technical requirements. Furthermore, every job must be processed on both machines in no-wait style, i.e., when a sublot finishes its treatment on the first machine, it must be processed on the second machine immediately. There is no idle time between the two successive treatments.

The problem we intend to resolve is how to process these jobs to minimize the sum of material handling cost and inventory holding cost. It should be pointed out that although we do not consider the time of transferring items between the two machines, there do exist a sum of material handling cost for passing sublots from the first machine to the second one. Since there are N jobs, arranging the order of these jobs to be processed is necessary.

3.1.2 Assumptions

With the general problem description presented above, we give the following specific problem assumptions shown below.

1. All jobs are available at time zero. This assumption limits the analysis to the static no-wait flowshop for which the quantities of the products to be processed for a planning horizon are known in advance.
2. No machine can process more than one sublot of any job at a time.
3. The processing mode of sublots is no-wait. It means that each sublot of a job must be processed from start to finish, without any interruption either on or between machines. There is no buffer between the two machines in a no-wait environment.
4. No preemption is allowed. Preemption in this context means that sublots of a job are intermingled with those of other jobs. This assumption means that all sublots of a job are produced consecutively.
5. The processing of a sublot is proportional to its size, i.e., $p_{1,k_j} = p_{1j}k_j$ and $p_{2,k_j} = p_{2j}k_j$.
6. Setup times are independent of the sequence of job processing on the machines. An off-line setup can be performed on a machine as soon as this machine becomes available even without the physical presence of the arriving sublot of a job.
7. Unit holding cost is same for all items.
8. Unit material handling cost is same for all sublots.
9. Holding costs are directly proportional to holding time.

3.1.3 An Illustrative Example

Example 3.1

In a two-machine flow shop, there are four jobs J_1, J_2, J_3 and J_4. Each job comprises Q_j (j = 1, 2, 3, 4) items. These jobs are waiting to be processed by the two machines with no-wait in process. The setup time, s, processing time, p, and the number of items, Q, of each job are given in Table 3.1.

Table 3.1 The Given Data in Example 3.1

Job	s_1 (min.)	s_2 (min.)	p_1 (min.)	p_2 (min.)	Q (unit)
J_1	4	5	2	3	10
J_2	2	1	4	3	20
J_3	4	6	2	4	25
J_4	4	2	5	6	15

Our aim is to process these jobs with minimum production cost, TC, consisting of holding cost and material handling cost. Assume unit holding cost C_h = 0.05 and unit material handling cost C_{mh} = 5.00 in this example.

Consider the following two different cases. These two cases show that if the scheme of number of sublots assigned to each job or the sequence of jobs to be processed changes, then the production cost will vary as well.

Case 1: Same Processing Sequence but Different Number of Sublots

Assume that two schemes of number of sublots {5, 6, 8, 5} and {5, 6, 8, 6} are used in two production plans. The four numbers in each scheme are the number of sublots assigned to J_1, J_2, J_3 and J_4 respectively. We then can calculate two groups of sublot sizes for each job. If the same processing sequence J_3 - J_2 - J_1 - J_4 is used, the values of makespan and *TC* for each scheme can be calculated. The results are presented in Table 3.2. The difference due to the two schemes is clear.

Case 2: Same Sublot Number Arrangement but Different Processing Sequence

In this case, the scheme of number of sublots, n = {5, 6, 8, 5}, keeps unchanged. Sublot sizes are the same as those in Table 3.2. Two different processing sequences, J_3 - J_2 - J_1 - J_4 and J_3 - J_4 - J_1 - J_2, are used. Two different results in Table 3.3 reveal the difference due to the two different processing sequences.

In Tables 3.2 and 3.3, there are some notations. N_j is the number of sublots assigned to job j, Y_j are the sublot sizes of job j, and F is the makespan. The formal definitions of these notations are given in the next section.

Table 3.2 Two Groups of Y_j, F and TC Corresponding to Two Different Scheme of n_j (When Sequence Is: J_3 - J_2 - J_1 - J_4)

Job	J_1	J_2	J_3	J_4	Job	J_1	J_2	J_3	J_4
Scheme 1 of n_j	5	6	8	5	Scheme 2 of n_j	5	6	7	5
Y_j	1	6	1	2	Y_j	1	6	1	2
	1	5	1	2		1	5	1	2
	2	3	1	3		2	3	1	3
	2	3	1	4		2	3	1	4
	4	2	1	4		4	2	3	4
		1	2				1	6	
			6					12	
			12						
F	311min.				F	309min.			
TC	$1208.50				TC	$1196.50			

Table 3.3 Two Groups of F and TC Corresponding to Different Job Sequence

(When Scheme of n_j Is {5, 6, 8, 5})

Job Sequence	Sequence 1: $J_3 - J_2 - J_1 - J_4$	Sequence 2: $J_3 - J_4 - J_1 - J_2$
F	311min.	324min.
TC	\$1208.50	\$1254.00

From the above two tables, we can observe that both the change in the number of sublots and the change in the processing sequence will cause the change on makespan and TC values. Thus, it is necessary to decide sublot numbers for each job and, at the same time, to choose the processing sequence in order to minimize the total cost.

In the next section, we present a mathematical model to minimize the total cost of processing multiple jobs in a two-machine no-wait flow shop followed by a procedure to solve this model.

3.2 Model Development

We first present the notations used in the model.

3.2.1 Notations

m	machine index, $m = 1, 2$;
M_1, M_2	the two machines in the flow shop;
N	total number of jobs to be processed;
j	job index, $j = 1, ..., N$;
n_j	maximum allowable number of sublots for job j;

X_{kj} total number of items of job j in its kth sublot, where X_{kj} is a non-negative real number to allow formation of continuous-sized sublots, i.e., $X_{kj} \geq 0$;

Y_{kj} the total number of items of job j in its kth sublot, where Y_{kj} is a positive integer, i.e., $Y_{kj} \geq 1$;

$X(j)$ the vector of continuous-sized sublots for job j, i.e., $(X_{1j}, X_{2j}, \ldots, X_{nj})$. Index j will be dropped for the single job case;

$Y(j)$ the vector of integer-sized sublots for job j, i.e., $(Y_{1j}, Y_{2j}, \ldots, Y_{nj})$. Index j will be dropped for the single job case;

s_{1j}, s_{2j} setup times of job j on M_1 and M_2, respectively;

p_{1j}, p_{2j} processing times of one item in job j on machines M_1 and M_2 respectively, $p_{1j} > 0$, $p_{2j} > 0$;

$p_{1,kj}, p_{2,kj}$ processing times of sublot k of job j on machines M_1 and M_2, respectively;

$P_{\sigma(j)}$ the jth job in the sequence σ to be produced;

C, F the value of makespan of single-job problem and multi-job problem, respectively;

$C_{h.j}$ cost for holding one item in job j in a unit time period (such as, an hour, a week, etc.);

$C_{mh.j}$ cost of operating material handling equipment, pallets and fixtures to transfer one sublot of job j between machines each time;

HC holding cost;

MHC material handling cost;

TC total cost; the sum of holding cost and material handling cost;

Q_j number of items consisted in job j.

Decision variables:

μ_j desirable number of sublots for job j, $1 \leq \mu_j \leq n_j$;

σ the sequence of jobs to be processed.

3.2.2 The Cost Function

➢ **Cost Components**

- **Material Handling Cost (*MHC*)**

The material handling cost accounts for the fixed and variable operating costs to use the material handling equipment, such as, AGV/forklift, pallet and fixtures. When the route for transferring one sublot is determined, the material handling cost can be evaluated by the number of times of the material handling equipment to be used multiplying the unit material handling cost. This unit material handling cost can be estimated considering maintenance, energy, depreciation, and system operation.

If we transfer one sublot of job j with one pallet every time, the number of times of operating material handling equipment during processing job j equals to the total number of pallets used. Thus we can get the following expression of material handling cost for all jobs.

$$MHC = \sum_{j=1}^{N} n_j C_{mh,j}$$

- **Holding Cost (*HC*)**

The holding cost corresponds to the opportunity cost (lost interest) of the capital required for the work in progress and other intangible cost associated with WIP inventory. The unit holding cost $C_{h,j}$ of product j is calculated by multiplying the percentage of the inventory value per time period and the value of the item. In this study, we use makespan to measure the duration of the parts in the system.

The holding cost is calculated by:

$$HC = F \times \sum_{j=1}^{N} Q_j C_{h,j}$$

➢ **Cost Function**

In this study, the total cost is the sum of holding cost and material handling cost. Hence, we obtain the cost function as following:

$$TC = HC + MHC = F \times \sum_{j=1}^{N} Q_j C_{h,j} + \sum_{j=1}^{N} n_j C_{mh,j} \qquad (3.1)$$

Based on assumptions 7 and 8 discussed in Section 3.1.2, we can rewrite Equation (3.1) as following:

$$TC = F \times Q \times C_h + n \times C_{mh} \qquad (3.2)$$

where

$$Q = \sum_{j=1}^{N} Q_j \text{ and } n = \sum_{j=1}^{N} n_j .$$

3.2.3 Cost Function Analysis

To achieve the minimum makespan in order to reduce holding cost, one way is to transfer sublots more frequently. Vickson and Alfredsson (1992) point out that the minimum makespan can be obtained by using unit-sized sublots. However, this solution is based on the assumption of no transferring cost. When material handling cost is considered, the solution will lead to increased material handling cost due to increased number of times of part transfer. Obviously, intentions of minimizing holding cost and material handling cost are in conflict.

From Equation (3.2), it is clear that *TC* is a function of the makespan *F* and the total number of sublots *n*, i.e., *TC = TC (F, n)*. From above discussion, we know that to minimize the total cost, we have to make a suitable trade-off between the makespan and the number of sublots.

In the example at the beginning of this chapter, we noticed that the makespan changes while either the processing sequence of jobs or the sublot size of jobs changes. It is clear that makespan depends on both sublot size (decided by the number of sublots) and sequence of jobs to be processed on the machines. From Potts and Wassenhove (1992), we know that the makespan problem under lot streaming belongs to NP-hard problems.

3.3 Solution Procedure

We can infer from Equation (3.2) that once n is given in advance, $TC = TC\ (F,\ n)$ becomes $TC = TC\ (F)$. Since $F = F\ (n,\ \sigma)$, it is apparent that F now is only associated with sequence σ. The problem of how to minimize the total cost then is to sequence the jobs in order to minimize the makespan. Once we obtain the minimum makespan, we can calculate the total cost.

Similarly, we then can change the number of sublots to obtain another total cost. Comparing these two cost values, we can decide which scheme of sublots is better one. Iteratively like this, we can finally find the best combination of sublots for each job to obtain the minimum total cost. To search for the optimal total cost systematically, we implement simulated annealing (SA) in this study.

In a real manufacturing environment, pallets normally are limited resource. In other words, the number of pallets can be assigned as maximum n_j for job j before the job is processed. We need to decide the optimal number of pallets μ_j ($1 \leq \mu_j \leq n_j$) for job j. We now briefly describe our solution procedure.

First, with specification of each n_j, we need to calculate discrete-sized sublots Y_j of each job j and then to sequence all jobs. The goal is to minimize the makespan. With this makespan, we can calculate an initial total cost. Next, simulated annealing is used to search for the global optimal number of sublots, μ_j ($1 \leq \mu_j \leq n_j$), of job j. During each search, sublots size of each job is recalculated and the sequence of all jobs is reallocated. Finally, we can obtain the discrete-sized sublots of each job, the sequence of jobs to be processed, the value of makespan, and the minimized total cost.

3.3.1 Makespan: Model and Solution

To study the makespan problem of multiple jobs with discrete-sized sublots going through a two-machine flow shop with no-wait in process, we need to start from a single job with continuous-sized sublots. By studying the single-job problem, we can obtain more valuable insight about the multi-job problem.

We define sequence S as the ordered set of the sublot indices. Without loss of generality, assume the order of the sublot indices being $S = \{1, 2, \ldots, n\}$. Let π be the schedule obtained from S as shown in Figure 3.1.

Let Δ_i be the idle time on machine 2 (M_2) before processing sublot i and Γ_i be the idle time on machine 1 (M_1) after processing sublot i. Since we are considering a single job, the job index j is dropped at the moment.

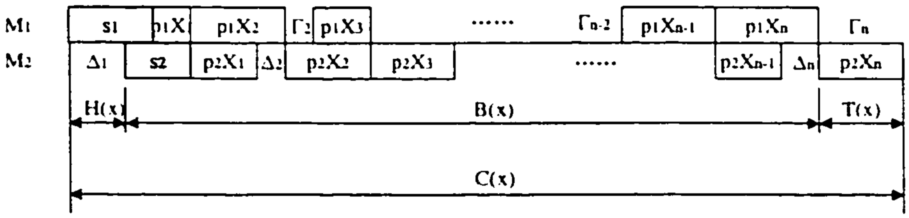

Figure 3.1 Schedule π of Sublots of a Single Job

The makespan *C(X)* can be written as:

$$C(X)=s_2+p_2Q+\max\{0,s_1-s_2+p_1X_1\}+\sum_{i=2}^{n}\max\{0,p_1X_i-p_2X_{i-1}\} \qquad (3.3)$$

or,

$$C(X)=s_1+p_1Q+\max\{0,s_2-s_1-p_1X_1\}+\sum_{i=2}^{n}\max\{0,p_2X_{i-1}-p_1X_i\}+p_2X_n \qquad (3.4)$$

In Equation (3.3), the makespan is given as a function of the idle time on M_2, while in Equation (3.4) it is given as a function of the idle time on M_1. We will first solve the minimum makespan problem by allowing continuous sublots and then find the optimal discrete sublots.

Let $Z=\sum_{i=1}^{n}\Delta_i$, where $\Delta_1=\max\{0,s_1-s_2+p_1X_1\}$, and $\Delta_i=\max\{0,p_1X_i-p_2X_{i-1}\}$, $i=2,\ldots,n$.

Note that minimization of the makespan *C(X)* is equivalent to the minimization of total idle time ($\sum_{i=1}^{n}\Delta_i$) on M_2.

The minimum makespan problem in Equation (3.3) can be solved by the following linear programming model.

$$\min Z = \sum_{i=1}^{n} \Delta_i \qquad (3.5)$$

subject to

$$\sum_{i=1}^{n} X_i = Q \qquad (3.6)$$

$$\Delta_1 \geq s_1 - s_2 + p_1 X_1$$

$$\Delta_i \geq p_1 X_i - p_2 X_{i-1}, \qquad i = 2, \ldots, n$$

$$X_i \geq 0, \ i = 1, \ldots, n$$

$$\Delta_i \geq 0, \ i = 1, \ldots, n$$

In Sriskandarajah and Wagneur (1999), an optimal solution $X^* = (X_1^*, X_2^*, \ldots, X_n^*)$ to Equations (3.5) and (3.6) is given by

$$X_i^* = p_1^{n-i} p_2^{i-1} \frac{Q}{\sum_{k=1}^{n} p_2^{n-k} p_1^{k-1}} \qquad i = 1, \ldots, n. \qquad (3.7)$$

Sriskandarajah and Wagneur (1999) also study N-job makespan problem in a two-machine no-wait manufacturing environment. We use their method (S & W method) to develop the solution procedure in our research. For this reason, we show the S & W method with an example.

