CONTENT

LIST OF FIGURES

LIST OF TABLES

Chapter 1

General Introduction

In modern societies, all walks of life bear witness to progressively fiercer competition. As a result, extensive management mode cannot satisfy the demands posed by the competition anymore. Therefore, the question of how we are able to be more effective and take advantage of resources has become a focus that has captured the attention of all businesses and is becoming one of the core components for modern enterprises and administrations. A scheduling problem is solved by working out an appropriate plan for a set of tasks to be performed over time on a set of scarce resources to achieve one or several goals. Within the task-performing period, a certain power over resources needs to be consumed. However, the number of resources that a person (enterprise, technology) can use is finite. The ultimate goal of solving a scheduling problem is to, on the basis of achieving one or several objectives, ensure (or realize) the highest utilization rate of the resources as much as possible. Generally speaking, a scheduling problem may, according to different processing demands, fall into three categories: A Single Machine Model, Parallel Machines Model and Multi-Operation Model. In most of the articles involving scheduling problems, the problem of serial workshops (flow-shop) has been one of the first to be studied by the early fifties. Johnson, in 1954, was the first to study the problem of flow-shop with two machines [Johnson, 1954]. Then, appeared later studies other multi-operation models such as the job shop problem and the open shop problem (or flexible, depending on the design of the workshop). It has been found, at the early stage of scheduling theory, that there is a huge gap between research findings and practical production problems. The most important problem is the neglect of the inevitable limitations on the practical producing process such as the period between the finishing time of one operation of performing a task and the beginning time of its next operation, denoted as time delays, time lags, communication time delays, or transportation times, depending on the context. Time delays may take different forms in diverse industries. For instance, time delays may refer to transportation times to move a job from one place to another or to the time of heating or cooling a job before another process takes place.

Basically, there are two ways of approaching the resolution of scheduling

algorithms: the exact approach and the approximation approach. The latter approach can be divided further into heuristic algorithms and meta-heuristic algorithms.

This dissertation is mainly concerned with study of minimizing the overall completion time, also known as the makespan or the schedule length, on two-machine open shop problems with time delays considerations.

In addition to the introductory chapter, this thesis contains four chapters and a conclusion. The second chapter introduces some basic definitions, concepts and scheduling models such as the Gantt chart, time delays, and details on appropriate algorithms to solve scheduling problems.

The third chapter first considers the two-machine open shop problem without time delays which is solved by two exact algorithms: the Gonzalez (GS) algorithm and the Pinedo and Schrage (LAPT) algorithm. For implementation reasons, we proposed a simple way of stating the latter algorithm, along with a proof of its optimality. We also conducted a simulation study to compare the performance of these two algorithms.

The fourth chapter is about the two-machine open shop with time delay considerations. This chapter is divided into two parts. Part 1 mainly presents heuristic algorithms for some special cases. The performance evaluation of the different heuristic algorithms involves the analytic and the empirical approaches. Part 2 introduces meta-heuristic algorithms for the general open shop problem. The strategy of the meta-heuristic algorithms includes stopping criteria, internal structure, and hybrid algorithms. For the stopping criteria, the algorithms are interrupted by some special conditions, such as the makespan is equal to a lower bound. We propose and prove several lower bounds for the problem under study, and apply them to our meta-heuristic algorithms. For the internal structure, the development of a tabu search algorithm and a simulated annealing algorithm is discussed with the addition of intensification and diversification procedures. For the hybrid algorithm that we propose, we combine the advantages of the two algorithms.

Finally, a conclusion is drawn with a discussion on the present work and certain avenues of investigation for further research.

Chapter 2

Scheduling Problems

2.1 Introduction

Scheduling theory is about building solutions that assign starting and finishing times of tasks to scarce resources in order to minimize one or several goals. Resources could be central processing units (CPU) in computers, machines in workshop factory, runways in airports, etc. A task is a basic entity which is scheduled over the resources such as the execution of a program, the process of an aircraft taking off and landing, the process production, etc. The various tasks are characterized by a degree of priority and execution times. The goal correspond to performance measures to evaluate the quality of solutions such as: minimizing the maximum completion time (known as the schedule length, overall completion time or the makespan), minimizing waiting times, etc. A schedule is then built according to one or several of these objectives. For instance, an optimal schedule is needed in order to reduce flight delays. In that sense, some basic elements need to be known, for instance the number of runways, departure times, arrival times, etc. However, it has to be noted that these information are subject to changes at any time. For example, the runway is occupied by another aircraft or vehicle. Bad weather or other unpredictable factors may lead to flight delays. So we must keep abreast of the latest news and take a new scheduling scheme.

2.2. Scheduling models

The classical scheduling problem has been considered by a constraint: if a job includes x operations, then we assume that every operation is performed by a distinct machine at a given moment. Usually, production scheduling can be classified into the following three models:

1. Single machine model.
2. Parallel machines model.
3. Multi-operations and scheduling problems model
   - flow-shop,
   - open-shop,
   - job-shop.

Let us note that we implicitly assume, in all the above models, that, at any time, a job can only be processed by a most one machine, and a given machine can only process at most one job.

In a single machine model, a group of tasks are assigned (processed) by a single machine or resource. The tasks are arranged so that one or more performance measures may be optimized as pictured by Figure 2-1.

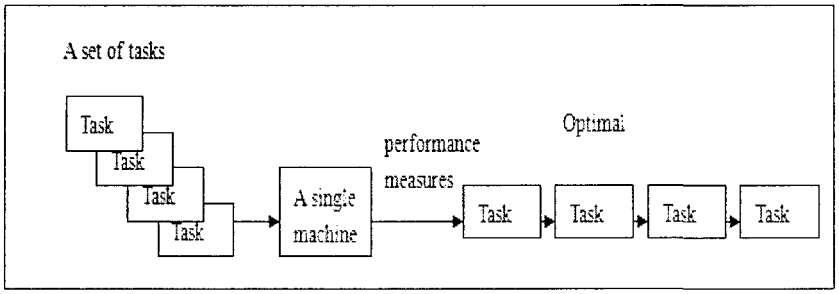


Figure 2-1: Single Machine Model

Along with the Industrial development, in the most complex environments, a single machine scheduling cannot meet the requirements of the production as in manufacturing industry, food processing industry, etc. In these industries, all task processes are the same, so we expand the production scale by adjusting the number of production lines. To schedule n independent tasks on m identical machines that operate in parallel is called the parallel machine scheduling problem, as pictured by Figure 2-2. In the case of a workshop assembly process, for instance, each task j is only allowed to be processed on a specified subset of machines. According to the processing speeds, parallel machines can be classified into identical parallel machines (the speeds of machines are the same), uniform machines (the speeds of the machines are proportionate), and unrelated machines (the speeds are not related and the processing times depend only on the jobs), respectively.

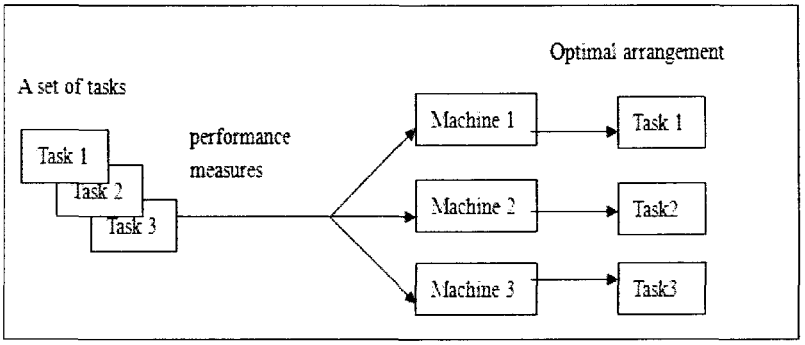


Figure 2-2: Parallel Machine Model

In the context of transportation, computation and logistics management, every task is independent and operations (routes, courses, etc) of every task are different. These problem common features are that every task must be processed on several different machines. The mode is called multi-jobs and non-preemptive scheduling problem model, which includes three models: flow-shop, open-shop and job-shop. In flow-shop, all jobs will be processed on all the machines in the same order such as Figure: 2-3. In this figure, we have a set of three jobs and a set of three machines. And we can notice that each job has followed a same order on the three machines.

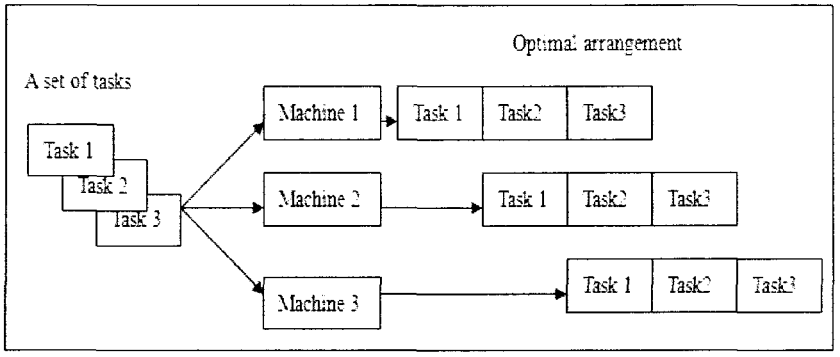


Figure 2-3: Flow Shop Model

There exists within this model a special case which occurs in its own right in many applications. In this flow-shop model, jobs do not overtake other. This means that, if a job precedes another job on one machine, then this remains the same for the rest of the machines. This models scheduling processes with queuing restrictions. This model is known as the permutation flow-shop.