Example 3.2

There are three jobs J_1, J_2 and J_3 to be processed on two machines in a flow shop with no-wait in process. Data are given in Table 3.4. The problem is to decide the sublot size for each job and the processing sequence of these jobs to minimize the makespan.

Table 3.4 The Given Data in Example 3.2

Job	s_1 (hour)	s_2 (hour)	p_1 (hour)	p_2 (hour)	Q (unit)	n
J_1	2	3	1	2	10	3
J_2	3	0	2	1	15	3
J_3	5	5	4	3	20	3

The procedure and the result of using S & W method to solve this problem are illustrated below.

- Calculate Continuous-Sized Sublots of Every Job

  With formula (3.7), continuous-sized sublots of each job can be calculated.

  $X_1^* = \{1.429, 2.857, 5.714\}$

  $X_2^* = \{8.571, 4.286, 2.143\}$

  $X_3^* = \{8.649, 6.486, 4.865\}$

- Calculate Discrete-Sized Sublots of Every Job

  With a heuristic approach developed in Sriskandarajah and Wagneur (1999), the continuous-sized sublots are transferred into the discrete-sized sublots as follows.

  $Y_1 = \{1, 3, 6\}$

  $Y_2 = \{9, 4, 2\}$

  $Y_3 = \{9, 6, 5\}$

- Calculate the H, B and T (see figure 3.1) of Every Job

  $H_1 = 0$, $B_1 = 12$, $T_1 = 12$

  $H_2 = 21$, $B_2 = 13$, $T_2 = 2$

  $H_3 = 36$, $B_3 = 52$, $T_3 = 15$

- Sequencing Jobs And Calculating The Makespan

With the algorithm in Gilmore and Gomory (1964) (G & G method), the optimal sequence of these jobs can be obtained according to the values of Head and Tail of each job. Then, the optimum makespan can be calculated.

Sequence σ: J_1 - J_3 - J_2

Makespan F: 136 (hours)

The schedule of the three jobs is shown in Figure 3.2.

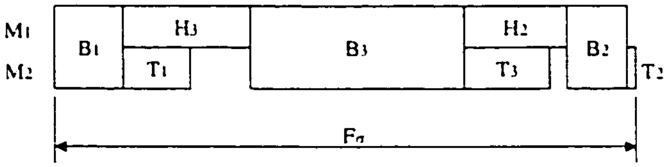

Figure 3.2 The Schedule σ of Example 3.2

3.3.2 Minimizing Total Cost

From Equation (3.2), we can calculate the total cost TC because makespan F has been obtained. However, this initial TC may not be the best one. We can replace n_j with another number of sublots $n_j^{'}$ to compute another TC'. Comparing TC' with the initial TC, we can make a decision to keep n_j or $n_j^{'}$. To minimize total cost, we can do this iteratively to finally find the best number of sublots for each job, μ_j $(1 \leq \mu_j \leq n_j)$.

To find μ_j, one obvious way is to solve our model repeatedly of every possible sublot number between 1 and n_j for job j. This means we have to repeat calculation up to $\prod_{j=1}^{N} n_j$ times, which is very inefficient for large size problems. For this reason, we search for μ_j using simulated annealing (SA).

3.4 Simulated Annealing Search for Optimal Solutions

3.4.1 Introduction

Simulated Annealing (SA) is a numerical optimization technique based on the principles of thermodynamics. Physical annealing refers to the process in which a solid is heated until all particles randomly arrange themselves in the liquid state, followed by a slow cooling, spending a relatively long time near the freezing point. The particles of the material attempt to arrange themselves in a low energy state during the cooling process. The energy states of the collection of particles can be considered the configuration of the solid. The probability that a particle is at any energy level can be calculated using Boltzmann distribution. As the temperature of the material decreases, the Boltzmann distribution tends toward the particle configuration that has the lowest energy.

Metropolis *et al.* (1953) first realized that the thermal equilibrium process could be simulated for a fixed temperature by Monte Carlo methods to generate sequences of energy states. During this process, the system is perturbed to yield a new configuration. The energy level before perturbation (E_j) and the energy level after perturbation (E_k) are compared. If E_j is greater than E_k (i.e., $\Delta E < 0$), the new perturbed system is accepted as the new configuration of particles. If $\Delta E > 0$, the probability of accepting the perturbed system is calculated in following equation.

$$p = exp(-\Delta E/kT) \qquad (3.8)$$

Where k is the Boltzmann constant and T is a fixed temperature. Using this criterion, the material will eventually reach its equilibrium configuration.

This basic concept can be applied to numerical optimization problems.

3.4.2 The Procedure of SA

Simulated Annealing applies Equation (3.8) to a series of variable configurations for the system to be optimized. The new variable settings are obtained by perturbing the current configuration. For numerical optimization, the objective function (*C*) replaces the energy term (*E*). The concept of temperature is retained and used as an important control factor. The probability of accepting an unfavorable step (i.e., $\Delta C > 0$, assuming minimization) is governed by Equation (3.9).

$$p = exp(-\Delta C /T) \qquad (3.9)$$

A random number, *P*, is drawn from a uniform random distribution on the interval [0,1]. If $P \geq p$, the unfavorable step is rejected and a new step is taken from the current position. If $P < p$, the unfavorable step is accepted and the new configuration replaces the old one. A new step is then taken relative to this configuration. This criterion allows the possibility of the current configuration to be replaced by an inferior configuration. This feature allows the algorithm to escape from local optima. New steps like this are taken iteratively until a termination criterion is reached.

3.4.3 Features of SA

The primary advantage of SA is the ability to move from local optima. Thus, the ability to find the global optimum is not related to the initial conditions (i.e., the starting point). In addition, the implementation process of SA is easy and fast. The primary disadvantages to SA are the subjective nature of choosing the SA configuration parameters (e.g., *T* and step size) and that it typically requires more objective function evaluations than other optimization approaches.

SA is a generic optimization technique of wide applicability. It has found wide application in many fields. For example, digital image processing, VLSI design, location of missile units, and heuristic approaches in Operations Research. Kirkpatrick *et al.* (1983) and Cerny (1985) independently pointed out the relevance of simulated annealing technique to combinatorial optimization problems. In this research, our goal is to find μ_j ($1 \leq \mu_j \leq n_j$) and finally to minimize *TC*. n_j is given before searching procedure. This means that n_j is finite if there are finite jobs to be processed.

3.4.4 The Implementation of SA in This Thesis

To solve a particular combinatorial optimization problem by the SA algorithm, a number of decisions have to be made. They include:

- Initial Temperature (T_i);
- Final temperature (T_f);
- Temperature function ($T(t)$); Where, t represents the number of times the temperature parameter has changed.
- Number of iterations (N_r);
- Neighbor generation;
- Stop criterion.

The initial temperature T_i should be sufficiently high to make search moves away from local optima; and, the final temperature T_f should be sufficiently low to ensure that the system is “frozen”. If T_i is too high, we might waste time trying to abolish the results of a long search process accepted because of the high starting temperature. If T_f is too low, we might waste time trying to get out of a local minimum at the end of the search, which may not be possible.

In the SA algorithm, temperature decreases according to the temperature function. At each value of temperature $T(t)$ ($T(t)$ is between T_f and T_i), search is taken N_r times iteratively. The more iterations the better the results, and obviously, the longer search time. Therefore, an appropriate trade-off between solution quality and search time should be made. A stopping criterion needs to be determined.

The search neighbor is crucial to the final results. In our study, we try to minimize *TC* by deciding the best scheme of sublots size of each job and then sequencing all jobs optimally. From the preceding discussion, we know that once the sublot size of each job is determined and assigned to the job, the optimal sequence of these jobs can be obtained by G & G method in a polynomial times. Therefore, different scheme of number of sublots assigned to the jobs will result in different *TC*. By searching for the best combination of the number of sublots assigned to every job, we can find the minimum *TC*. Since decrease by even one from the number of sublots assigned to any job would change the *TC*, the search neighbor is set by one sublot.

The implementation of the SA and its pseudo-code is presented as below.

Step 0: Initialization of SA:

Initial temperature: $T_i = H$ (H is a constant)

Final temperature: $T_f = L$ (L is a constant)

Temperature Function: $T(t) = T(t\text{-}1) - C$ (C is a constant, $T_f < T(t) \leq T_i$)

Number of Repetitions: $N_r = R$ (R is a constant)

Neighbor generation: one sublot

Stopping Criterion: $T(t) \leq T_f$

Constants H, L, C, and R vary in different problems.

Step 1: Input $S_0 = \{n_1, n_2, \ldots, n_N\}$ and the initial value of TC_0;

Set S_0 as the starting point, so $S_1 = S_0$, $TC_1 = TC_0$.

Step 2: Let $T(t) = T_i$;

$T(t)$ is reduced according to the temperature function. That means $T(t)$ is decreased by C each time.

Step 3: Let $N_r = 1$;

Each time, N_r is increased by one. When $N_r > R$, go to step 2.

Step 4: Generate a new state $S_2 = \{n_1', n_2', \ldots, n_N'\}$ as the neighbor of S_1;

Here, $\sum_{j=1}^{N} n_j' = \sum_{j=1}^{N} n_j - 1$. So, $S_1 \neq S_2$;

Calculate $\Delta TC = TC_2 - TC_1$ in moving from S_1 to S_2;

Calculate the acceptance probability: $p = e^{-\frac{\Delta TC}{T(t)}}$.

Step 5: If $\Delta TC \leq 0$, accept the new state S_2. Let $S_1 = S_2$. Then, go to step 3;

If $p > P$ (P is a random number, $0 < P \leq 1$), accept the new state S_2. Let $S_1 = S_2$. Then, go to step 3; otherwise, keep the former state as the current state, i.e., set $S_1 = S_1$. Then, go to step 3.

Step 6: If $T(t) < T_f$, stop.

3.5 The Application Program

In this research, to solve the problem of minimizing the TC in a two-machine no-wait flow shop efficiently, we develop an application program in C language. The program code is attached in the Appendix.

The program comprises two parts. The first part is designed to achieve the function of the S & W method. Given all known data of every job in advance, this function module can supply initial results of sublots size of every job, processing sequence, and makespan. The second part is designed to accomplish the function of the SA algorithm. Given the makespan and the initial number of sublots, this function module calculates the initial *TC*. Then, with this initial *TC* as the starting point, the SA algorithm is implemented to search for the optimal *TC* and associated sublots size. The flow diagram of the application program is showed in Figure 3.3.

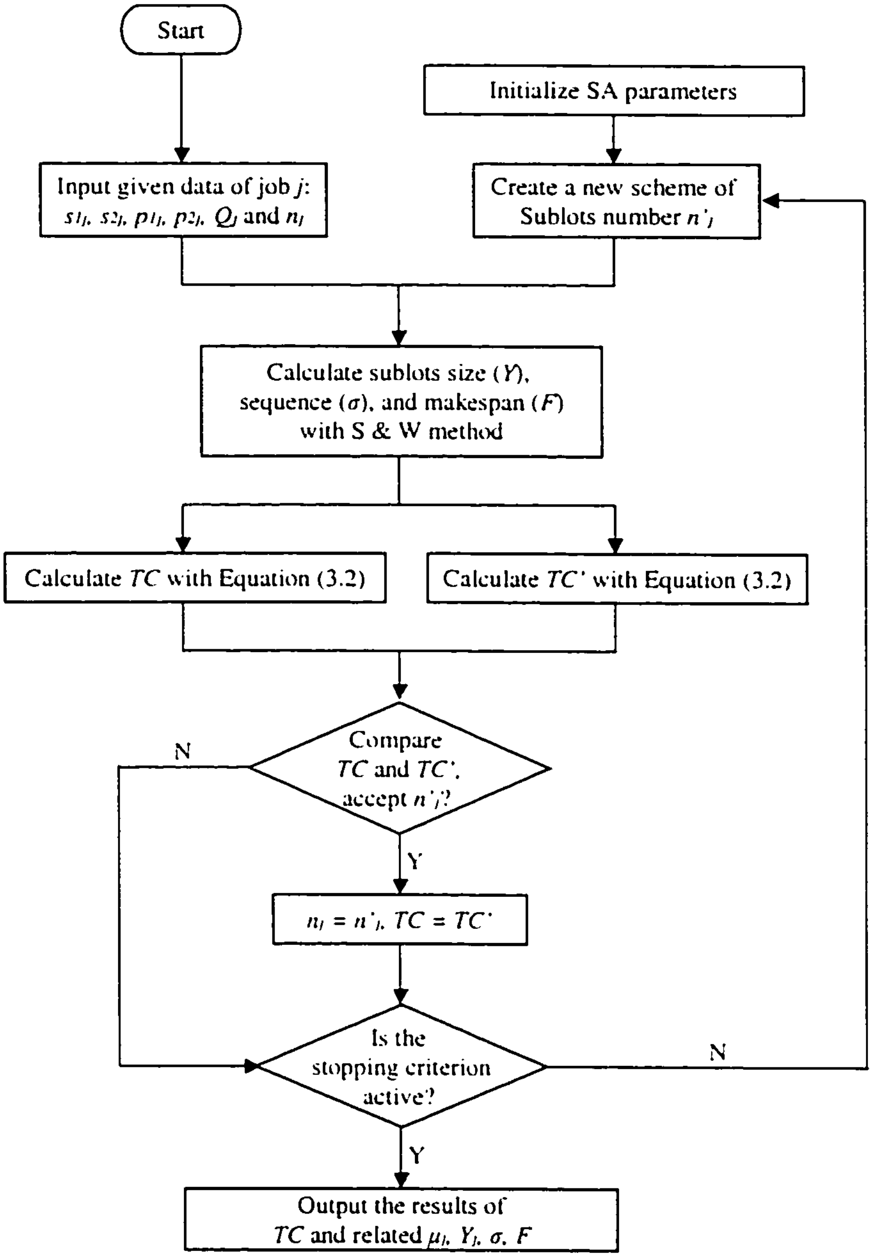
Start
Initialize SA parameters
Input given data of job j: s_{1j}, s_{2j}, p_{1j}, p_{2j}, Q_j and n_j
Create a new scheme of Sublots number n'_j
Calculate sublots size (Y), sequence (σ), and makespan (F) with S & W method
Calculate TC with Equation (3.2)
Calculate TC' with Equation (3.2)
Compare TC and TC', accept n'_j?
N
Y
$n_j = n'_j$, $TC = TC'$
Is the stopping criterion active?
N
Y
Output the results of TC and related μ_j, Y_j, σ, F

Figure 3.3 Flow Chart of the Program

Chapter 4

Numerical Examples and Computational Aspects

4.1 Numerical Examples

Example 4.1

In this example, the flow shop and the jobs to be processed are the same as in Example 3.2. However, the objective of this example is to decide lot streaming and processing sequence to minimize *TC*. Here, C_h = 0.04 ($ per hour per piece) and C_{mh} = 11.00 ($ per transference).

Solution Procedure

- Calculate Initial *TC*

In Example 3.2, we have obtained the makespan F = 136 hours. It corresponds to the given number of sublots n_j = 9. We now calculate *TC* using Equation (3.2).

$$TC = 136 \times 45 \times 0.04 + 9 \times 11.00 = 343.80\ \$$$

- Searching for μ_j With SA

With n_j = {3, 3, 3} as the starting point, we begin to search for μ_j. Our final goal is to find the optimal or near-optimal *TC*. In each searching step, we arbitrarily create a new n_j by decreasing the number of sublots of any job by 1. For example, we can first create n_j = {2, 3, 3}. We use S & W method to obtain the makespan corresponding to this new n_j, and calculate the *TC*. SA method is then used. We can accept {2, 3, 3} as the current state of the system or keep {3, 3, 3} as the current state of the system by comparing the old and new TCs. This search process continues until the stopping criterion of the SA method is reached.

The reduction of *TC* values for a number of SA search iterations is shown in Figure 4.1 and Figure 4.2. The SA parameters set in this example are: Initial Temperature $T_i = 300$, Final Temperature $T_f = 0$, Number of Iteration $N_r = 20$, and Cooling Speed $C = 1$.

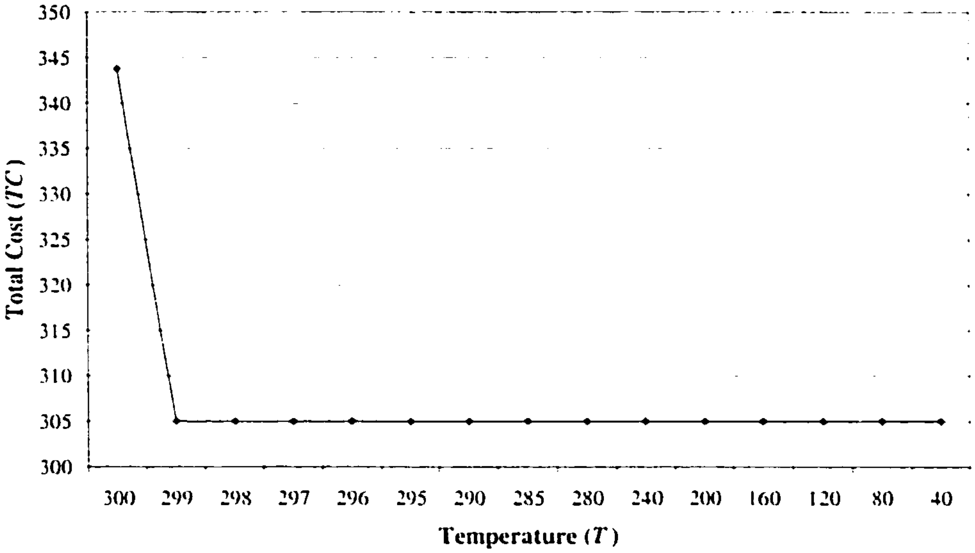

Figure 4.1 *TC-T* Relationship in Searching Procedure

The final results of the number of sublots μ_j and the sublot size Y_j for each job j in this example are presented in Table 4.1. In addition, we obtain the optimal sequence σ: $J_1 - J_3 - J_2$, the makespan $F = 145$ hours, and the minimum total cost $TC = 305.00\$$ concurrently.

In this example, the objective function value is reduced from 343.8 to 305 after 6000 times of iterations.

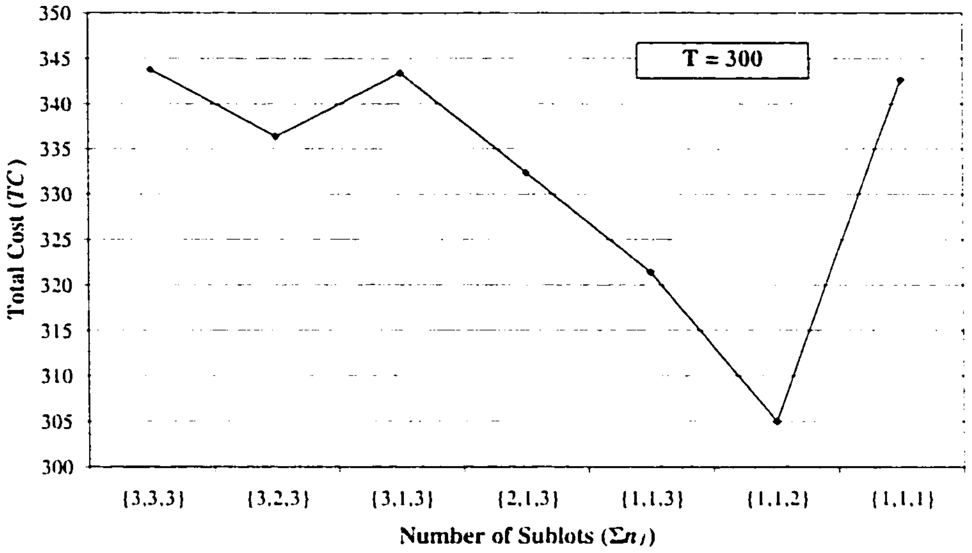

Figure 4.2 *TC*-Σn_j **Relationship When** T **= 300**

Table 4.1 The Final Results of Example 4.1

Job	J_1	J_2	J_3
μ_j	1	1	2
Y_j	10	15	11
			9

- To Verify the Optimality of the Results

Since there are only three jobs and total nine sublots in this example problem, it is possible to verify the effectiveness of our method by taking an exhausted search. The results of the exhausted search are listed in Table 4.2.

Comparing the final *TC* obtained with SA algorithm with those in Table 4.2, we can see that the results corresponding to *TC* = 305 are the optimal solutions.

Table 4.2 *TC*s of Exhaustive Search of Example 4.1

n_j	*TC ($)*	n_j	*TC ($)*	n_j	*TC ($)*
{3,3,3}	343.80	{2,3,3}	332.80	{1,3,3}	321.80
{3,3,2}	338.20	{2,3,2}	327.20	{1,3,2}	316.20
{3,3,1}	386.60	{2,3,1}	375.60	{1,3,1}	364.60
{3,2,3}	336.40	{2,2,3}	325.40	{1,2,3}	314.40
{3,2,2}	327.20	{2,2,2}	316.20	{1,2,2}	305.20
{3,2,1}	375.60	{2,2,1}	364.60	{1,2,1}	353.60
{3,1,3}	343.40	{2,1,3}	332.40	{1,1,3}	321.40
{3,1,2}	327.00	{2,1,2}	316.00	{1,1,2}	**305.00**
{3,1,1}	364.60	{2,1,1}	353.60	{1,1,1}	342.60

Example 4.2

In this example, we consider 20 jobs to be processed by two machines in a flow shop with no-wait in process. All data s_{1j}, s_{2j}, p_{1j}, p_{2j}, Q_j, and n_j for each job j are given in Table 4.3. The objective is to minimize *TC*. In this example, $C_h = 0.10$ and $C_{mh} = 8.00$ are used.

In this problem, all data are created randomly. The ranges of the data are: $0 \le s_{1j}$, $s_{2j} < 10$; $0 < p_{1j}, p_{2j} \le 10$; $10 \le Q_j \le 110$; and $3 \le n_j \le 7$.

Solution Procedure

A sequence σ: 1 – 19 – 17 – 4 – 9 – 2 – 13 – 12 – 18 – 16 – 6 – 14 – 8 – 11 – 15 – 7 – 5 – 3 – 10 – 20 and a makespan F = 7528 hours is obtained by S & W method. The σ and F correspond to the given number of sublots n_j. At the same time, discrete-sized sublots Y_j are obtained. They are shown in Table 4.4.

Having F and n_j, we can calculate the initial *TC* by Equation (3.2). That is

$$TC = 7528 \times 1194 \times 0.10 + 99 \times 8.00 = 899{,}635.20\,(\$)$$

Table 4.3 Input Data for Example 4.2

Job	s_1 (hour)	s_2 (hour)	p_1 (hour)	p_2 (hour)	Q (unit)	n
J_1	6	8	1	10	103	6
J_2	8	2	6	2	71	4
J_3	8	8	7	2	51	6
J_4	5	7	7	9	12	4
J_5	3	0	3	8	17	7
J_6	3	6	4	4	103	4
J_7	3	1	5	3	49	7
J_8	3	3	9	7	24	5
J_9	1	1	3	5	33	3
J_{10}	6	6	1	7	49	6
J_{11}	1	5	5	7	26	4
J_{12}	9	1	10	7	102	6
J_{13}	9	3	2	4	95	6
J_{14}	7	3	7	4	81	3
J_{15}	6	1	10	8	58	5
J_{16}	2	6	7	6	103	7
J_{17}	9	1	1	2	17	3
J_{18}	4	5	8	8	43	6
J_{19}	6	8	9	2	96	4
J_{20}	3	7	9	1	61	3

Table 4.4 Initial Results of Y_j Corresponding to n_j in Example 4.2

Job	n_j	Y_j
J_1	6	1, 1, 1, 1, 8, 91
J_2	4	48, 16, 5, 2
J_3	6	36, 10, 2, 1, 1, 1
J_4	4	2, 3, 3, 4
J_5	7	1, 1, 1, 1, 1, 3, 9
J_6	4	25, 26, 26, 26
J_7	7	20, 12, 7, 4, 3, 2, 1
J_8	5	7, 6, 5, 3, 3
J_9	3	6, 10, 17
J_{10}	6	1, 1, 1, 1, 4, 41
J_{11}	4	4, 5, 7, 10
J_{12}	6	35, 24, 17, 12, 8, 6
J_{13}	6	2, 3, 6, 12, 24, 48
J_{14}	3	43, 24, 14
J_{15}	5	17, 14, 11, 9, 7
J_{16}	7	22, 19, 17, 14, 12, 10, 9
J_{17}	3	2, 5, 10
J_{18}	6	7, 7, 7, 7, 7, 8
J_{19}	4	75, 16, 4, 1
J_{20}	3	54, 6, 1

Following the initial results, SA method is used to search for the final results. The SA parameters set in this example are: Initial Temperature T_i = 1000, Final Temperature T_f = 100, Number of Iterations N_r = 50, and Cooling Speed C = 1. Reduction of TC with SA method is shown in Figures 4.3 to 4.5. In this example, the TC value is decreased from 899,635.20$ to 852,397.80$ after 45000 times of iteration. Corresponding to TC = 852,397.80$, the sequence σ: 17 – 5 – 13 – 2 – 7 – 18 – 9 – 8 – 14 – 4 – 11 – 3 – 1 – 12 – 16 – 19 – 10 – 6 – 15 –20 and the makespan F = 7137 hours are obtained. The final results of μ_j and Y_j are given in Table 4.5.

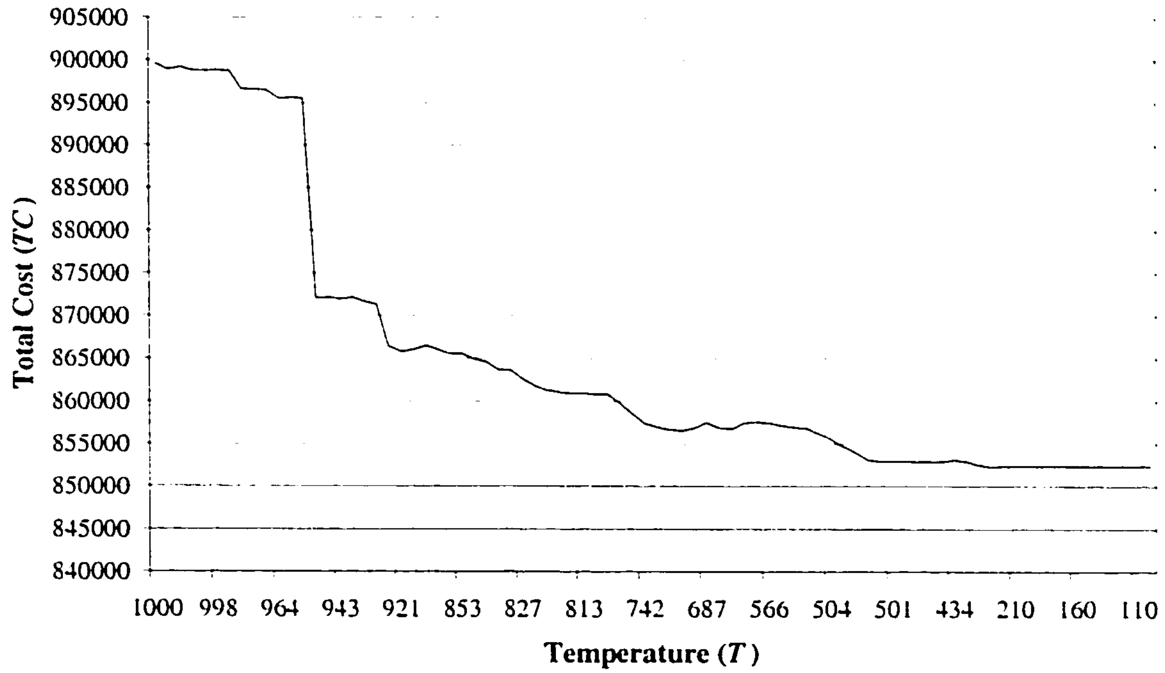

Figure 4.3 *TC-T* Relationship During Searching Procedure (Example 4.2)