In the open-shop model, a set of jobs are processed on several machines. Each job has to be processed on each machine and does not have a fixed route. In fact, this route should be built when constructing a solution. In Figure 2-4, we have a set of three jobs and a set of three machines. We can notice that each job has followed a different sequence on the three machines.

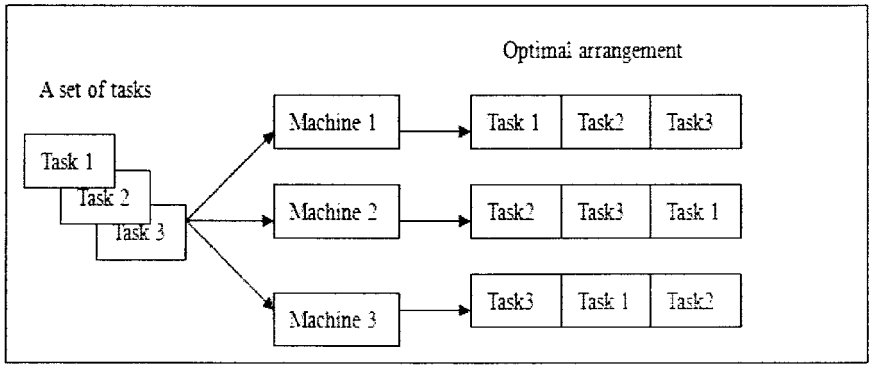


Figure 2-4: Open Shop Model

In job-shop model, a set of n jobs are processed on m machines. Each job consists of a fixed processing order though the machines. In Figure 2-5, we have four jobs and six machines. Each job follows a fixed route on the six machines, which is highlighted by a color. For example, a job may pass through machine 1, then machine 4, and finally machine 5.

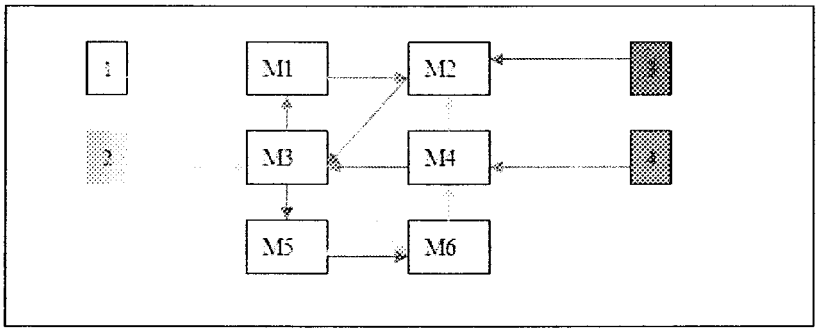


Figure 2-5: Job Shop Model

Let us conclude this section by mentioning the fact some of the above models can be mixed together to form new models, for example one can assume that the set of job can be divided into a flow shop type and an open shop type. We can also that in a job shop model each stage can be formed by a set of parallel machines.

2.3 Basic Definitions

We present in this section some basic definitions, used in scheduling theory that we will need for the rest of this dissertation.

Definition 2-3-1: *A schedule may be viewed as a way of assigning, over time, tasks to resources. It is possible to determine planning and design for a problem by means of calculation in advance.*

A resource is a basic device where jobs are scheduled/processed/assigned [Blazewicz *et al.*, 2004], which is known as CPUs, memory, storage space; workshop factory. Each resource has a limited capacity, a speed and a load. The limited capacity is a number of CPUs, amount of memory, the size of storage space and so on.

The speed is defined as how quickly a job can be processed on a resource. The load measures how much of the capacity of a resource is used over a time interval. If a resource can be used by different tasks, it is called renewable; otherwise it is called nonrenewable.

Definition 2-3-2 [Fibich *et al.*, 2005]

A machine is a set of cumulative resources (CPUs, memory, storage space, workshop factory) with limited capacities. The characteristics of a machine include its capacity, load, speed and location, which are described by descriptors of the machine.

Defintion 2-3-3 [Fibich *et al*, 2005]

A job (task, activity) is a basic entity which is scheduled on the resources. A job has specific requirements on the amount and type of resources (including machines) or required time intervals on these resources.

A detailed description of these conditions is quite difficult to undertake. Graham *et al.* [1979] introduced a three-field notation to describe scheduling problems.

2.4 Three-Field Notation

Scheduling problems are often classified according to Graham's standard three-field $\alpha \mid \beta \mid \gamma$ notation [Graham *et al.*, 1979], where α describes the machine environment, β provides details of characteristics or restrictive requirements, and γ stands for the criterion performance measure.

1. **The α field:** The basic machine environments include single machine problems, various types of parallel machine problems, flow shops, job shops, open shops, mixed shops and multiprocessor task systems problems.

A single machine is denoted by "1" in the α field to indicate that the scheduling problem is solved by one machine. If jobs are scheduled on m identical machines operating in parallel, then this denoted by P_m; with uniform machines where the speed are proportionate (Q_m), unrelated machines where there no special relationship between the speeds of the machines (R_m), and flow shop, job shop, open shop with m machines are denoted by F_m, J_m, O_m, respectively.

2. The β field: It provides details of characteristics or restrictive requirements. Details of characteristics and restrictive requirements are further described and qualified for the current resources and the environment.

- Processing Time: the time by which taken to complete a prescribed procedure.

-Time delay (time lag): the time that must elapse between the completion of an operation and the beginning of the next operation of the same job.

- Breakdown (brkdwn)**:** The breakdown indicates that the machines (resources) can break down thus not available for processing.

3. The γ field: It denotes performance measures for evaluating the quality of solutions. The performance measures define the quality of the obtained schedule based on input parameters of particular tasks and, usually, on their completion times. They take into account all the tasks existing in the system in order to estimate its

behavior from a global point of view. The parameters that are usually associated with the set of jobs are as follows:

- Arrival time: Time at which a given job becomes available for processing.
- Due date: Time by which a given job should be completed.
- Weight: A positive value associated with a given job to denote its relative importance or priority.

Those performances are usually given as a function of the completion times of the jobs.

Definition 2-4-1

The completion time C_j *of a job j denotes the time at which the last operation of this job is completed.*

A list of objective criteria, commonly used in scheduling theory, could include the following:

- **Makespan**: The completion time of the latest job also is known as the time difference between the start and finish of a sequence of jobs that is denoted as $C_{max} = \max\ C_i$. So the goal is to minimize C_{max}.
- **Total weighted mean completion time:** The goal is to find a feasible schedule of the n jobs that minimizes $\sum w_j C_j$.
- **Maximum lateness:** The goal is to find a schedule which minimizes $L_{max} = \max(\ C_j - d_j)$.
- **Tardiness:** A job is tardy if $C_j > d_j$. Tardiness is defined as $T_j = \max\{\ 0, L_j\}$.

  If for some schedule the maximum lateness is not positive then the maximum tardiness is 0 which is obviously optimal. Otherwise the maximum lateness is positive, and so the maximum tardiness is equal to the maximum lateness.

- **Number of tardy jobs:** we denote the number of tardy jobs by U_j .Let $U_j = 1$ if $C_j > d_j$, otherwise. Then, the goal is to seek a schedule of the jobs so as to minimize $\sum U_j$.

2.5 Gantt Chart

Gantt chart, a useful tool for analyzing and planning more complex projects, was designed by Gantt [1916]. Gantt diagram is a type of bar chart that illustrates a graphical representation of the start and completion time of the jobs of a project, which includes two perpendicular axes. While the vertical axis represents the number of machines, its horizontal counterpart represents a time scale.

Example 2-1

Let $m = 2$ and $n = 3$ .Tables 2.1 summarizes the processing times of the jobs.

	J_1	J_2	J_3
M_1	2	3	4
M_2	3	2	2

Table 2-1: Instance with m = 2 and n = 3

Figure 2-6 shows the Gantt chart associated with a scheduling solution. The Gantt chart provides a fast, intuitive way to monitor the scheduling progress and to determine where troubles are in a given solution. For this diagram, we can observe the status of each task (for e.g. start time, completion time, etc), and make time adjustments to change the processing sequence in order to obtain an optimal sequence.

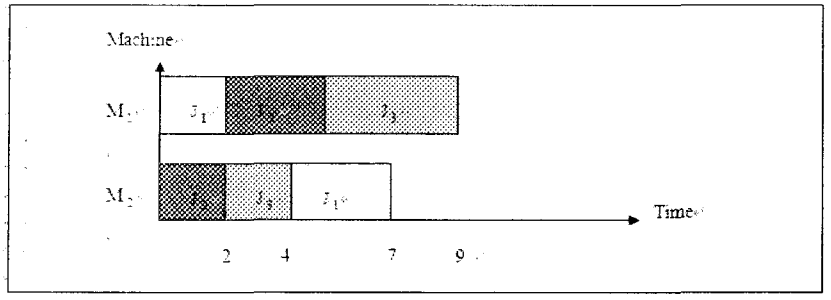


Figure 2-6: Gantt Chart

The Gantt diagram visualizes the different orderings of jobs on machines and other information of a given solution. In what follows, the Gantt chart is used to picture the effects of time delays and idle time on the quality of a schedule.

2.6 Scheduling with time delays

In scheduling theory, some objective factors cannot be evaded in practice, such as the time spent when transmitting a work-piece after its completion on one machine to another to process; the time spent in clearing off the runway between an aircraft's landing and take-off which are controlled by the airport schedule, etc. In some cases, they have to be considered if we want to build a valid solution.