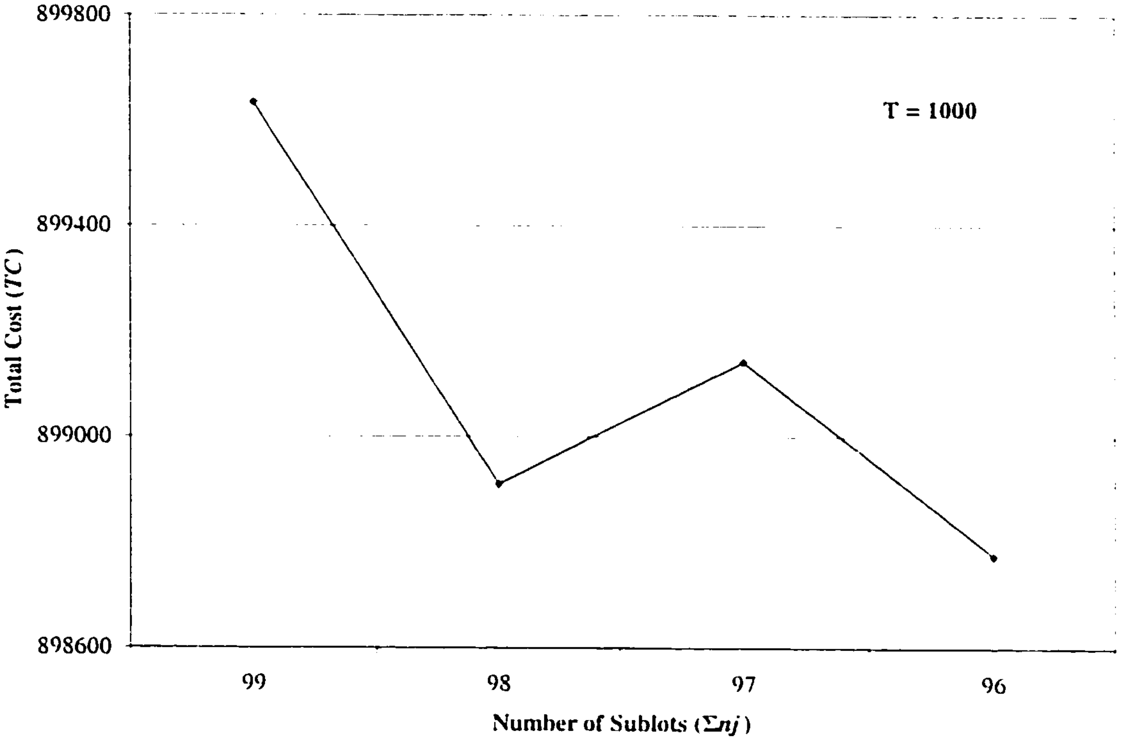

Figure 4.4 *TC*-Σn_j Relationship When *T* = 1000 (Example 4.2)

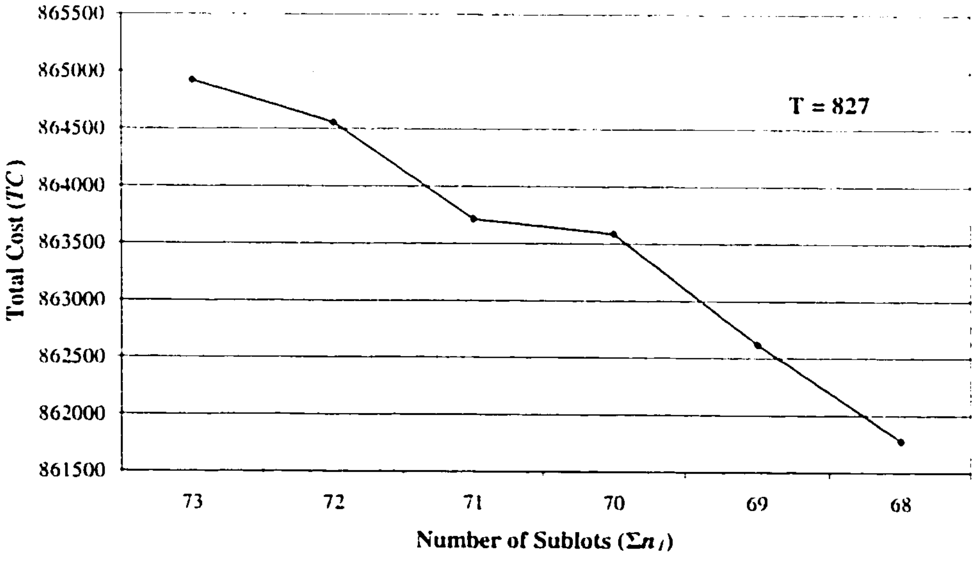

Figure 4.5 *TC*-Σn_j Relationship When *T* = 827

Table 4.5 Final Results of μ_j and Y_j in Example 4.2

Job	μ_j	Y_j
J_1	1	103
J_2	1	71
J_3	1	51
J_4	1	12
J_5	1	17
J_6	1	103
J_7	2	31, 18
J_8	1	14
J_9	1	33
J_{10}	1	49
J_{11}	1	26
J_{12}	1	102
J_{13}	1	95
J_{14}	3	43, 24, 14
J_{15}	1	58
J_{16}	1	103
J_{17}	1	17
J_{18}	4	10, 11, 11, 11
J_{19}	3	75, 17, 4
J_{20}	3	54, 6, 1

4.2 Computational Experiments

Simulated Annealing based search procedure requires a number of parameters such as the range of temperature, the cooling speed, and the number of iterations. The appropriate values of these parameters can be determined through a number of experiments for a specific problem. In this research, several such experiments were performed to investigate and to find the best values of these parameters.

Example 4.3

In this example problem, there are 10 jobs to be processed in a two-machine no-wait flow shop. The data s_{1j}, s_{2j}, p_{1j}, p_{2j}, Q_j, and n_j for the jobs are shown in Table 4.6. These data are created randomly. The ranges of each type of parameter is: $0 \leq s_{1j}$, $s_{2j} < 10$; $0 < p_{1j}$, $p_{2j} \leq 10$; $10 \leq Q_j \leq 100$; and $3 \leq n_j \leq 7$. We assume that $C_h = 0.80$ and $C_{mh} = 8.00$. We compare results by changing following parameters:

- Temperature Range
- Cooling Speed
- Number of Iterations

In Experiment 1, we first let the temperature range be (100, 1000], the cooling speed be 1, and the number of iterations be 30. The relationship of *TC* and temperature and the final search results are shown in Figure 4.6.

A new experiment (Experiment 2) is set by changing the temperature range from (100, 1000] to (100, 500]. Other parameters are the as in Experiment 1. The search effect is shown in Figure 4.7.

We conducted several other experiments. The values of the parameters used in these experiments, including Experiments 1 and 2, are summarized in Table 4.7.

Corresponding search results (reduction of *TC*s) are shown in Figures 4.6 to 4.9.

Table 4.6 Input Data in Example 4.3

Job	s_1 (hour)	s_2 (hour)	p_1 (hour)	p_2 (hour)	Q (unit)	n
J_1	8	9	1	5	58	6
J_2	1	7	6	8	71	4
J_3	3	9	7	5	42	6
J_4	6	8	7	6	57	4
J_5	9	10	3	10	35	7
J_6	4	5	4	5	31	4
J_7	5	5	5	8	67	7
J_8	0	7	9	6	60	5
J_9	2	9	3	9	96	3
J_{10}	6	5	1	2	13	6

Table 4.7 SA Parameters Set in Different Experiments

SA Parameters	Experiment 1	Experiment 2	Experiment 3	Experiment 4
Initial Temperature	1000	500	1000	1000
Final Temperature	100	100	100	100
Cooling Speed	1	1	5	1
Number of Iteration	30	30	30	10

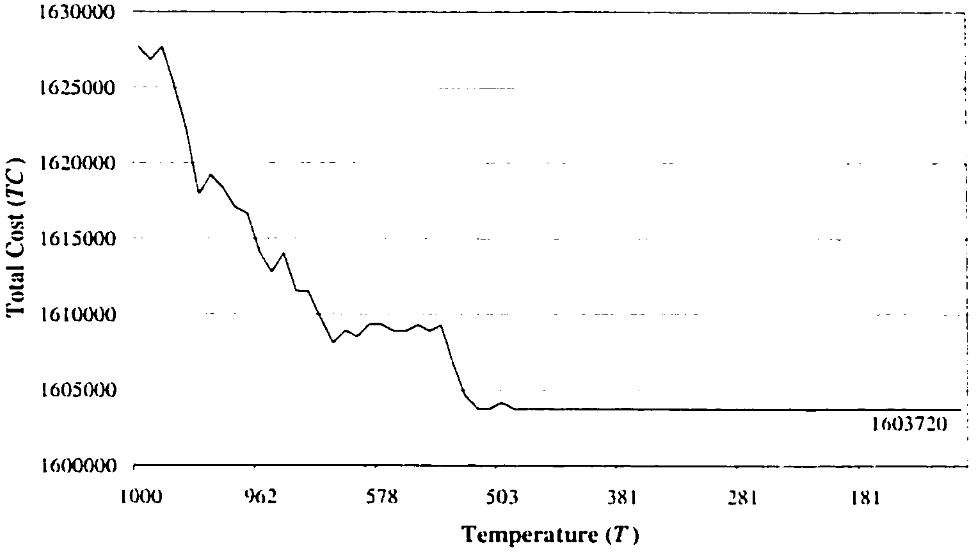

Figure 4.6 *TC-T* Relationship During Searching Procedure (Experiment 1)

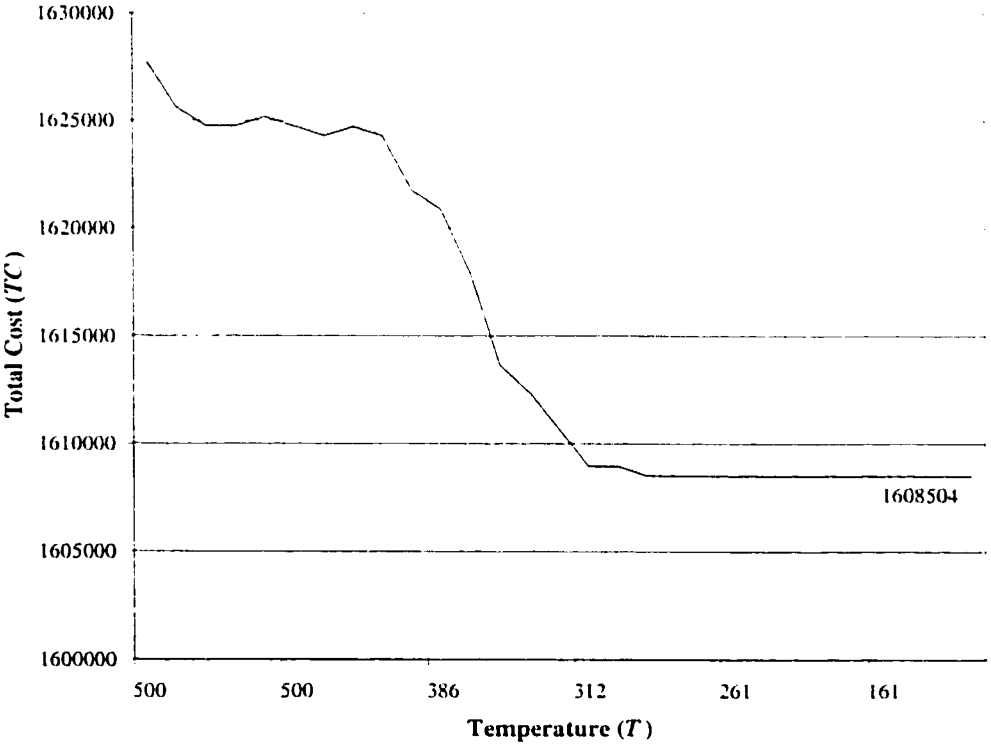

Figure 4.7 *TC-T* Relationship During Searching Procedure (Experiment 2)

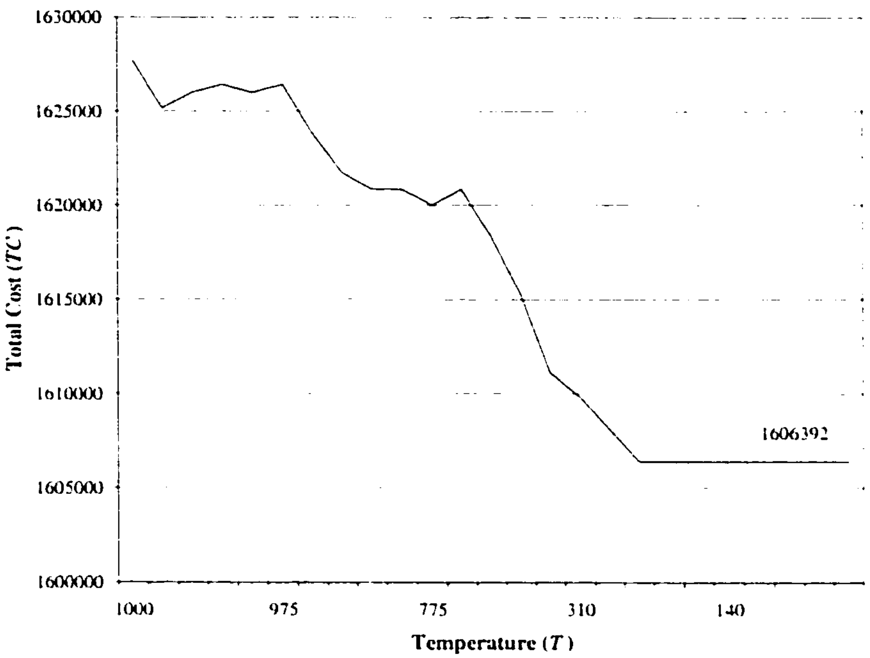

Figure 4.8 *TC-T* Relationship During Searching Procedure (Experiment 3)

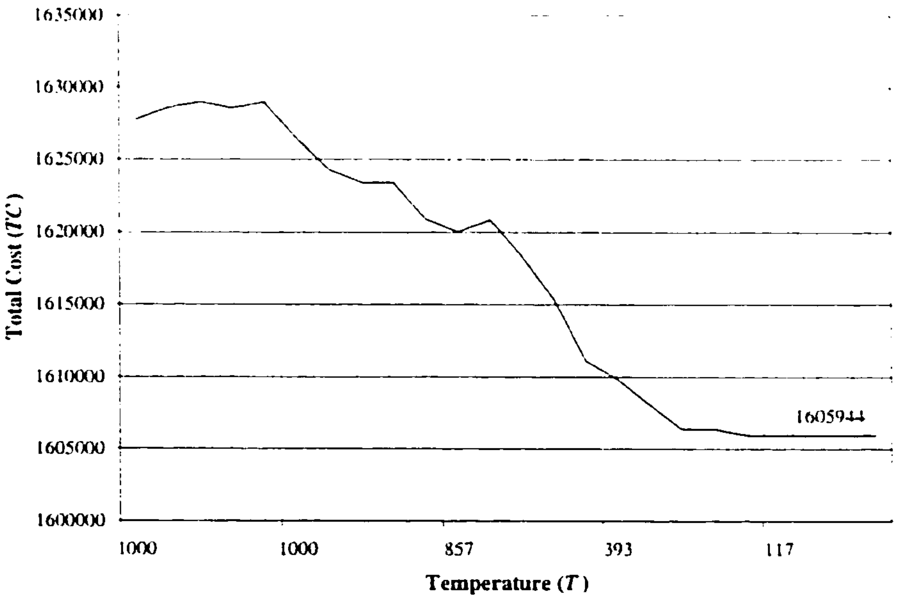

Figure 4.9 *TC-T* Relationship During Searching Procedure (Experiment 4)

Comparing Figure 4.6 and the other three figures, we can find that the latter three search results are not better than the first one. It demonstrates an outstanding feature of SA, i.e., SA demands extensive experimentation. To obtain high quality solutions, we have to adjust each parameter carefully. It often requires conducting a series of experiments.

4.3 Comparison of Two Influent Factors: Makespan and Number of Sublots

In our study, the *TC* is the sum of two cost components: holding cost (*HC*) and material handling cost (*MHC*). This decides that, to a given problem, we search for the optimal value of *TC* by either shortening the manufacturing duration or cutting down the times of conveyance, which equals the number of sublots. The relative level of values between *HC* and *MHC* decides the priority of decreasing makespan or number of sublots. We will compare these two factors in this section.

To a given problem, the total number of units *Q* is a constant. The variation between *HC* and *MHC* is decided by the variation between C_h and C_{mh}. To reflect the change, we introduce a coefficient λ ($0 \leq \lambda \leq 1$) into Equation (3.2). Then we have

$$TC = Q \times F \times \lambda \times C_h + n \times (1-\lambda) \times C_{mh} \qquad (4.1)$$

From Equation (4.1), when λ changes from 0 (*TC* = *HC*) to 1 (*TC* = *MHC*), we can observe the influence of makespan and number of sublots on *TC*.

Consider a problem with data given in Table 4.8. Since in most cases $C_{mh} > C_h$, we assume that the upper limits of $C_{mh} = 10.00$ and $C_h = 1.00$.

The initial makespan of the schedule corresponding to $\Sigma n_j = 17$ is 178. The initial TCs corresponding to different λ and the final results obtained by using SA algorithm are presented in Table 4.9. The impact of different λ on makespan and number of sublots is plotted in Figures 4.10 and 4.11.

Table 4.8 Input Data of Each Job in the Example of Comparing Two Influent Factors: Makespan and Number of Sublots

Job	s_1 (hour)	s_2 (hour)	p_1 (hour)	p_2 (hour)	Q (unit)	n
J_1	5	5	1	2	10	3
J_2	5	7	2	3	20	4
J_3	7	5	2	1	30	5
J_4	4	5	2	3	10	5
Total	-	-	-	-	70	17

Table 4.9 The Variation Tendency of Values of F and Σn_j With Different λ

λ	Initial Results	Final Results			$\Delta TC\%$ =
	TC_i	F	$\sum \mu_i$	TC_f	$(TC_i - TC_f) / TC_i$
0.0	170.00	182	4	40.00	76.47
0.1	1399.00	165	10	1245.00	11.01
0.3	3857.00	164	11	3521.00	9.28
0.5	6315.00	164	11	5795.00	8.42
0.7	8773.00	164	11	8069.00	8.11
0.9	11231.00	164	11	10343.00	7.96
1.0	12460.00	164	11	11480.00	7.87

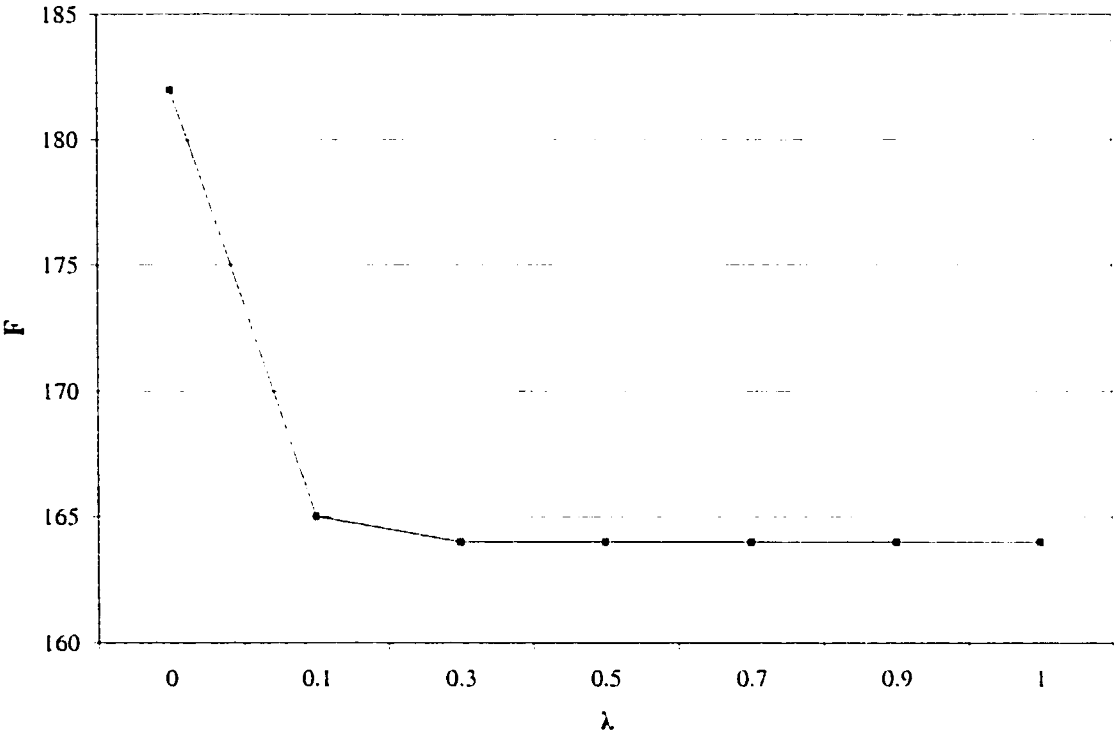

Figure 4.10 Variation of Makespan Corresponding to Different λ

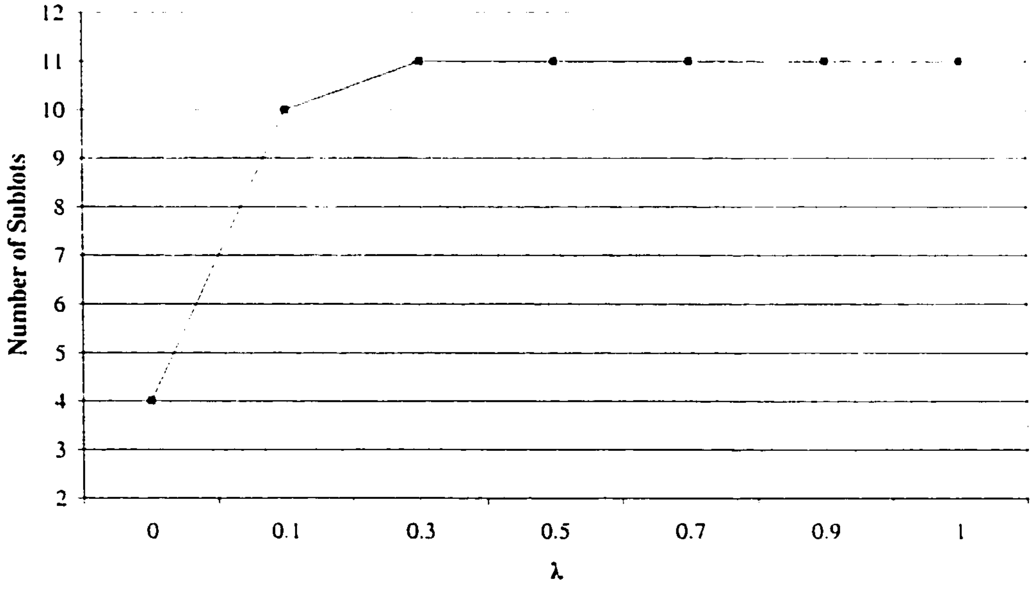

Figure 4. 11 Variation of Number of Sublots Corresponding to Different λ

From the computational results of this example, we have the following observations.

- When $\lambda = 0$, the only cost component is *MHC* in Equation (4.1). This means that we can minimize *TC* by decreasing the number of sublots. In fact, this result was obtained when $\Sigma\mu_j = 4$. This is to process the four jobs without lot streaming. It is noticed that in this situation $F = 182$ in the final state is larger than $F = 178$ in the initial state.
- When $\lambda = 1$, the only cost component is *HC* in Equation (4.1). In this situation, only the value of F decides the *TC*. Because the upper limit of the number of pallet is 17, we may infer that the minimum *TC* can be obtained when $\Sigma\mu_j = 17$. However, we find that the minimum *TC* is obtained while $\Sigma\mu_j = 11$. The reason is the no-wait production, i.e., when jobs are processed with no-wait in duration, increase in the number of sublots may extend the duration.
- Generally, a better *TC* more depends on makespan when C_h increases. Similarly, when C_{mh} is far large, decrease on the number of sublots will cut *TC*.
- From values of $\Delta TC\%$ in Table 4.9, we observe that *TC* can be reduced significantly by using SA search. The effectiveness of this method is clear.

4.4 Comparing Johnson's Method and Gilmore & Gomory's Method

Johnson's optimal sequence depends only on the values of the assembled numbers p_{1i} and p_{2i}, $i = 1, 2, \ldots, N$. Any change in these numbers does not change the optimal sequence.

However, in the method devised by Gilmore and Gomory, the sequence of N jobs comes from a permutation of N+1 jobs. To minimize the total cost, which happens while changing the initial state of a machine from one to another in order to process different product economically, they transform the problem into a Traveling Salesman Problem (TSP). p_{1i} and p_{2i} are distances between cities.

The result of the following example shows the differences of these two types of problems.

Table 4.10 Jobs and Their Processing Times on Two Machines

Job	1	2	3	4	5	6	7	8
p_1	5	2	1	7	6	3	7	5
p_2	2	6	2	5	6	7	2	1

- Sequence by exerting Johnson's method is: 3 – 2 – 6 – 5 – 4 – 7 – 1 – 8.
- Sequence by exerting Gilmore and Gomory's method is: 1 – 7 – 6 – 3 – 2 – 5 – 4 – 8.
- Two Gantt charts are shown in Figures 4.12 and 4.13.

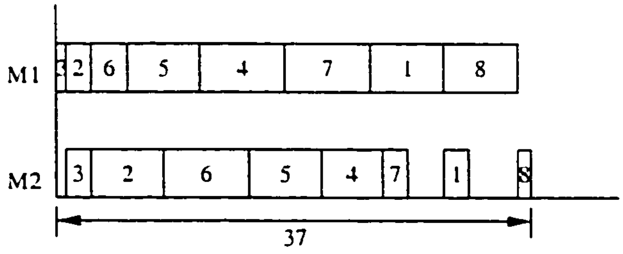

Figure 4.12 Gantt Chart by Johnson's Method

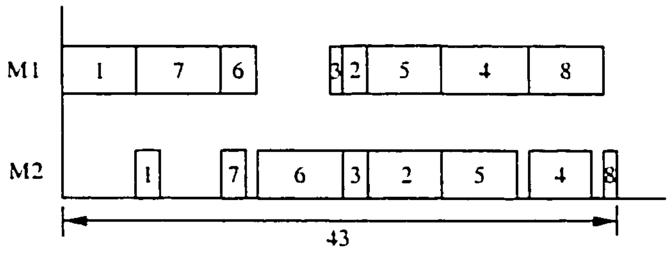

Figure 4.13 Gantt Chart by Gilmore and Gomory's Method

Comparing the two schedules, we can see that Johnson's method cannot replace Gilmore and Gomory's method because the sequence from Johnson's method cannot satisfy the constraint of no-wait.

4.5 Blocking situation

Finally, we discuss the validity of our results under the processing of sublots as blocking instead of no-wait.

The blocking mode of processing occurs when there are no buffers between the machines. When a sublot has completed processing on the first machine, it must remain on the machine until the second machine becomes available. In this case two types of detached setup can occur depending on the processing technology.

Type 1: the setup cannot be performed in anticipation of the arriving product until the sublot previously processed is removed from the machine.

Type 2: setup can be performed as soon as the previous sublot is completed on the machine even if it stays on the machine until the downstream machine becomes available.

It is easy to show that the makespan problem studied in Sriskandarajah and

Wagneur (1999) is equivalent to the problem with blocking under Type 1 setup. Thus, the heuristics methods developed in that paper are also valid for solving two-machine flow shop problems with blocking under Type 1 setup.

It was proven in Logendran and Sriskandarajah (1993) that the two-machine problem of simultaneous lot streaming and scheduling with blocking under Type 2 setup is NP-hard even if the number of sublots for each product is one. Because Type 2 setup is less restrictive than Type 1 setup, the heuristics presented in Sriskandarajah and Wagneur (1999) can also be used.

The method developed in this thesis can also be applied to solve multi-job two-machine flow shop scheduling problems with blocking.

Chapter 5

Solving Three-Machine Problems

5.1 Motivation

In Chapter 1, we discussed Johnson's algorithm. It is an optimal scheduling method for multiple jobs in a two-machine flow shop where there are buffers between the two machines. Under certain conditions, Johnson's algorithm can also generate optimal schedules for three-machine flow shop problems.

The method developed in this research can also be modified for three-machine problems. The three-machine problem is first transformed to a two-machine problem, and then the same method is used.

5.2 Prerequisite of Transformation

In Chapter 1, variable and consistent sublots were discussed. In a two-machine problem, it is not necessary to differentiate between variable sublots and consistent sublots. However, for a three-machine problem, they must be differentiated. The method developed in Chapter 3 for two-machine problems can be used to solve three-machine problems if sublots size of each job is the same in transferring between machine 1 and machine 2 as well as between machine 2 and machine 3.

5.3 Problem Transformation

To apply the two-machine algorithm, we first transform a three-machine problem to a two-machine problem by allocating the second machine processing times to the first and the third machines.

There are two approaches to allocate p_{2j} (j = 1, 2, ..., N). The first approach is splitting p_{2j} into two parts, p^1_{2j} and p^2_{2j}. p^1_{2j} is the product of p_{2j} and a number r ($0 < r \leq 1$), i.e., $p^1_{2j} = r \times p_{2j}$. $p^2_{2j} = p_{2j}$ - p^1_{2j}. Like this, in a transformed two-machine problem, processing time on M_1 and M_2 of job j can be expressed respectively as: $p'_{1j} = p_{1j} + p^1_{2j}$, $p'_{2j} = p_{3j} + p^2_{2j}$. The second approach is adding p_{2j} to p_{1j} and p_{3j} respectively, i.e., $p'_{1j} = p_{1j}$ + p_{2j}, $p'_{2j} = p_{3j} + p_{2j}$. We adopt the second allocating approach in this study.