Time delays can be classified into minimal, maximal and exact time delays. In some cases, a time delay of a minimum length must elapse before handling the considered product, like when transporting products form one center to another. This is termed as the minimum time delay situation. However, if the interval between two operations is too long, it is very likely to lead to the scrap or quality decline of the processed products. Take food processing industry, where too long time of storing food makes food decay. Faced with these circumstances, an upper bound and lower bound are set on time delays, which are termed as minimum time delays and maximum time delays. In some other cases, the waiting time between the completion of an operation and the beginning of the next operation of the same job must be fixed. This is termed as the exact time delay situation. Let us note that, in general, if there are not special indications, time delays refer to the minimum time delays case. Let us consider the following instance with $m = 2$ machines and n = 4 jobs.

Example 2-2

	J_1	J_2	J_3	J_4
M_1	2	2	3	3
M_2	2	2	2	8

Table 2-2: An Instance with m = 2 and n = 4

An optimal solution for this problem is as illustrated by Figure 2-7.

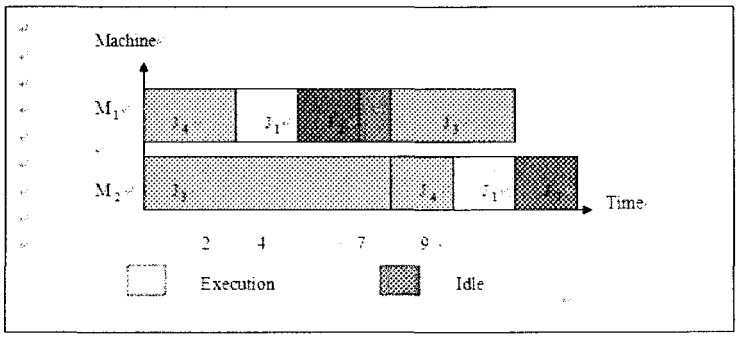


Figure 2-7: Optimal schedule without time delays

Let us now consider the same instance, but with time delays considerations given as in Example 2-3.

Example 2-3

Job	J_1	J_2	J_3	J_4
Time delay l_j	2	2	1	3

Table 2-3: Time Delays for Example 2-2

The optimal solution becomes as in Figure 2-8.

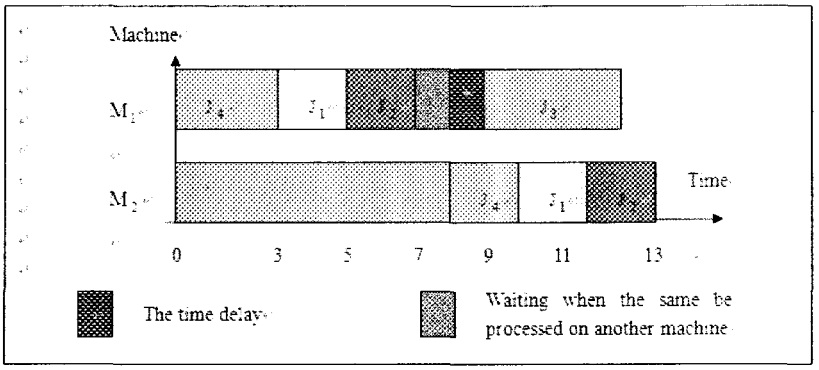


Figure 2-8: Optimal Schedule with Time Delays

2.7 A Brief introduction to the complexity theory

Computational complexity theory, an active field in theoretical computer science and mathematics, deals with the resources required during computation to

solve a given problem. To put it simply, the aim of the complexity theory is to understand the intrinsic difficulty of solving a given problem.

Definition 2-7-1

The complexity of an algorithm is the "cost" used by the algorithm to solve a given problem. The cost can be measured by terms of executed instructions (the amount of work the algorithm does), running time, memory consumption or something similar.

Among all "costs", we focus our attention on the running time of an algorithm. The best-case, worst-case, and average-case complexities refer to three different ways of measuring the time complexity as a function of the input size. We are more interested in understanding the upper and lower bounds on the minimum amount of time that are required by the most efficient algorithm solving a given problem. Therefore, the time complexity of an algorithm usually refers to its worst-case time complexity, unless specified otherwise. The worst-case or average-case running time or memory usage of an algorithm is often expressed as a function of the length of its input using the big O notation. In typical usage, the formal definition is not used directly; rather the O notation for a function $T(n)$ is derived by the following simplification rules: If $T(n)$ includes several factors, only the one with the largest growth rate is kept, and all other factors and any constants are omitted. For instance, if the execution time of an algorithm $T(n) = 6n^3 + 3^2 - 2n + 5$, then the worst-case complexity is $T(n) = O(n^3)$.

Definition 2-7-2

An algorithm is said to be polynomial time if its running time is upper bounded by a polynomial in the size of the input for the algorithm.

Definition 2-7-3

If a problem can be solved in polynomial time, it is called tractable; otherwise it is called intractable.

Definition 2-7-4

A decision problem is a type of computational problem whose answer is yes or no.

Decision problems are the core objectives of study in computational complexity theory. In this section, our discussion is hence restricted to decision problems.

- **P and NP classes**

Definition 2-7-5

The P-class consists of all problems that can be solved in polynomial time as a function of the size of their input.

Definition 2-7-6

If a decision problem can be solved in polynomial time, then it belongs to the P-class.

Definition 2-7-7

NP represents the class of decision problems which can be solved in polynomial time by a non-deterministic model of computation.

From the above definitions, we can easily derive the relationship: P is included in NP, but if we do not know whether P = NP.

In 1971, Stephen Cook published a paper "The complexity of theorem proving procedures", in which he further proposed the concept of NP-completeness.

- **NP-hard and NP-complete classes**

Definition 2-7-8

A decision problem L is NP-complete if it is in NP and if every other problem in NP is reducible to it.

The term "reducible" means that there exists a polynomial-time algorithm to transform an instance $l \in L$ into an instance $c \in C$ such that the answer to c is YES if, and only if, the answer to l is YES.

Definition 2-7-9

NP-hardness (non-deterministic polynomial-time hardness), in computational complexity theory, refers to a class of problems that are, informally, "at least as hard as the hardest problems in NP". A problem H is NP-hard if and only if there is an NP-complete problem L that is polynomially reducible in time to H.

NP-complete problems are the most challenging in the NP class. A problem is NP-hard if it is at least as hard as any problem in class NP. If there is a polynomial algorithm for any NP-hard problem, then there are polynomial algorithms for all problems in NP, and hence P = NP. Figure 2-9 shows the relationship between P, NP, NP complete and NP-hard classes.

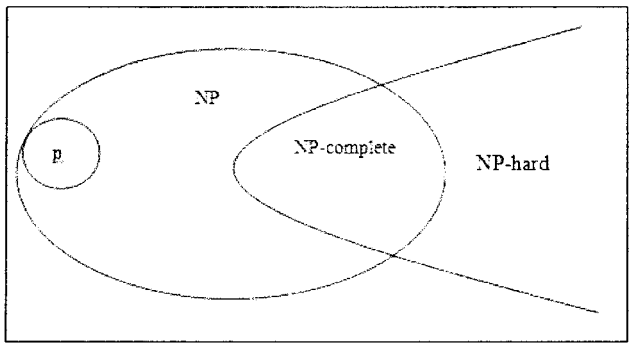


Figure 2-7: Relationship between P, NP, NP-Complete and NP-Hard classes

2.8 Analysis of Scheduling Problems

In this section, we describe the way to analyze or deal with deterministic scheduling problems. In this dissertation, we have limited our study to "deterministic" scheduling problems. In other words, all parameters are assumed to be known and fixed in advance.

Generally speaking, the idea of analyzing deterministic scheduling problems is to find the appropriate solution according to their respective characteristics. In most cases, the time "cost" is limited, so that only low order polynomial time algorithms may be used. Thereby, understanding the complexity of algorithm is very important and is also the basis for further analysis. In section 2.6, we have introduced some categories of the class complexity (P class, NP-complete class, and NP-hard class). The complexity of problems is the basis for further analyzing the problem solving process. If the problem is in the class P, then a polynomial time algorithm must already have been found. Its usefulness depends on the order of its worst-case complexity function and on the particular application. Except the worst-case complexity analysis, probabilistic analysis of algorithms is a common approach to estimate the computational complexity of an algorithm..

As illustrated in Figure 2-10, if a problem is NP-complete or NP-hard, then either of following approaches is used in order to solve it: exact method, relaxation, approximation method, and pseudo-polynomial method.

Figure 2-10 shows a schematic view to analyze scheduling problems. These methods are further explained in the following sections.

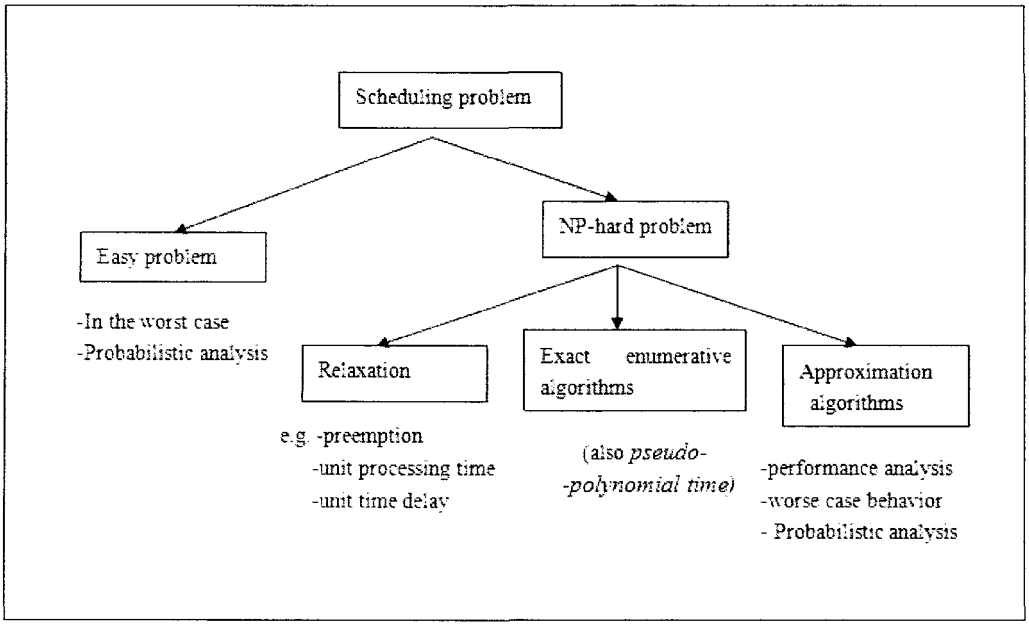


Figure 2-10: Analysis of Scheduling Problems

2.8.1 Exact methods

An exact algorithm is an algorithm that can obtain an optimal solution to a given problem. This class of algorithms is divided into polynomial time algorithms and enumerative algorithms. For some special structured problems, we may find polynomial time algorithms to solve them. For example, when the time delays are ignored, an optimal solution to two-machine open shop problems can be obtained in polynomial time by a simple algorithm. However, for $m \geq 3$, the open shop scheduling problem becomes NP-complete [Pinedo, 1995].

Let us observe that most scheduling problems are NP-hard problems, which means that the only algorithms we have at hand to solve these problems need exponential running time. Such algorithms are mainly enumerative algorithms, linear programming, and dynamic programming.

2.8.2 Relaxation

We may try to relax some constraints imposed on the original problem in order to reduce the difficulty of its resolution. The solution may be equal to or more closer

to the optimal solution of problem. Within the scheduling context, these relaxations may include:

- allowing preemptions,
- assuming unit-time operations,
- assuming simpler precedence graphs,
- etc.

2.8.3 Pseudo-Polynomial Algorithms

Although all NP-hard problems are computationally hard, some of them may be solved efficiently in practice. This is because the time complexity of those algorithms mainly depends on the input length and the maximal number. In practice, the maximal number is not large, and is usually bounded by a constant; this leaves us with a polynomial algorithm. This is where the name of *pseudo-polynomial* algorithm comes from. It does not mean the algorithm really is a polynomial algorithm.

2.8.4 Approximation Algorithms

It is time consuming (and thus difficult) to find optimal solutions to NP-hard problems. Therefore, the approximation approach becomes almost an inevitable choice. The approximation algorithm generally falls into heuristic algorithm and meta-heuristic algorithm. These approaches are described in details in Section 2.9.

Generally speaking, heuristic algorithms are used to solve special problems, but the improvement space of a heuristic algorithm is limited, so researches often try to find a new and better algorithm to solve it. The same problem is often able to be solved by several different heuristic algorithms; moreover we have difficulties in intuitively judging which one is better. For some cases, the results of some heuristic algorithms are better, but for other cases, some contrary conclusions may be drawn. Thereby, some uniform measurement standards and calculating methods are indispensable. We commonly evaluate the performances of different heuristic algorithms by using two methods: the *Analytic Approach* and the *Empirical Approach.*

2.8.4.1 Analytic Approach

The analytic approach is about finding the distance between an optimal solution and the solution produced by a heuristic algorithm. The commonly used methods include the worst-case analysis and probabilistic analysis.

- **Worst-case Analysis**

The quality of a given heuristic is measured by the maximal distance between the optimal solution and the solution produced by the heuristic under study.

Usually, the maximal distance is measured by the relative error between the two solutions. If S_H and S^* denote the makespan produced by heuristic H and an optimal solution, respectively, then the goal is to find a ratio performance guarantee (or worst case bound) ρ such that the following relationship holds:

$$C_{\max}(S_H)/C_{\max}(S^*) \leq \rho,$$

where $C_{\max}(S)$ denotes the makespan of schedule S.

- **Probabilistic Analysis**

Probabilistic analysis starts from an assumption about a probabilistic distribution of the set of all possible inputs. This model usually assumes that all parameter values are realizations of independent probabilistic variables of the same distribution function. This assumption is then used to design an efficient algorithm or to derive the complexity of a known algorithm. Then for an instance I^n of the considered optimization problem (n being a number of generated parameters) a probabilistic analysis is performed. The result is an asymptotic value OPT (I^n) expressed in terms of problem parameters. Then, algorithm A is probabilistically evaluated by comparing solution values (A (I^n) being an independent probabilistic variables) with OPT (I^n) [Rin87]. The two evaluation criteria used are absolute error and relative error. The absolute error is defined as the difference between the approximate and optimal

solution values

$$a_n = A(I^n) - OPT(I^n).$$

The relative error is defined as the ratio of the absolute error and the optimal solution value

$$b_n = \frac{A(I_n) - OPT(I_n)}{OPT(I_N)}.$$

2.8.4.2 Experimental Approach

The experimental approach is based running the corresponding algorithm on a large number of effective data to evaluate its performance. This approach is mainly used to compare multiple heuristic algorithms.

Let us note that the analytic and the experimental approaches are complementary: the former proves strong theoretical foundations under some hypotheses, and the latter shows the practical performance tendency of the considered algorithm.

In Section 4.3.2., we present a complete example to compare two heuristic algorithms.

2.9 Description of an Exact and a Heuristic Approach

In the above section, we have mentioned different approaches to solve a scheduling problem. In this section, we will give more details on some of these methods.

2.9.1 Exact Algorithms

In addition to special cases, enumeration (brute force search) algorithm is a very general problem-solving technique for obtaining exact solutions that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement. Enumeration algorithm mainly includes branch and bound, dynamic programming and so on. In what follows, we present in more details the algorithm of branch and bound (B&B for short), which is used extensively in practice.

- **Branch and Bound**

Branch and Bound (B&B) is the most commonly used enumeration algorithms for combinatorial optimization problem (NP-hard problems) to generate an optimal solution.

The efficiency of a branch and bound method is determined by the branching efficiency and pruning ability. The branching efficiency is determined by the branching strategy and searching strategy. The pruning ability is determined by the values of the upper bound, lower bound and the effect of dominance rule at hand. An upper bound corresponds to the value produced by an arbitrary schedule. A lower bound value corresponds to the smallest value that can be achieved by any solution, whereas a dominance rule states that any solution cannot be better than the one produced by the solution with a certain property.

The method of branch and bound was used for the first time by Danzig, Fulkerson, and Johnson [1995] to solve the problem of traveling salesman (TSP).

The idea of the method of branch and bound is to first confirm the upper and lower bounds of the goal values and then cut off some branches of the search tree while searching, to improve the efficiency of the search.

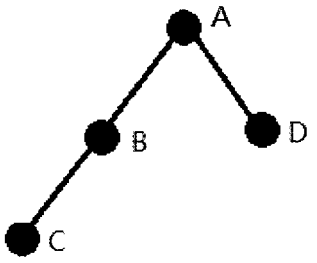
1. Bounding

A lower bound represents the smallest value that can be obtained by a feasible solution. As far as the method of branch and bound is concerned, if the lower bound of a given node, in a search tree, is not smaller than the known upper bound, a downward search from this node will not be needed. Therefore, if a superior upper bound can be produced, then many unnecessary listing calculations will be eliminated.

2. Search Strategy

The searching ways are divided into two categories, depth-first search and breadth-first search.

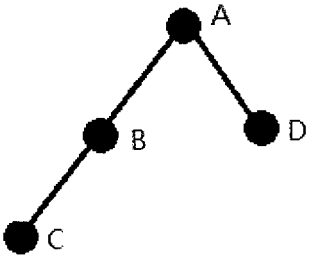
Depth-First-Search (DFS) starts at the root (selecting some node as the root in the graph case) and explores as far as possible along each branch before backtracking.



The nodes are visited in the order
A, B, C, D

Depth-First-Search (DFS)

Breadth-First-Search (BFS) begins at the root node and explores all the neighboring nodes. Then for each of those nearest nodes, it explores their unexplored neighbor nodes, and so on, until it finds the goal.



The nodes are visited in the order
A, B, D, C

Breadth-First-Search (DFS)

Branching

The operating principle of pruning is like a running maze. If we regard the searching process as a tree traversal, pruning literally means "cutting off" the "branches" that our needed solutions cannot reach, the dead ends in a maze, to reduce the searching time.

Of course, not all the branches can be cut off. However, more branches are pruned, the faster is the method. Pruning principle is as follows:

3.1. Correctness

As observed above, not all branches can be cut off. If the optimal solution is cut off, then the pruning does not make any sense. Thus, the precondition for pruning is ensuring that correct results will not be lost.

3.2. Efficiency

Therefore, it is equally important how to strike the balance between optimization and efficiency to lower the time complexity of program as much as possible. If a judgment for pruning produces a very good result, but it has taken much time to make the judgment and the comparison, with the result that there is no difference between the operations of the optimized program and the original one; it is more loss than gain.

2.9.2 Heuristic Algorithms

The heuristic approach (from the greek "*heuriskein*" meaning "to discover") is based on experience techniques that help in problem solving, learning and discovery. Heuristics are "rules of thumb", educated guesses, intuitive judgments or simply common sense rather than by following some pre-established formula.