5.4 Numerical Example

In this three-machine flow shop example, we assume 10 jobs to be processed on three machines in a flow shop with no-wait. Each job is to be processed on M_1 first and then on M_2 and finally on M_3. The setup times, processing times, quantity of items, and initial number of sublots for each job are given in Table 5.1. The objective is again to minimize the total cost TC. Here, TC is the sum of material handling cost from M_1 to M_2, and then from M_2 to M_3, and holding cost. In this example, we let C_h = 0.10 and C_{mh} = 8.00.

Table 5.1 Given Data for Three-Machine Problem

Job	s_1	s_2	s_3	p_1	p_2	p_3	Q	n
	(hour)	(hour)	(hour)	(hour)	(hour)	(hour)	(unit)	
J_1	9	4	1	20	2	14	78	6
J_2	3	3	2	1	9	7	75	6
J_3	3	5	8	5	7	17	44	4
J_4	6	8	4	16	20	12	78	5
J_5	0	1	0	2	12	15	31	3
J_6	7	5	1	2	5	5	13	3
J_7	0	1	2	10	15	3	75	6
J_8	6	3	6	7	12	14	33	4
J_9	2	7	2	1	17	12	92	7
J_{10}	6	8	3	15	2	12	78	8

The processing times on the second machine are allocated to the first and third machines. A new set of data was generated for the transformed two-machine problem. These data are shown in Table 5.2. When there are no setup times on the second machine, the data for transformed two-machine problem are shown in Table 5.3.

Applying the two-machine scheduling method with SA search, the final results for both situations, i.e., $s_{2j} \neq 0$ and $s_{2j} = 0$, are shown in Tables 5.4 and 5.5 respectively.

The scheduling method developed in the previous chapters is applied to solve this problem. *TC* reductions are shown in Figures 5.1 and 5.2. Figure 5.1 shows *TC* values in the search process when setup time on the second machine, s_{2j}, is considered, Figure 5.2 shows *TC* values in the search process when $s_{2j} = 0$ for all jobs is assumed.

Table 5.2 Transformed Data for Two-Machine Problem When $s_{2j} \neq 0$

Job	s_1	s_2	s_3	$\dot{p}_1$	$\dot{p}_2$	Q	n
	(hour)	(hour)	(hour)	(hour)	(hour)	(unit)	
J_1	9	4	1	22	16	78	6
J_2	3	3	2	10	16	75	6
J_3	3	5	8	12	24	44	4
J_4	6	8	4	36	32	78	5
J_5	0	1	0	14	27	31	3
J_6	7	5	1	7	10	13	3
J_7	0	1	2	25	18	75	6
J_8	6	3	6	19	26	33	4
J_9	2	7	2	18	29	92	7
J_{10}	6	8	3	17	14	78	8

Table 5.3 Transformed Data for Two-Machine Problem When $s_{2j} = 0$

Job	s_1	s_2	s_3	$\dot{p}_1$	$\dot{p}_2$	Q	n
	(hour)	(hour)	(hour)	(hour)	(hour)	(unit)	
J_1	9	0	1	22	16	78	6
J_2	3	0	2	10	16	75	6
J_3	3	0	8	12	24	44	4
J_4	6	0	4	36	32	78	5
J_5	0	0	0	14	27	31	3
J_6	7	0	1	7	10	13	3
J_7	0	0	2	25	18	75	6
J_8	6	0	6	19	26	33	4
J_9	2	0	2	18	29	92	7
J_{10}	6	0	3	17	14	78	8

Table 5.4 Final Results When $s_{2j} \neq 0$

Job	μ_j	Y_j
J_1	6	25, 18, 13, 10, 7, 5
J_2	1	75
J_3	1	44
J_4	3	29, 26, 23
J_5	1	31
J_6	1	13
J_7	1	75
J_8	2	14, 19
J_9	2	35, 57
J_{10}	1	78

- The sequence and makespan are obtained concurrently as: σ: 6 – 8 – 5 – 3 – 9 – 7 – 10 – 2 – 4 – 1 and F = 8339.

Table 5.5 Final Results When $s_{2j} = 0$

Job	μ_j	Y_j
J_1	6	25, 18, 13, 10, 7, 5
J_2	1	75
J_3	1	44
J_4	3	29, 26, 23
J_5	1	31
J_6	1	13
J_7	1	75
J_8	2	14, 19
J_9	2	35, 57
J_{10}	1	78

- The sequence and makespan are obtained concurrently as: σ: 6 – 8 – 5 – 3 – 9 – 7 – 10 – 2 – 4 – 1 and F = 8329.

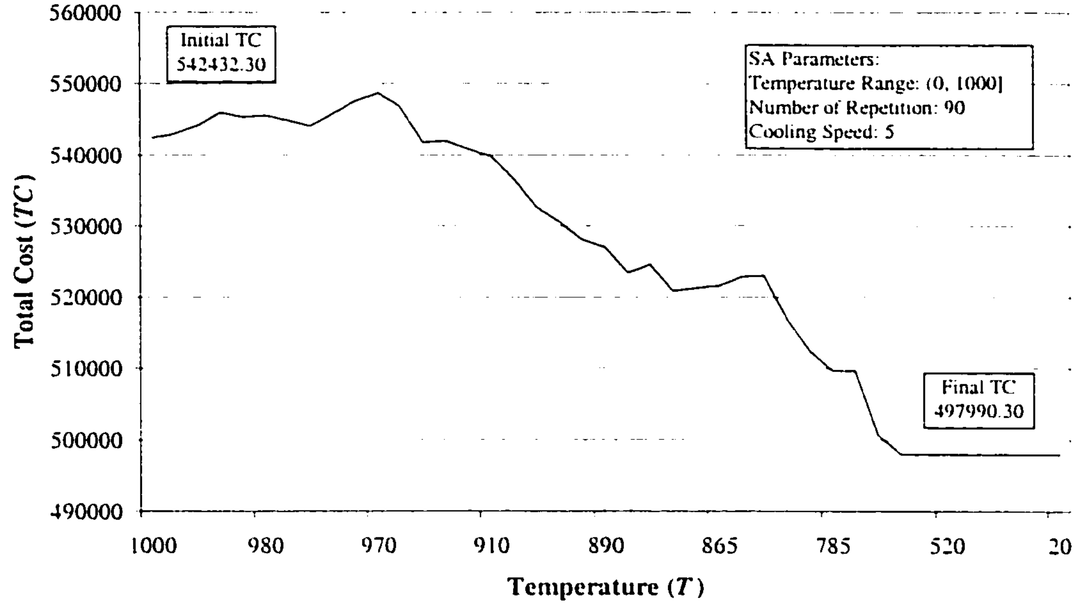

Figure 5.1 Search Process When $s_{2j} \neq 0$

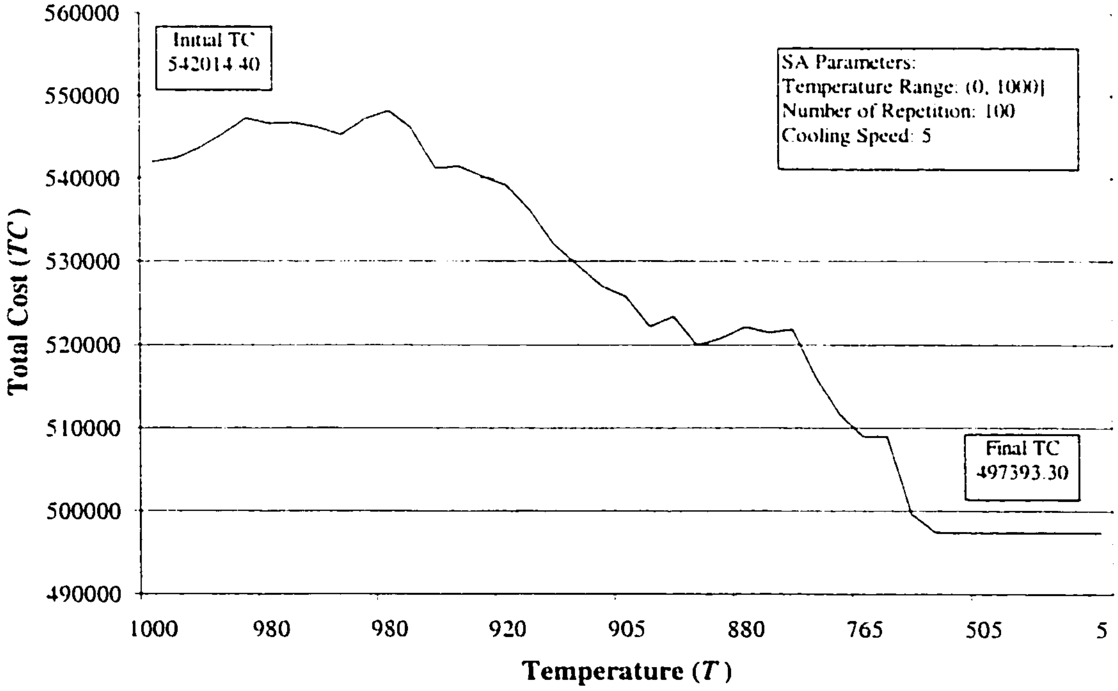

Figure 5.2 Search Process When $s_{2j} = 0$

5.5 Conclusion

This example shows that the heuristic method developed in this research for solving two-machine flow shop problems can be applied to solve three-machine problems. Since the developed algorithm is heuristic in nature, it is a heuristic method in solving three-machine problems. It may be expanded to solve general multi-machine flow shop problems.

Chapter 6

Conclusions

6.1 Summary

In this research, multi-job two-machine flow shop scheduling problems are studied. The objective of the scheduling is to minimize the sum of material handling cost and the cost related to makespan. This study is limited to manufacturing systems in which jobs have to be treated with no-wait in process.

Chapter 1 presents an introduction to the systems and the problems considered in this research. A literature survey of lot streaming and scheduling methods was given in Chapter 2. In Chapter 3, a cost model to minimize the sum of holding cost and material handling cost, in a two-machine flowshop with no-wait in process, is built. Because of computational complexity of the problem, a heuristic method (S & W method) is modified and used. The core of this heuristic method is the algorithm due to Gilmore and Gomory to obtain optimal solutions for its sub-problems in polynomial time. Based on the heuristic method, simulated annealing search is used to find the best combinations of the number of sublots to minimize the total cost.

With three numerical examples in Chapter 4, we can observe that the heuristic method is effective in reducing the total cost. We also studied the relationship between the makespan and the number of sublots.

In Chapter 5, we extend the application of the heuristic method to three-machine no-wait flow shops. A numerical example demonstrates the effectiveness of this method in solving three-machine problems.

6.2 Conclusions

In an advanced job shop-type manufacturing system, there may be several computer numerically controlled (CNC) machining centers or workstations served by industrial robot arms. To reduce production costs, job shop production is normally carried out in batches due to machine setup and material handling costs. On the other hand, market demands for cutting down delivery time require short makespan. Effective operations management leads to minimization of work-in-process (WIP) inventory. This however may increase material handling costs. Consequently, a new cost control technique should be developed to meet the challenge.

When there are multiple jobs to be processed, the sequence of these jobs plays an important role in reducing the makespan. The complex interrelationship between lot streaming of each job and job sequencing makes the scheduling problem NP-hard. It is NP-hard for a flow shop problem even with only two machines.

Research in this thesis concentrates on minimization of the total cost in a deterministic multi-job two-machine flow shop. The solutions are to decide the best sublots size of each job and the best job processing sequence. A mathematical model is formulated to achieve this purpose. The main assumptions used in developing the model are 1) the analysis is confined to the static flow shop; 2) jobs are processed with no-wait; 3) jobs are non-preemptive; 4) the number of sublots of each job is known in advance; and 5) setups of jobs are independent of the sequence.

We achieve our goal through two steps.

First, we relax the original problem of cost minimization to the problem of makespan minimization by giving the number of sublots before lot streaming. We analyze the lot streaming of a single job in a two-machine flow shop with no-wait in process. With linear programming, we obtain the closed expression of its sublot sizes in continuous version. The S & W method was used to solve the problem.

Following the first step, we implemented the Simulated Annealing (SA) technique to search for the global optimal solution. Each search neighborhood is obtained by changing the value of the number of sublots by one. Finally, the optimal or near-optimal results are found. Because the G & G algorithm is embedded in the heuristic method, each search step will reveal the best processing sequence of jobs.

Our heuristic method is coded in C language. Several numerical examples are solved with the program. The effectiveness of the method is shown. In addition, we studied the performance of our method in a three-machine flow shop with no-wait in process. We showed that in a three-machine flow shop, our method can be used to generate reasonable results.

Although we include only holding cost and material handling cost in the total cost, many other akin cost components can be included conveniently because manufacturing cycle, quantity of pallets and fixtures as well as the frequency of dispatching material handlers can be calculated accurately. Furthermore, if an additional job arrives while the current production plan is being carried out, we may adjust the current production plan dynamically with our method. We can combine the new job and those unprocessed jobs into a new job group. A new production plan can be obtained by running the application program. By comparing the original plan with the new one, we can choose to carry out the original production plan continuously or to modify the sequence of those unprocessed jobs according to the new production plan.

6.3 Suggestions for Future Work

The numerical examples in this study show that the searched total numbers of sublots are normally less than the given total number of sublots. It means that sublots should be transferred and processed in larger size. In a real manufacturing system, this result may exceed the capacity of either material handling equipments or processing machines. For this reason, we suggest that factors concerned with capacity of both material handling equipments and machines should be considered in future work in this area. Extending the method developed in this research to solve multi-machine flow shop problems should also be considered for future research. In addition, use of other search methods such as Tabu search and genetic algorithm for solving similar problems should be studied.

References

1. Anderson, E. J., 1994, *The Management of Manufacturing – Models and Analysis*, Addison-Wesley, Workingham, UK.
2. Baker, K. R. & Pyke, D. F., 1990, "Solution procedures for the lot-streaming problem", *Decision Sciences*, Vol. 21, 475-491.
3. Baker, K. R., 1995, "Lot streaming in the two-machine flow shop with setup times", *Annals of Operations Research*, Vol. 57, 1-11.
4. Collins, N., Eglese, R. & Golden, B., 1989, "Simulated annealing – an annotated bibliography", *American Journal of Mathematical and Management Sciences*, Vol. 8, 209-307.
5. Egbelu, P. J., 1990, "Machining and material flow system design for minimum cost production", *International Journal of Production Research*, Vol. 28, No. 2, 353-368.
6. Egbelu, P. J., 1991, "Batch production time in a multi-stage with material-handling consideration", *International Journal of Production Research*, Vol. 29, No. 4, 695-712.
7. French, S., 1982, *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*, Wiley, New York, NY.
8. Gilmore, P. G. & Gomory, R. E., 1964, "Sequencing a one state-variable machine: a solvable case of the traveling salesman problem", *Operations Research*, Vol. 12, 655-679.