The core idea of a constructive heuristic is to build step by step a solution to problem. In other words, each step of the algorithm is only to consider the next step according to a given rule. The priority rule provides specific strategies for the sequence in which jobs should be processed according to some rules such as Shortest Processing Time (SPT), Earliest Due Date (EDD), and so on.

Although an optimal solution to every combinatorial problem can be found, some of these would be impractically slow for NP-hard (NPC) problems, since it is unlikely that there can be efficient exact algorithms to solve these problems. However, for some special cases, heuristic (suboptimal) algorithms can find the optimal (or close to optimal) solution in reasonable time complexity. A worst case analysis is commonly used to study the performance of these algorithms.

- **Local Search Methods**

Local search can solve some problems that find a maximum solution among a number of candidate solutions (candidate solution is a member of a set of possible neighborhood solutions to a given problem). Neighborhood search is continuously searching in the neighborhood domain of the current solution.

Local search algorithm only searches the neighborhood domain of the best present solution (like the above example), and if the new domain does not have a value that is better than the present value, then the iterative process will be stopped. Local search algorithms are typically local optimal solution algorithm, which is simple and rapid, but the accuracy of results may be poor. A computational simulation is commonly used to study the performance of these algorithms.

2.9.3 Meta-heuristic Algorithms

Generally, the heuristic algorithms have good results that are used to solve specific objectives but not all. For example, the results of some heuristic algorithms may depend on the selection of an initial point; if the objective function and constraints have multiple or sharp peaks, the quality of the result may become unstable. The computational drawbacks of existing heuristic methods have forced researchers to improve it.

In a general framework optimization heuristics are also called meta-heuristics which can be considered as a general skeleton of an algorithm applicable to a wide range of problems.

The meta-heuristics algorithms are that they combine rules and randomness to imitate natural phenomena such as the genetic algorithm (GA) proposed by [Holland, 1975] (the evolutionary), tabu search proposed by [Glover, 1986] (animal behavior) and simulated annealing proposed by [Kirkpatrick et al, 1983] (the physical annealing process), etc. These meta-heuristics algorithms are theoretically convergent, that is if the computation time tends to infinity, it will be able to find the global optimum under certain conditions. However, these conditions are rarely verified in practice.

The meta-heuristics includes a new random initial solution (or the solution of a constructive heuristic) and a black-box procedure (iterative search). The common method used to analyze the performance of these algorithms is the computational simulation.

2.9.3.1 Simulated Annealing

The idea of simulated annealing (SA) is presented by [Metropolis, 1953] at first and applied on combinational optimization problem by [Kirkpatrick, 1983]. The basic starting point of SA is based on metal (solid) in annealing process that is a process for finding low energy states of physical substance that refers to a process when physical substances are raised to a high temperature and then gradually cooled until thermal equilibrium is reached. The process can be simulated by Monte Carlo method, which initially serves the function that it is applied to find the equilibrium configuration of a set of atoms at a given temperature ($R = \exp(E_i - E_j)/kT$, where E_i denotes the energy in i state; T denotes temperature; k is the cooling factor).

In 1983, Kirkpatrick first introduced the Metropolis rule to combinational optimization problems. This algorithm process is called Simulated Annealing algorithm. In combinatorial optimization problems, an initial value i and its objective function $f(i)$ correspond separately to a state i and its energy E_i and use two control parameter t_0, t_f to simulate initial temperature and terminated temperature. Repeat the process: "create—judge—accept/abandon" until some stopping criterion is achieved. The basic annealing process for open shop problems may be as follows.

```
i=random sequence;
f*= f(i);
Initial temperature T*= T0; Terminated temperature Tf*= Tf;
Cooling factor α*=α;
Repeat
    Generate a random neighbor f(j) from f(i);
    Δf=f(j)-f(i);
    If Δf<=0 then accept the new sequence (f*= f(j),f(i)=f(j))
    Else
        If Δf>0 then
            get a random number h∈(0,1);
            If h > exp(-Δf/T)
            Then accept the new sequence (f*= f(j) ,f(i)=f(j))
    If T>= Tf then T=α T;
until (termination-condition);
```

Algorithm 2-1: Basic Simulated Annealing Process

2.9.3.2 Genetic Algorithms

Genetic Algorithms (GA) are adaptive heuristic search algorithm premised on the evolutionary ideas of natural selection and genetic that was invented by John Holland in the 1960s, and was developed with his students and colleagues at the University of Michigan, in the 1970s. Genetic algorithms are categorized as global search heuristics. The basic concept of GA is designed to simulate processes in natural system necessary for evolution, specifically those that follow the principles first laid down by Charles Darwin of survival of the fittest. As such, they represent an intelligent exploitation of a random search within a defined search space to solve a problem.

Genetic algorithms are implemented in a computer simulation in which a population of abstract representations (called chromosomes or the genotype of the genome) of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem evolves toward better solutions. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals and happens in generations. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), and modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. If the algorithm has terminated due to a maximum number of generations, a satisfactory solution may or may not have been reached.

A basic genetic algorithm comprises three genetic operators: Selection, Crossover, and Mutation.

a. Selection

This operator selects the chromosome in the population for reproduction. Based on the survival-of-the-fittest strategy, the more fit the chromosome, the bigger

chance to be selected for reproduction. The most commonly used strategy to select pairs of individuals is the method of roulette-wheel selection, in which every string is assigned a slot in a simulated wheel sized in proportion to the string's relative fitness. This ensures that highly fit strings have a greater probability to be selected to form the next generation through crossover and mutation.

b. Crossover

Crossover is a genetic operator that combines two chromosomes to produce one or two new chromosomes. The idea behind crossover is that the new chromosome may be better than both of the parents if it takes the best characteristics from each of the parents. Crossover occurs during evolution according to a user-definable crossover probability includes the following types of crossover.

b.1. Single Point Crossover

After randomly choosing a crossover point on two parent chromosomes and inter changing them at the crossover point, two offspring are produced. End each of them inherits the genes of its parent before the crossover point and the ones of the other parent after the crossover point. Consider the following two parents selected for a crossover.

Parent 1: 11111|000
Parent 2: 00000|111
After interchanging the parent chromosomes at the crossover point, the following offspring are produced:
Offspring1: 11111|111
Offspring2: 00000|000
The "|" symbol indicates the randomly chosen crossover point.

Figure 2-8: Single Point Crossover

b.2. Two Point Crossover

After randomly choosing two crossover points on two parent chromosomes and interchanging them at the crossover points, two offspring are produced. Each of them

inherits from its parent the genes except between the two points, and from the other parent the genes between the two points. Consider the following two parents selected for two crossovers.

Parent 1: 111|11|000
Parent 2: 000|00|111
After interchanging the parent chromosomes between the crossover points, the following offspring are produced:
Offspring 1: 111|00|000
Offspring 2: 000|11|111
The "|" symbol indicates the randomly chosen crossover point.

Figure 2-9: Two Point Crossover

b.3. Uniform Crossover

Bits are randomly copied from the first or the second parent. Consider the following two parents selected for a crossover.

Parent 1: 12345676
Parent 2: 34214575
If the mixing ratio is 0.5, approximately half of the genes in the offspring will come from parent 1 and the other half will come from parent 2. Below is a possible set of offspring after uniform crossover:
Offspring 1: 1435576
Offspring 2: 3224675

Figure 2-10: Uniform Crossover

c. Mutation

On certain odds, there is the possibility for every gene in the sequence to change. This is a method that can avoid the minimum in local. In scheduling, this change may be randomly exchanging the processing orders of two tasks. A simple genetic algorithm for the open shop problem is as described by Algorithm 2-2.

Generate random population of n chromosomes (suitable solutions for the problem)
Evaluate the fitness f(x) of each chromosome x in the population
Repeat:
    Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected).
    Setup a crossover probability; two parent chromosomes form two new offspring (children) by a certain crossover point. If no crossover is performed, an offspring is the exact copy of parents.
    With a mutation probability, mutate two new offspring at each locus (position in chromosome).
    Place new offspring in the new population to replace the old generation
If the end condition is satisfied, stop else go repeat;

Algorithm 2-2: The Basic Genetic Algorithm Process

2.9.3.3 Tabu search method

The basic principle of tabu search (TS) method is based on classical local search method (LS) improvement techniques and to overcome local optimal by crossing boundaries of feasibility. The essential feature of a TS method includes allowing non-improving moves, the systematic use of *memory* and relevant restrictions for improving the efficiency of the exploration process. Tabu search was presented by [Glover, 1986, 1989, 1990]. Let us note that the basic ideas of the method have also been advanced by [Hansen, 1986].

In order to avoid local convergence, the idea that "inferior solution" can be accepted to some extent is derived. The important objective of the method reasonably increases the scope of neighborhood domain and avoids searching as far as possible in the found neighborhood. The components of TS include tabu list (memory length), tabu length and candidates swap and aspiration criteria. Tabu list is a short-term memory which contains the solutions that have been visited in the recent past (candidate swaps). In the tabu list, certain moves are prohibited to be visited unless the move is "best so far". The time of the move is decided by the tabu length. The basic tabu search procedure for open shop might be as follows.

```
Obtain a random initial sequences f(i)
Clear up the Tabu list;
Repeat
    Select a new minimum sequence f(j) in the neighborhood of f(i);
    If f(j)<best_to_far then
    begin
    f(i)=f(j);
    let f(j) take place of the oldest sequence in the Tabu list;
    best_to_far:=f(j);
    end else
            begin
            if f(j) is not in the Tabu list then
               f(i)=f(j);
               f(j) take place of the oldest sequence in the Tabu list;
            end;
until (termination-condition);
```

Algorithm 2-3: The Basic Tabu Search Process

In recent years, two ideas have been incorporated into the TS method: intensification and diversification, in the perspective of improving the quality of the results produced by that method.

The idea of intensification is to explore in depth the best solution that have been searched out and its neighborhood. The idea of diversification is to force to search into previously unexplored areas of the search space in order to avoid the local convergence.

Intensification procedure

1 Record the current sequence
2 Insert job k in other (n-1) jobs to obtain a set of new sequence; $k \in \{1,2,...,n\}$;
3 Find the shortest sequence in this set;
4 Repeat step 1to 3 until the number of the shortest sequences is $\frac{n}{2}$.

Algorithm 2-4: Diversification Strategy

Diversification procedure

1 Save the best sequence;
2 Generate at random an initial sequence;
3 Regenerate $\frac{N}{2}$ random permutations;

Algorithm 2-5: Diversification Strategy

Chapter 3

The Two-Machine Open Shop without Time Delays

3.1 Introduction

Let us recall the description of an open shop scheduling problem. A set of n jobs $J = \{J_1, J_2, ..., J_n\}$ has to be processed on a set of m machines $M = \{M_1, M_2, ..., M_m\}$. The routing of the jobs through the machines is not known in advance. In fact, it is part of the solution as it becomes known during the process of building the schedule.

Let us mention that open shop scheduling problems may arise in many applications. Take a large aircraft garage with specialized work-centers for example. An airplane may require repairs on its engine and electrical circuit system. These two tasks may be fulfilled in any order. Other examples of open shop problems include examination scheduling, testing repair operation scheduling, satellite communications, semiconductor manufacturing, quality control centers, etc.

The two-machine open shop problem without time delays ($O_2 \parallel C_{\max}$) describes the simplest and easiest state of the problem. So, in some cases, the result of $O_2 \parallel C_{\max}$ problem is considered as a lower for other complex two-machine open shop problems, and may also provide an important theoretical basis for solving other complex open shop problems.

In this section, the Gonzalez-Sahni algorithm and the Schrage-Pinedo Algorithm (LAPT) are presented to solve the $O_2 \parallel C_{\max}$ algorithm. We restate the latter algorithm for an easy implementation and give a proof of its optimality.

3.2 Gonzalez-Sahni Algorithm

Gonzalez & Sahni [1976] present a polynomial algorithm to generate an optimal solution for the $O_2 \parallel C_{\max}$ problem, denoted hereafter as GS algorithm.

- **The basic idea of GS algorithm**

Let $a_j = p_{1j}, b_j = p_{2j}$; GS algorithm consists of splitting the set J of jobs into two parts as follows: $\phi = \{J_j \mid a_j \geq b_j\}$, $\gamma = \{J_j \mid a_j < b_j\}$. The schedule is built

from the "middle", with jobs from ϕ added on at the right and those from γ at the left. Finally, some finishing touches involving only the first and last jobs in the schedule are made. The algorithm can be described as follows:

Begin

Choose any two jobs J_k and J_j for which $a_k \geq \max_{J_i \in \sigma} \{b_i\}$ and $b_j \geq \max_{J_i \in \lambda} \{a_i\}$;

Set $\phi := \phi - \{J_k, J_j\}$;

Set $\gamma := \gamma - \{J_k, J_j\}$;

Construct separate schedules for $\phi \cup \{J_j\}$ and $\gamma \cup \{J_k\}$;

Join both schedules;

Move tasks from $\gamma \cup \{J_j\}$ processed on machine 2 to the right;

Change the order of processing on machine 2 in such a way that J_k is processed first on this machine;

End

Algorithm 3-1: GS algorithm

Example 3.1

Let us illustrate the GS algorithm on an instance of the two-machine open shop with four jobs. The processing times are as follow.

	J_1	J_2	J_3	J_4
M_1	2	6	8	3
M_2	6	2	7	8

Table 3-1: Processing times for an instance with 4 jobs

We have that $\phi = \{J_2, J_3\}$; $\gamma = \{J_1, J_4\}$; $b_l = J_4 (b_l \geq \max_{J_i \in \lambda} \{a_j\})$. It then follows that $b_l = J_4 (b_l \geq \max_{J_i \in \lambda} \{a_j\})$; $a_k = J_3 (a_k \geq \max_{J_i \in o} \{b_j\})$; $\phi' \cup \{J_j\} = \{a_3, a_2\}$; $\gamma' \cup \{J_k\} = \{b_1, b_4\}$; see Figure 3-1.

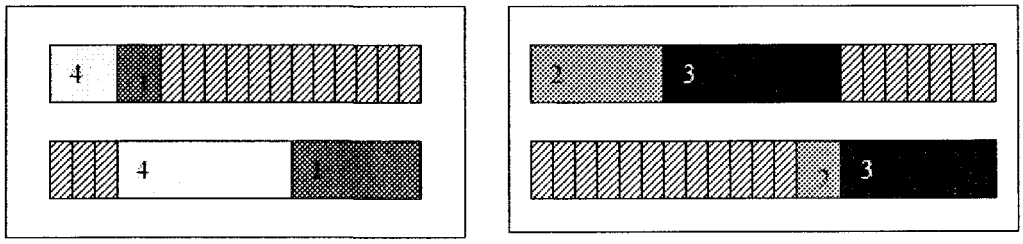


Figure 3-1: Two partial schedules

Both schedules are joined and the order of processing on machine 2 is changed in such a way that b_3 is processed first on this machine, as illustrated by Figure 3-2.

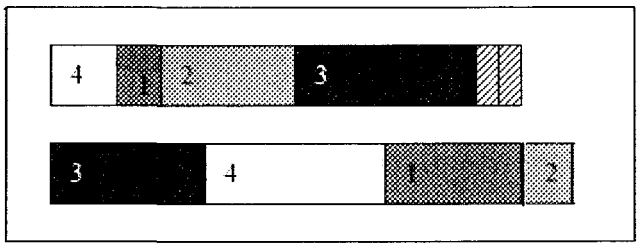


Figure 3-2: Optimal Solution produced by the GS algorithm

3.3 Pinedo-Schrage algorithm

The optimal solution of the above problem can be found in another way. Indeed, Pinedo and Schrage [Pindedo, 1982] presents LAPT (Longest Alternate Processing Time) algorithm. The idea of this algorithm is as follows.

1. Let p be the job with the longest processing time. If this happens on machine M1 (M2), then process job p on machine M2 (M1).
2. Process the rest of the jobs arbitrarily on both machines as soon as they become free.
3. Process job p on machine M2 (M1) either as the last job or before the last job which is being processed on machine M1.

LAPT Algorithm

Again, let us run LAPT algorithm on Example 3-1:

At time 0, J_4 is processed on M1, with $a_3 = 8$, and M2 is idle. So J_3 is processed on M2; at time 2, M1 is idle and $b_3 = 7$. But because J_3 is being processed on M2, $b_1 = 6$, so J_1 is processed on M1. This process is repeated until all jobs are completed on both machines. In the end, we get the following solution as in Fig 3-3.

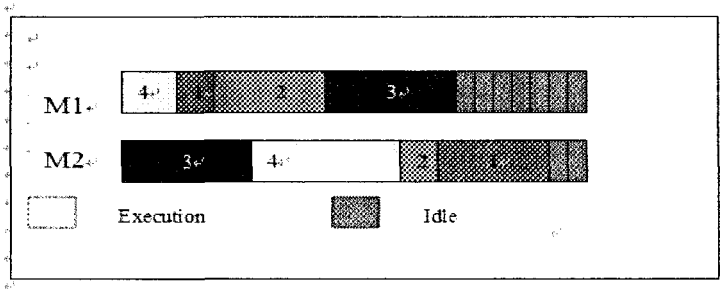


Figure 3-3: Optimal Solution produced by the LAPT algorithm

In what follows, we propose a simpler way to describe the above algorithm. Our proposal is twofold: the algorithm is easy to implement and prove its optimality. First, let us define the following.

Definition 3-3-1

In a two-machine open shop problem, one of the operations of a job is going to be processed before the other operation. Such an operation is called the first operation; the other one is called the second operation.

For an easy implementation, below is another way of stating LAPT algorithm:

1. Let p_{hk} =max { p_{ij} ; i=1,2; j=1...n};
2. Process job *k* first on machine M(3-*h*);
3. For (i=1; i<n ;i++) with i not equal to *k;*
   Process first operation of job *i* on the first available machine;
4. Process job *k* on machine *h*;
5. For (i=1; i<n ;i++) with *i* not equal to *k*
   Process second operation of job *i* on the first available machine;

Another Version of LAPT algorithm

Once again, let us run LAPT algorithm on Example 3-1. Let us denote $p_{hk} = \max\{p_{11}, p_{12}, p_{13}, p_{14}, p_{21}, p_{22}, p_{23}, p_{24}\} = p_{13}$. First, J_3 is processed on M2. Second, the first operations of J_1, J_2, J_4 are processed on the first available machine. Third, second operation of J_3 is processed on M1. At last, second operations of J_1, J_2, J_4 are processed on the first available machine. We therefore get the following optimal solution:

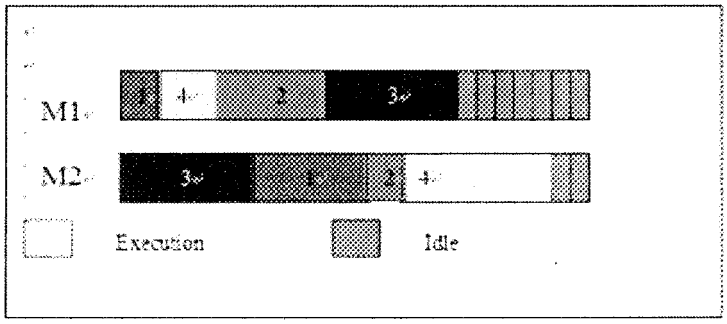


Figure 3-4: Optimal solution produced by the modified LAPT algorithm

Proof of optimality: Let n_1 and n_2 be the number of first operations processed on M1 and M2, respectively. Let also t_1 and t_2 be the time at which first operations on M1 and M2 are completed on M1 and M2, respectively. Let us recall that the algorithm process first operation greedily on both machines. This means that the next first operation is always processed on the first free machine. Let us distinguish the following two cases when it comes to process the second operations.