9. Glass, C. A. & Potts, C. N., 1998, "Structural properties of lot streaming in a flow shop", *Mathematics of Operations Research*, Vol. 23, No. 3, 624-639.

10. Goyal, S. K., 1976, "Note on manufacturing cycle time determination for a multi-stage economic production quantity model", *Management Science*, Vol. 23, No. 3, 332-333.

11. Hall, N. G. & Sriskandarajah, C., 1996, "A survey of machine scheduling problems with blocking and no-wait in process", *Operations Research*, Vol. 44, No. 3, 510-525.

12. Kirkpatrick, S., Gelatt, C. D. & Vecchi, M. P., 1983, "Optimization by simulated annealing", *Science*, Vol. 220, 671-680.

13. Langevin, A., Riopel, D. & Stecke, K. E., 1999, "Transfer batch sizing in flexible manufacturing systems", *Journal of Manufacturing Systems*, Vol. 18, No. 2, 140-151.

14. Logendran, R. & Sriskandarajah, C., 1993, "Two-machine group scheduling problem with blocking and anticipatory setups", *European Journal of Operational Research*, Vol. 69, 467-481.

15. Nahmias, S., 2001, *Production and Operations Analysis*, 4th Edition, McGraw-Hill/Irwin, Boston, MA.

16. Potts, C. N. & Baker K. R., 1989, "Flow shop scheduling with lot streaming", *Operations Research Letters*, Vol. 8, 297-303.

17. Potts, C. N. & Van Wassenhove, L. N., 1992, "Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity", *Journal of Operation Research Society*, Vol. 43, No. 5, 395-406.

18. Sriskandarajah, C. & Wagneur, E., 1999, "Lot streaming and scheduling multiple products in two-machine no-wait flowshops", *IIE Transactions*, Vol. 31, 695-707.

19. Steiner, G. & Truscott, W. G., 1993, "Batch scheduling to minimize cycle time, flow time, and processing cost", *IIE Transactions*, Vol. 23, 90-97.

20. Szendrovits, A. Z., 1975, "Manufacturing cycle time determination for a multi-stage economic production quantity model", *Management Science*, Vol. 22, 298-308.

21. Szendrovits, A. Z., 1976, "On the optimality of sub-batch sizes for a multi-stage EPQ model – A Rejoinder", *Management Science*, Vol. 23, No. 3, 334-338.

22. Tanaev, V. S., Sotskov, Y. N. & Strusevich, V. A., 1994, *Scheduling Theory. Multi-Stage Systems*, Kluwer Academic Publishers, Boston, MA

23. Trietsch, D. & Baker, K. R., 1993, "Basic techniques for lot streaming", *Operations Research*, Vol. 41, No. 6, 1065-1076.

24. Vickson, R. G. & Alfredsson, B. E., 1992, "Two- and three-machine flow shop scheduling problems with equal sized transfer batches", *International Journal of Production Research*, Vol. 30, No. 7, 1551-1574.

25. Vickson, R. G., 1995, "Optimal lot streaming for multiple products in a two-machine flow shop", *European Journal of Operational Research*, Vol. 85, 556-575.

Appendix

Code for the Program

```
#include "stdlib.h"
#include "stdio.h"
#include "math.h"
#include "time.h"

#define LEN sizeof (struct job)
#define INPUT_JOB scanf("%d %d %d %d %d %d %d %d %d\n",&p1-
                  >num,&p1->s1,&p1->s2,&p1->s3,&p1->a,&p1->b,&p1-
                  >c,&p1->q,&p1->n)
#define job_size (the number of jobs)
#define sublot_size   (the number of the maximum number of
                      sublots)
#define total_sublot_size(the number of total sublots)
#define tmp_h    (the value of initial temperature)
#define tmp_l    (the value of final temperature)
#define Ch       (the value of unit holding cost)
#define Cmh      (the value of unit material handling cost)
#define col_spd (the value of cooling speed)
#define V        (the value of number of iteration)
#define seed     (random number between 0,1)

struct job
{int num;
 int s1;
 int s2;
 int s3;
 int a;
 int b;
 int c;
 int q;
 int n;
 struct job *next;
};
int N;
int M;
int arr_job3[job_size][9]={0};
int arr_job[job_size][11]={0};
int arr[job_size][13]={0};
double arr_X[total_sublot_size]={0}, *p_X=arr_X;
int arr_Y[total_sublot_size]={0}, *p_Y=arr_Y;
int Y[job_size][sublot_size]={0};
int cm[4][4]={0};
int arr_T[total_sublot_size]={0}, *p_T=arr_T;
int arr_n[total_sublot_size]={0}, *p_n=arr_n;
double arr_TC[total_sublot_size]={0}, *p_TC=arr_TC;

struct job *creat();
void print(struct job *head);
void H_l1(int a,int b,int q,int n,double arr_x[sublot_size],int
          arr_y[sublot_size]);
int *H_g1(struct job *head);
int H(int s,int t,int a,int y1);
int T(int b,int yn);
int *GG(int *array, int n);
int max(int a, int b);
int min(int a, int b);
int alpha(int e,int g,int h);
int msp(int a,int b,int q,int n,int s,int t,int
arr_y[sublot_size]);
int *MSP(int *array);
int *sim_anl(double TC0);
```

```
void main()

{struct job *head;
 int i,j,*p,*ty,cyc=0,plt=0,qut=0;
 double TC, *tx;

 head=creat();
 print(head);
 printf("\nThe total number of jobs is: N = %d\n",N);
 p=H_g1(head);
 printf("\nThe total number of machines is: M = %d\n",M);

p_X=arr_X;
p_Y=arr_Y;
printf("\nThe original sublots size and H,B,T of each job are
       those as following:\n");
for(j=1;j<=N;j++)
{
      printf("job%d:\n", j);
      tx=p_X;
      ty=p_Y;
      for(p_X=tx+1,p_Y=ty+1;p_X<=tx+arr_job[j][6],p_Y<=ty+arr_job
           [j][6];p_X++,p_Y++)
      {
            printf("\tX=%f", *p_X);
            printf("\tY=%d\n", *p_Y);
      }
      printf("\tH=%d\tB=%d\tT=%d\n",arr_job[j][7],arr_job[j][8],
              arr_job[j][9]);
}

for(i=1;i<=N;i++)
{
      cyc=cyc+*(p+(i*13+5))+*(p+(i*13+10));
      plt=plt+*(p+(i*13+11));
      qut=qut+*(p+(i*13+12));
}
if(M==2)   cyc=cyc;
if(M==3)   cyc=cm[3][3];
TC=cyc*qut*Ch+plt*Cmh;

printf("MSP=%d\tPLT=%d\tQUT=%d\tOTC=%f\n",cyc,plt,qut,TC);
printf("The original sequence is:\t");
for(i=1;i<=N;i++)
    printf("%d->", *(p+(i*13+9)));
printf("\n");

p=sim_anl(TC);

p_Y=arr_Y;
printf("\nThe final sublots size and H,B,T of each job are those
         as following:\n");
for(j=1;j<=N;j++)
{
      printf("job%d:\n", j);
      ty=p_Y;
      for(p_Y=ty+1;p_Y<=ty+arr_job[j][6];p_Y++)
          printf("\tY=%d\n", *p_Y);
      printf("\tH=%d\tB=%d\tT=%d\n",arr_job[j][7],arr_job[j][8],
             arr_job[j][9]);
}

cyc=0;
plt=0;
qut=0;

for(i=1;i<=N;i++)
{
      cyc=cyc+(*(p+(i*13+5)))+(*(p+(i*13+10)));
      plt=plt+*(p+(i*13+11));
      qut=qut+*(p+(i*13+12));
}
if(M==2)   cyc=cyc;
if(M==3)   cyc=cm[3][3];
TC=cyc*qut*Ch+plt*Cmh;

printf("MSP=%d\tPLT=%d\tQUT=%d\tFTC=%f\n",cyc,plt,qut,TC);
printf("The final sequence is:\t");
for(i=1;i<=N;i++)
      printf("%d->", *(p+(i*13+9)));
printf("\n");
}


struct job *creat()

{struct job *head;
 struct job *p1,*p2;

N=0;
p1=p2=(struct job *)malloc(LEN);
INPUT_JOB;
head=0;
while (p1->num!=0)
{
      N=N+1;
      if(N==1) head=p1;
      else   p2->next=p1;
      p2=p1;
      p1=(struct job *)malloc(LEN);
      INPUT_JOB;
}
p2->next=0;
return(head);
}
```

```
void print(struct job *head)

{struct job *p;

printf("\nThe given data of all jobs are:\n");
p=head;
if(p!=0)
     do
      {
           printf("\tjob=%d: s1=%d s2=%d s3=%d a=%d b=%d c=%d
               q=%d n=%d\n", p->num,p->s1,p->s2,p->s3,p->a,p-
               >b,p->c,p->q,p->n);
            p=p->next;
      }
     while(p!=0);
}


int *H_g1(struct job *head)

{struct job *p;
int i,dvd2_1=0,sum=0,arr_mu[job_size]={0};
int *pp;

p=head;
for(i=1;p!=0;i++)
{
     arr_job3[i][0]=p->num;
     arr_job3[i][1]=p->s1;
     arr_job3[i][2]=p->s2;
     arr_job3[i][3]=p->s3;
     arr_job3[i][4]=p->a;
     arr_job3[i][5]=p->b;
     arr_job3[i][6]=p->c;
     arr_job3[i][7]=p->q;
```

```
    arr_job3[i][8]=p->n;
    p=p->next;
}
for(i=1;i<=N;i++)
    sum=sum+arr_job3[i][5];
if(sum==0) M=2;
else       M=3;
for(i=1;i<=N;i++)
{
    arr_job[i][0]=arr_job3[i][0];
    arr_job[i][1]=arr_job3[i][1];
    arr_job[i][2]=arr_job3[i][3];
    arr_job[i][3]=arr_job3[i][4]+arr_job3[i][5];
    arr_job[i][4]=arr_job3[i][6]+arr_job3[i][5];
    arr_job[i][5]=arr_job3[i][7];
    arr_job[i][6]=arr_job3[i][8];
}
for(i=1;i<=N;i++)
     arr_mu[i]=arr_job[i][6];
pp=MSP(arr_mu);
return(pp);
}


int *MSP(int *array)

{double arrx[sublot_size], *p1=arrx, *tx;
int arry[sublot_size], *p2=arry;
int i,f,*p,*ty,*p_arr;

p=array;
p_X=arr_X;
p_Y=arr_Y;
for(i=1;i<=N;i++)
{
      arr_job[i][6]=*(p+i);
      H_11(arr_job[i][3],arr_job[i][4],arr_job[i][5],
           arr_job[i][6],arrx,arry);
      arr_job[i][7]=H(arr_job[i][1],arr_job[i][2],arr_job[i][3],
                        arry[1]);
      arr_job[i][9]=T(arr_job[i][4],arry[arr_job[i][6]]);
      arr_job[i][10]=msp(arr_job[i][3],arr_job[i][4],
                          arr_job[i][5],arr_job[i][6],
                          arr_job[i][1],arr_job[i][2],arry);
      arr_job[i][8]=arr_job[i][10]-arr_job[i][7]-arr_job[i][9];
      tx=p_X;
      ty=p_Y;
      for(p_X=tx+1,p1=arrx+1;p_X<=(tx+arr_job[i][6]),p1<=(arrx+
           arr_job[i][6]);p_X++,p1++)
      *p_X=*p1;
      for(p_Y=ty+1,p2=arry+1,f=1;p_Y<=(ty+arr_job[i][6]),
           p2<=(arry+arr_job[i][6]),f<=arr_job[i][6];
           p_Y++,p2++,f++)
      {
            *p_Y=*p2;
            Y[i][f]=*p2;
      }
}
p_arr=GG(*arr_job, N);
return(p_arr);
}

int max(int a, int b)

{int t;

if(a>=b)    {t=a;a=b;b=t;}
else        t=b;
return(t);
}
```

```
int min(int a, int b)

{int t;

if(a<=b)    {t=a;a=b;b=t;}
else        t=b;
return(t);
}


int alpha(int e,int g,int h)

{int alpha;

if(e!=h&&g==h)   alpha=e;
if(e==h&&g!=h)   alpha=g;
if(e!=h&&g!=h)   alpha=h;
return(alpha);
}


void H_11(int a,int b,int q,int n,double arr_x[sublot_size],
           int arr_y[sublot_size])

{int h,i,y,w0=0,w1=q,maxo;
 double maxd=0.0,sigma=0.0;
 int arro[sublot_size];
 double arrd[sublot_size];

for(i=1;i<=n;i++)
     sigma=pow(a,n-1)+b*sigma/a;
for(i=1;i<=n;i++)
{
     arr_x[i]=(pow(a,n-i)*pow(b,i-1))*(q/sigma);
     arro[i]=i;
```

```
    y=((int) arr_x[i])+1;
    w0=w0+y;
    arr_y[i]=y;
    arr_y[i]>1?arrd[i]=arr_y[i]-arr_x[i]:arrd[i]=0.0;
}
w0=w0-w1;
while(w0>0)
{
    for(i=1;i<=n;i++)
    {
        if(maxd>=arrd[i])      continue;
        maxd=arrd[i];
        maxo=i;
    }
    if(i==(n+1)&&maxd==0.0)
    {
        for(h=1;h<=n;h++)
        {
            if(arrd[h]==0)   continue;
            maxd=arrd[h];
            maxo=h;
            break;
        }
        for(h=1;h<=n;h++)
        {
            if(arrd[h]==0)   continue;
            else if(maxd>=arrd[h])      continue;
            else  {maxd=arrd[h];maxo=h;}
        }
    }

    i=maxo;
    arr_y[i]=arr_y[i]-1;
    if(arr_y[i]==1)  arrd[i]=0.0;
    else             arrd[i]=arr_y[i]-arr_x[i];
```

```
      maxd=0.0;
      w0=w0-1;
}
}

int H(int s,int t,int a,int y1)

{int h,H;
h=s-t+a*y1;
H=max(h,0);
return H;
}

int T(int b,int yn)

{int T;
T=b*yn;
return T;
}

int msp(int a,int b,int q,int n,int s,int t,
          int arr_y[sublot_size])

{int i,max1,c=0,max2,C;

max1=s-t+a*arr_y[1];
max1=max(max1,0);
for (i=2;i<=n;i++)
{
      max2=a*arr_y[i]-b*arr_y[i-1];
      max2=max(max2,0);
      c=max2+c;}
      C=t+b*q+max1+c;
      return C;
}
```

```
int *GG(int *array, int n)

{int *p;
int edge[job_size][5]={0},g[job_size][4]={0},cm[4][4]={0};
int group[job_size+1][job_size+1]={0};
int f,h1,h2,i,j,s,t,k,m,y,q,q1,q2,ng,gp1,gp2,count;

p=array;
for(i=1;i<=n;i++)
{
      arr[i][0]=*(p+(i*11+0));
      arr[i][4]=arr[i][1]=*(p+(i*11+7));
      arr[i][2]=*(p+(i*11+9));
      arr[i][10]=*(p+(i*11+8));
      arr[i][11]=*(p+(i*11+6));
      arr[i][12]=*(p+(i*11+5));
}

for (j=1;j<n;j++)
for (i=1;i<=n-j;i++)
if(arr[i][2]>arr[i+1][2])
{
      arr[0][0]=arr[i][0];
      arr[0][1]=arr[i][1];
      arr[0][2]=arr[i][2];
      arr[0][10]=arr[i][10];
      arr[0][11]=arr[i][11];
      arr[0][12]=arr[i][12];
      arr[i][0]=arr[i+1][0];
      arr[i][1]=arr[i+1][1];
      arr[i][2]=arr[i+1][2];
      arr[i][10]=arr[i+1][10];
      arr[i][11]=arr[i+1][11];
      arr[i][12]=arr[i+1][12];
      arr[i+1][0]=arr[0][0];
```

```
        arr[i+1][1]=arr[0][1];
        arr[i+1][2]=arr[0][2];
        arr[i+1][10]=arr[0][10];
        arr[i+1][11]=arr[0][11];
        arr[i+1][12]=arr[0][12];
}

for (j=1;j<n;j++)
for (i=1;i<=n-j;i++)
if(arr[i][4]>arr[i+1][4])
{
        t=arr[i][4];
        arr[i][4]=arr[i+1][4];
        arr[i+1][4]=t;
}

for (j=1;j<=n;j++)
{
        for (i=1;arr[j][1]!=arr[i][4];i++)
                continue;
        arr[i][3]=j;
        arr[i][4]=-1;
}

p=array;
for(i=1;i<=n;i++)
       arr[i][4]=*(p+(i*11+7));
for (j=1;j<n;j++)
for (i=1;i<=n-j;i++)
if(arr[i][4]>arr[i+1][4])
{
        t=arr[i][4];
        arr[i][4]=arr[i+1][4];
        arr[i+1][4]=t;
}
```

```
for(i=0;i<=12;i++)
    arr[0][i]=0;

for(j=1;j<n;j++)
{
    m=max(0,(min(arr[j+1][2],arr[j+1][4])-
        max(arr[j][2],arr[j][4])));
    edge[j][0]=m;
}

for(i=1;i<=n;i++)
    arr[i][6]=0;

for(j=1;j<=n;j++)
{
    if(arr[j][6]!=0) continue;
    if(arr[j][3]==j)
    {
        for(i=1;group[i][1]!=0&&i<=n;i++)
            continue;
        group[i][1]=j;
        arr[j][6]=1;
    }
    else
    {
        for(i=1;group[i][1]!=0&&i<=n;i++)
            continue;
        m=i;
        s=j;
        y=1;
        while(arr[s][3]!=j)
        {
            arr[s][6]=1;
            group[m][y]=s;
```

```
                  for(i=1;i!=arr[s][3];i++)
                        continue;
                  s=i;
                  y=y+1;
            }
            arr[s][6]=1;
            group[m][y]=s;
      }
}

for(i=1;i<=n;i++)
      arr[i][6]=0;

for(i=1;group[i][1]!=0;i++)
      continue;
ng=i-1;

if(ng==1)  goto step7;

count=0;
while(count<ng-1)
{
      for(m=1;m<=ng;m++)
      {
            for(y=1;group[m][y]!=0;y++)
                  continue;
            group[m][0]=y-1;
            for(j=1;j<=group[m][0]-1;j++)
            {
                  for(y=1;y<=group[m][0]-j;y++)
                  {
                        if(group[m][y]>group[m][y+1])
                        {
                              t=group[m][y];
                              group[m][y]=group[m][y+1];
```

```
                group[m][y+1]=t;
            }
        }
    }
}

for(i=1;i<=ng;i++)
{
    if(group[i][1]!=0)    continue;
    for(y=0;y<=group[i+1][0];y++)
        group[i][y]=group[i+1][y];
    for(j=0;j<=n;j++)
        group[i+1][j]=0;
}



for(i=1;i<=n-1;i++)
{
    edge[i][1]=0;
    edge[i][3]=i;
    edge[i][4]=888888888;
}



for(m=1;m<=ng;m++)
{
    for(y=1;group[m][y]!=0&&group[m][y]!=n;y++)
    {
        if(group[m][y+1]==group[m][y]+1) continue;
        else  edge[group[m][y]][1]=1;
    }
}

m=1;
for(y=1;group[m][y]!=0;y++)
```

```
        continue;
gp1=group[m][y-1];
for(y=1;group[m+1][y]!=0;y++)
        continue;
gp2=group[m+1][y-1];

        h1=max(gp1,gp2);
        h2=group[m+1][1];

        for(i=h2-1;i<h1;i++)
        {
                if((edge[i][1]==0)||(edge[i][2]==1))   continue;
                else  edge[i][4]=edge[i][0];
        }

        for(j=1;j<n-1;j++)
        {
                for(i=1;i<=n-1-j;i++)
                {
                        if (edge[i][4]>edge[i+1][4])
                        {
                                t=edge[i][4];
                                edge[i][4]=edge[i+1][4];
                                edge[i+1][4]=t;
                                s=edge[i][3];
                                edge[i][3]=edge[i+1][3];
                                edge[i+1][3]=s;
                        }
                }
        }

        k=edge[1][3];
        edge[k][2]=1;
```

```
if(arr[k][4]>=arr[k][2])
{
        for(q=1;g[q][0]!=0&&g[q][1]!=0;q++) continue;
        g[q][0]=k;
        g[q][1]=k+1;
}
else
{
        for(q=1;g[q][2]!=0&&g[q][3]!=0;q++) continue;
        g[q][2]=k;
        g[q][3]=k+1;
}
count=count+1;
if(count==ng-1)  break;

for(i=1;i<=ng;i++)
{
        for(y=1;y<=group[i][0];y++)
        {
                if(group[i][y]!=k)     continue;
                q1=i;
        }
}

for(i=1;i<=ng;i++)
{
        for(y=1;y<=group[i][0];y++)
        {
                if(group[i][y]!=k+1)  continue;
                q2=i;
        }
}

for(i=1;i<=group[q2][0];i++)
        group[q1][group[q1][0]+i]=group[q2][i];
        for(i=0;i<=n;i++)
            group[q2][i]=0;
}

step7: for(i=1;i<=n-1;i++)
      {
          g[0][0]=g[0][0]+g[i][0];
          g[0][2]=g[0][2]+g[i][2];
      }
      g[n][0]=0;g[n][1]=1;
      s=g[0][0];
      t=g[0][2];

if(s==0)   s=1;
    for(j=1;j<n+1;j++)
        for(i=1;i<=n+1-j;i++)
            if(g[i][1]<g[i+1][1])
            {
                g[0][0]=g[i][0];
                g[0][1]=g[i][1];
                g[i][0]=g[i+1][0];
                g[i][1]=g[i+1][1];
                g[i+1][0]=g[0][0];
                g[i+1][1]=g[0][1];
            }
if(!t==1)  t=0;
else
{
    for(j=1;j<n-1;j++)
        for(i=1;i<=n-1-j;i++)
            if(g[i][2]>g[i+1][2]&&(!g[i+1][2])==0)
            {
                g[0][2]=g[i][2];
                g[0][3]=g[i][3];
```

```
                        g[i][2]=g[i+1][2];
                        g[i][3]=g[i+1][3];
                        g[i+1][2]=g[0][2];
                        g[i+1][3]=g[0][3];
                  }
}

j=0;
if(t==0&&s==0)
{
      for(k=1;k<=n;k++)
            arr[k][8]=arr[arr[k][3]][0];
      goto order;
}

for(i=1;g[i][2]!=0;i++)
      continue;
t=i-1;
for(i=1;g[i][0]!=0;i++)
      continue;
s=i;

while(j<=n)
{
      y=j;
      if(t==0)
      {
            i=s;
            goto step85;
      }
      else  i=t;

      while(i>=1)
      {
            y=alpha(g[i][2],g[i][3],y);
```

```
           i=i-1;
      }

      if(s==0)   goto step86;
      else i=s;

step85: while (i>=1)
           {
                y=alpha(g[i][0],g[i][1],y);
                i=i-1;
           }

step86: arr[j][8]=arr[arr[y][3]][0];
           j=j+1;
}

order: arr[0][9]=arr[0][0];
         arr[1][9]=arr[0][8];
for(j=1;j<=n;j++)
{
      for(i=0;arr[j][9]!=arr[i][0];i++)
           continue;
      arr[j+1][9]=arr[i][8];
      if(arr[j+1][9]==arr[0][0])
         break;
      continue;
}

if (M==2)
{
      for(j=2;j<=n;j++)
      {
           for(i=1;i<=n;i++)
           {
                if(arr[i][0]==arr[j][9])   break;
```

```
            else continue;
        }
        s=arr[i][1];
        for(m=1;m<=n;m++)
        {
            if(arr[m][0]==arr[j-1][9]) break;
            else continue;
        }
        t=arr[m][2];
        arr[j][5]=max(s,t);
    }

    for(i=1;i<=n;i++)
    {
        if(arr[i][0]==arr[1][9])    break;
        else continue;
    }
    s=arr[i][1];
    for(i=1;i<=n;i++)
    {
        if(arr[i][0]==arr[n][9])    break;
        else continue;
    }
    t=arr[i][2];
    arr[1][5]=s+t;
}

if(M==3)
{
    for(j=1;j<=N;j++)
    {
        for(i=1;Y[arr[j][9]][i]!=0;i++)
        {
            if(i==1)    {
                                    for(f=1;f<=3;f++)
```

```
                      cm[f][1]=arr_job3
                            [arr[j][9]][f];
                }
        else    {
                    for(f=1;f<=3;f++)
                        cm[f][1]=0;
                }
        for(f=1;f<=3;f++)
        {

cm[f][2]=arr_job3[arr[j][9]][f+3]*Y[arr[j][9]][i];
            if(f==1)
cm[f][0]=cm[f][3]+cm[f][1]+cm[f][2];
            else    cm[f][0]=cm[f][3]+cm[f][1];
        }
        if(cm[1][0]>=cm[2][0])
        {
            if((cm[2][0]+cm[2][2])>=cm[3][0])
            {
                cm[1][3]=cm[1][0];
                cm[2][3]=cm[1][3]+cm[2][2];
                cm[3][3]=cm[2][3]+cm[3][2];
            }
            else
            {
                if((cm[3][0]-cm[1][0])>=cm[2][2])
                {
                    cm[3][3]=cm[3][0]+cm[3][2];
                    cm[2][3]=cm[3][0];
                    cm[1][3]=cm[3][0]-cm[2][2];
                }
                else
                {
                    cm[1][3]=cm[1][0];
                    cm[2][3]=cm[1][3]+cm[2][2];
```

```
                        cm[3][3]=cm[2][3]+cm[3][2];
                    }
                }
            }
            else
            {
                if((cm[2][0]+cm[2][2])>=cm[3][0])
                {
                    cm[2][3]=cm[2][0]+cm[2][2];
                    cm[3][3]=cm[2][3]+cm[3][2];
                    cm[1][3]=cm[2][0];
                }
                else
                {
                    cm[3][3]=cm[3][0]+cm[3][2];
                    cm[2][3]=cm[3][0];
                    cm[1][3]=cm[2][3]-cm[2][2];
                }
            }
        }
    }
}

return(*arr);
}


int *sim_anl(double TC0)

{int mu[job_size]={0},mu1[job_size]={0},mu2[job_size]={0};
int v,i,j,k,t=0,tmp;
int cyc=0,plt=0,qut=0;
double r=0,pro,delta,x,TC1,TC2;
int *p;
```

```
TC1=TC0;
for(i=1;i<=N;i++)
      mu1[i]=arr_job[i][6];

p_T=arr_T;
p_n=arr_n;
p_TC=arr_TC;

tmp=tmp_h;

while(tmp>tmp_l&&tmp<=tmp_h)
{
      v=1;
reproduce: srand(time(NULL));
            j=(1+rand()%N);

            while(v<=V)
            {
                  printf("\ntmp=%d\nv=%d\t",tmp,v);
                  printf("j=%d\t",j);

                  for(k=1;k<=N;k++)
                  {
                        if(mu1[j]>1)
                        {
                              if(k==j)    mu2[k]=mu1[k]-1;
                              else        mu2[k]=mu1[k];
                        }

                        else  {v=v+1;goto reproduce;}

                  }

                  srand(seed);
                  r=((double)rand()/(double)(RAND_MAX+1));
```

```
mu2[0]=0;
for(i=1;i<=N;i++)
{
      mu2[0]=mu2[0]+mu2[i];
      printf("%d,",mu2[i]);
}

p=MSP(mu2);

cyc=0;
plt=0;
qut=0;

for(i=1;i<=N;i++)
{
      cyc=cyc+*(p+(i*13+5))+*(p+(i*13+10));
      plt=plt+*(p+(i*13+11));
      qut=qut+*(p+(i*13+12));
}

if(M==2)    cyc=cyc;
if(M==3)    cyc=cm[3][3];

TC2=cyc*qut*Ch+plt*Cmh;

printf("TC1=%f TC2=%f\t",TC1,TC2);

delta=TC2-TC1;
x=delta/tmp;
pro=1/exp(x);
if(delta<=0)     {
                 TC1=TC2;
                 *p_TC=TC1;
                 p_TC=p_TC+1;
```

```
                                 for(i=1;i<=N;i++)
                                      mu1[i]=mu2[i];
                                 *p_n=mu2[0];
                                 p_n=p_n+1;
                                 j=j;
                                 t=tmp;
                                 *p_T=t;p_T=p_T+1;
                            }

                # else  {
                            if(r<pro)  {
                                            TC1=TC2;
                                            *p_TC=TC1;
                                            p_TC=p_TC+1;
                                            for(i=1;i<=N;i++)
                                                  mu1[i]=mu2[i];
                                            *p_n=mu2[0];
                                            p_n=p_n+1;
                                            j=j;
                                            t=tmp;
                                            *p_T=t;p_T=p_T+1;
                                       }
                            else  p=MSP(mu1);
                       }
           printf("random=%f",r);
           v=v+1;
           }
tmp=tmp-col_spd;
}
printf("\ntemperature=%d\n",t);
return(p);
}
```