Case 1: $n_1 = 1$ **and** $n_1 > 1$: We distinguish two sub-cases either $t_1 \geq t_2$ or $t_1 < t_2$.

Subcase 1.1. $t_1 < t_2$. This case is pictured by Figure 3-5:

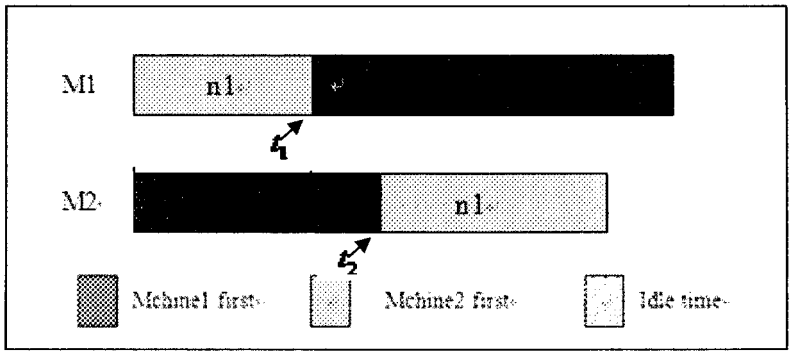


Figure 3-5: Optimal Solution for Case 1 ($t_1 < t_2$)

Since $t_1 < t_2$., then second operation of jobs processed on machine 1 can be processed without an idle time. If $C_{max}(I)$ denotes the generated makespan for instance *I*, then we have

$$C_{max}(I) = \max\{\sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}\},$$

which is nothing else than one of the lower bounds given above. Therefore, this solution is optimal.

Subcase 1.2. $t_1 \geq t_2$**:** In this case, job 1 may cause an idle time, if the processing of its operations is bigger than those of the first operations of jobs processed on M2. This case is shown by Figure 3-6.

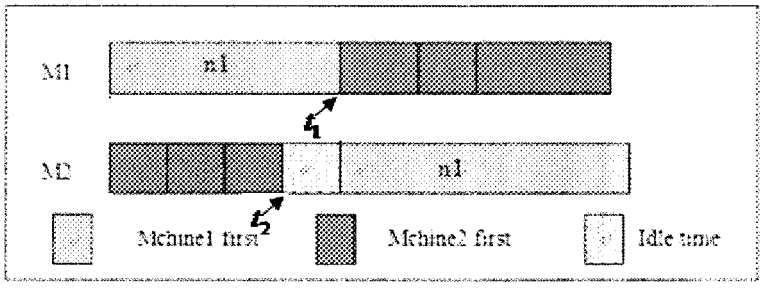


Figure 3-6: Optimal Solution for Case 1 ($t_1 \geq t_2$)

If $C_{max}(I)$ denotes the generated makespan for instance *I*, then have that

$$C_{max}(I) = \max\{p_{1j} + p_{2j}, \sum_{j=1}^{n} p_{1j}\},$$

which is nothing else than one of the lower bounds given above. Therefore, this solution is optimal. Now, if there is no idle time on two machines, we have that

$$C_{\max}(I) = \max\{\sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}\},$$

which is nothing else than one of the lower bound given above. Thus, the optimality of the solution follows immediately.

Case 2. $n_1 > 1, n_2 \geq 1$: We distinguish four sub-cases:

Subcase 2.1. $t_1 \geq t_2 (n_2 = 1)$**:** This case is pictured by Figure 3-7:

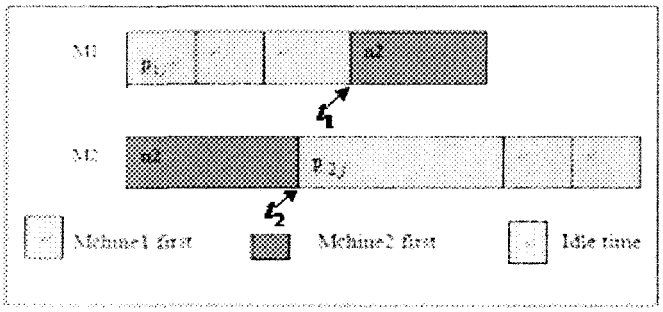


Figure 3-7: The Optimal Solution for Case 2 $(t_1 \geq t_2 (n_2 = 1))$

Obviously, processing time p_{2j} is bigger than the other operation processing times. Therefore, $t_2 + p_{2j} > t_1$. It then follows that second operations of jobs processed on M1 or M2 can be processed without idle time. If $C_{\max}(I)$ denotes the generated makespan of instance *I*, then we have

$$C_{\max}(I) = \max\{\sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}\}.$$

Subcase 2.2. $t_1 \geq t_2 (n_2 > 1)$**:** This is pictured by Figure 3-8.

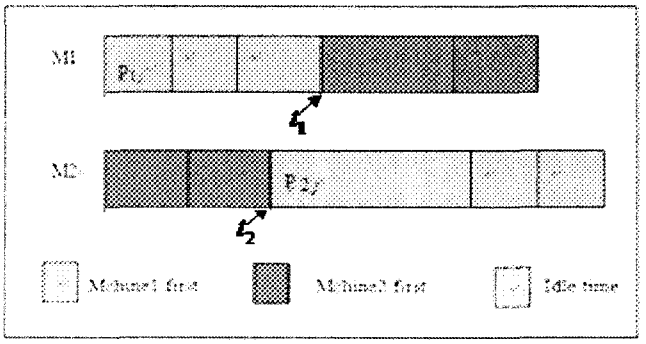


Figure 3-8: Optimal Solution for Case 2 ($t_1 \geq t_2 (n_2 > 1)$)

Since p_{2j} is bigger than the other operations, then $t_2 + p_{2j} > t_1$, so second operations of jobs processed on M1 and M2 can be processed without idle time. If $C_{max}(I)$ denotes the generated makespan of instance *I*, then we have

$$C_{max}(I) = \max\{\sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}\}.$$

Subcase 2.3. $t_1 < t_2$: If there is no idle time on M1, then this case is pictured by Figure 3-9.

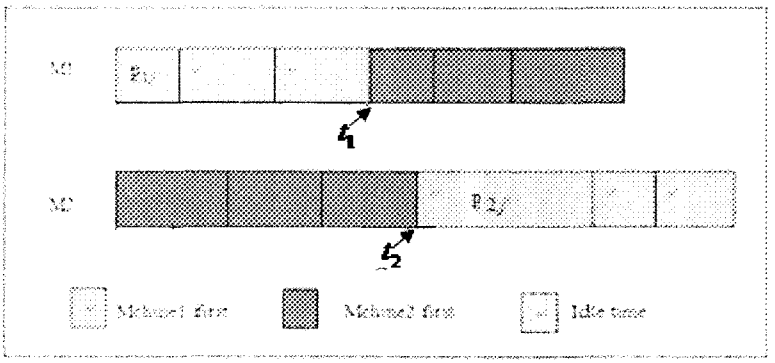


Figure 3- 9: The Optimal solution for Case 2 ($t_1 < t_2$)

Since second operations of jobs processed on M2 are processed without idle time, then, if $C_{max}(I)$ denotes the generated makespan of instance *I*, then we have that

$$C_{max}(I) = \max\{\sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}\}.$$

Subcase 2.4. $t_1 < t_2$: Now, if there is an idle time on M1, then this case is pictured by Figure 3-10.

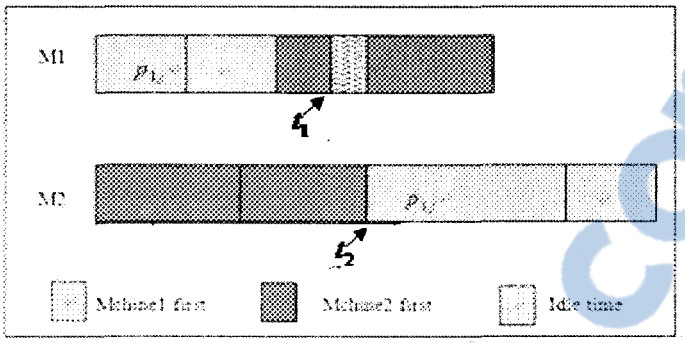


Figure 3-10: Optimal Solution for Case 2 ($t_1 < t_2$)

Since we have that p_{2j} is bigger than the rest of the processing times, then

$$t_2 + p_{2j} \geq \sum_{j=1}^{n} p_{1j} + \text{possible idle time.}$$

So, if $C_{\max}(I)$ denotes again the generated makespan of instance I, then we have

$$C_{\max}(I) = \sum_{j=1}^{n} p_{2j}.$$

In any case, we have shown that the makespan generated is a lower bound. Therefore, the optimality of algorithm follows.

3.3.1 Experimental study

In order to compare the running time of the two algorithms, we run GS algorithm and the new version of LAPT algorithm on the same input data. Due to the fact that both algorithms produce optimal solutions, we only care about their running times. The conducted experiment witnessed 6 stages, where the sizes of problem successively are 50, 100, 200, 500, 800, and 1000, as shown in Table 3.2. For each size, 10 sets of data were generated at random from [1,100]. Column 2 and column 3 of Table 3.2 present, for each size, average running times of GS and LAPT algorithms, respectively. The algorithm was implemented in Visual C++ 6.0 and the tests were run on a personnel computer with a 1.66 GHz Intel® Core™ Duo CPU on the MS Windows XP operating system. The results of the experiment are summarized in Table 3-2.

Value of N	GS algorithm	LAPT algorithm
50	0.09765	0.09457
100	0.09829	0.09712
200	0.10262	0.09809
500	0.10451	0.10101
800	0.10675	0.10356
1000	0.10895	0.10543

Table 3- 2: The running times of GS and LAPT algorithms

Discussion

From the structure perspective, LAPT algorithm is better than GS algorithm. In the latter algorithm, all jobs can be divided into two groups (ϕ, γ) and find, in each group, two jobs meeting conditions $a_k \geq \max_{J_k \in \phi}\{b_j\}, b_l \geq \max_{J_l \in \lambda}\{a_j\}$ and place them on the corresponding positions to process. However, in the LAPT algorithm, we only need to find the maximum processing time of the whole set of the jobs

When comparing these two algorithms, from the running times point of view, then, without a surprise, LAPT algorithm outperforms slightly better GS algorithm, as we can see from the results of Table 3-2 (even though, this difference is not significant).

Chapter 4

The Two-Machine Open Shop With Symmetric Time Delays

4.1 Introduction

In actual production runs, all kinds of consumptions are inevitable, such as the time consumed, the weight consumed, Man-made loss of goods brought and so on. in which time delays between the completion time of one job on one machine and the starting time on another machine are one of the most important losses. The time delays describe the waiting time between the completion of an operation and the beginning of the next operation of the same job.

The time delay between the completion of job k on machine i and its start on machine j is denoted by l_{ijk}. Now, if $l_{ijk} = l_{ijk}$, we say that the time delays are symmetric, and if $l_{ijk} = l_{jik}$, then the time delays are said to be job dependant. In this thesis, we restrict our study to the symmetric time delay case.

Let us observe that in some applications, time delays might be larger than the processing times themselves. That is to say that we do not have choice than considering them, when building a solution.

Most of open shop problems with time delays are NP-hard problems, even with unit-time operations and symmetric time delays [Yu, 1996]. In other words, it is difficult to discover an appropriate exact algorithm running in reasonable time, even for the simplest case.

Let us illustrate this by the following example. In an airport, due to the influence of weather changes, breakdowns of machines, lack of fuel oil, passenger boarding delays and other factors, at a certain interval (for e.g. 10 minutes) the scheduling plan, based on the current situation, needs to be readjusted to avoid accidents, which poses the requirement that the algorithm should be prompt and accurate. Therefore, in many cases, we often employ heuristic algorithms which approach toward but not necessarily ensure the discovery of optimal solution., it needs a set of unified algorithm-evaluating criteria to judge the quality of algorithms, among which the worst-case analysis and the mean performance are the most frequently used criteria.

Apart from ad-hoc algorithms, general heuristic approaches are worth to discuss

as they may be used as a general framework for designing algorithms to solve particular problems. Such approaches are often called meta-heuristics.

In this chapter, there will be an introduction to some different algorithms and improvement strategies for solving the two-machine open shop problem with symmetric time delays problem. Due to the fact that most of two-machine open shop problems with time delays problems are NP-hard problems, how to be able to strike the balance between the quality (precision) of the solution and the time to spend becomes the main issue for us to discuss.

4.2 Lower Bounds

The lower bound of a problem is the minimum value of the considered criterion that is the smallest makespan that can be obtained under ideal constraints. Different focuses on a problem may produce several different lower bounds for the problem.

Whether in heuristic algorithms, meta- heuristic algorithms or exact algorithms, lower bounds play an important role. Indeed, for the heuristic approach, we usually use lower bounds to derive an upper bound on the error (either relative or absolute) of the optimal solution. In the meta-heuristic approach, the lower bound might guide us either in the process of designing the meta-heuristic or in evaluating the solution processed to the lower bound which we have in hand. Finally, in the exact method, lower bounds might help us to either process to the optimality the partial solution at hand or ignore many partial solutions, whenever a branch and bound algorithm is used.

It is evident that the closer a lower bound is to the optimal solution, the more important role it will play. However, a lower bound is only a focus on one aspect of a problem. So, generally, the same problem includes several different lower bounds.

In this section, we focus on the minimal time delay and see how to establish lower bounds. Let us first consider the special case of unit-time operations, before proceeding with the general case.

4.2.1 Unit-time operations

Let us recall that the problem we have at hand in this section is denoted by $O_2 \mid p_{ij} = 1, l_i \mid C_{\max}$. Let us also recall that because there are only two machines, then we have that $l_{ijk} = l_k$ to denote the symmetric time delay associated with job k, no matter the direction this job is moving to. This problem is known to be NP-hard in the strong sense as proved in Yu [1996]. In what follows, we present several lower bounds.

Lemma 1: If $\omega_{opt}(I)$ denotes the optimal makespan for an instance I for the two-machine open shop problem with unit-time operations, then we have that

$$\omega_{opt}(I) \geq \max \{l_j + 2 : j = 1,..., n\} \qquad (1\text{-}1)$$

Proof: Let us consider job k in an optimal schedule. The earliest time it can be completed its processing is in time 1. Due to its time delay, l_k then the earliest time it can be completed on the other machine is $1 + l_k + 1$. Therefore, the result follows.

Lemma 2: If $\omega_{opt}(I)$ denotes the optimal makespan for an instance I for the two-machine open shop problem with unit-time operations, then we have that

$$\omega_{opt}(I) \geq n \qquad (1\text{-}2)$$

Proof: This is clear that if the schedule has no idle time slot on the two machines, then it is the best schedule on ca produce. Since the processing times are unit and there are n jobs, the result follows immediately.

In what follows, we aim at getting lower bound involving the whole set of time delays.

Lemma 2: If $\omega_{opt}(I)$ denotes the optimal makespan for an instance I for the

two-machine open shop problem with unit-time operations, then we have that

$$\omega_{opt}(I) \geq \left\lceil \frac{\sum_{i=1}^{n} l_j}{n} \right\rceil + \left\lceil \frac{n}{2} \right\rceil. \qquad (1\text{-}3)$$

Proof: Given a schedule, we let OS1 denote the set of operations that are processed first on machine M1 and then on machine M2, respectively, and OS2 denote the set of operations that are processed first on machine 2 and then on machine M1.

Let us focus on OS1 (OS2) and assume that the number of jobs processed this way is $n_1 (n_2)$. Let sequence $\sigma(\tau)$ denote the permutation of processing of jobs in OS1 (OS2). If job $k \in OS1$, then $\sigma^{-1}(k)$ $(\tau^{-1}(k))$ denotes the position of job k processed in permutation $\sigma(\tau)$; see Figure 4-1.

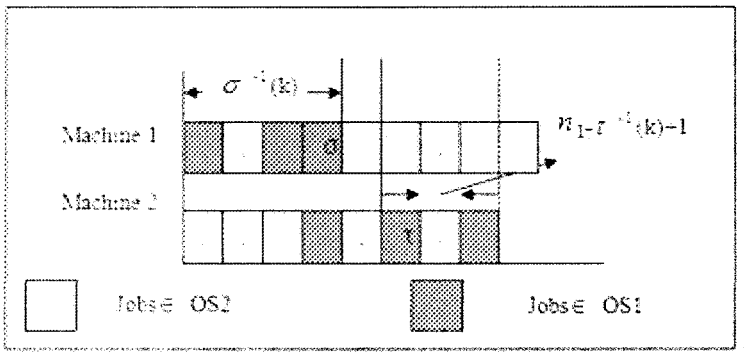


Figure 4-1: A Schedule for OS1

Now, we obtain C_{os1} for job k on OS1 as follows:

$$C_{os1} \geq \sigma^{-1}(k) + l_k + (n_1 + 1 - \tau^{-1}(k)) \text{ for } k \in N_1.$$

Similarly, for $k \in$ OS2, let $\omega^{-1}(k)$ denote $\sum_{j \leq \sigma^{-1}(k)} p_{1\sigma}(j)$ and $\beta^{-1}(k)$ denote $\sum_{j \leq \tau^{-1}(k)} p_{2\tau}(j)$. We can obtain C_{os2} for job k on OS2:

$$C_{os2} \geq \omega^{-1}(k) + l_k + (n_2 + 1 - \beta^{-1}(k)); \text{ for } k \in N_2.$$

For OS2 the quantity of similar inequalities is N2; we end up with the new following inequality:

$$n_2 C_{os2} \geq \sum_{j \in n_2} \omega^{-1}(j) + \sum_{j \in n_2} l_j + (n_2^2 + n_2 - \sum_{j \in n_2} \beta^{-1}(j)) \text{ ; for all } j \in \mathrm{N}_2 . \quad (1\text{-}5)$$

Since $\sigma^{-1}(j)$ and $\tau^{-1}(j)$ for n_1, $\omega^{-1}(j)$ and $\beta^{-1}(j)$ for n_2 are permutations whose values are in $\{1,...,n_1\}$ and $\{1,...,n_2\}$, respectively, it then follows that

$$\sum_{j \in n_1} \sigma^{-1}(j) - \sum_{j \in n_1} \tau^{-1}(j) = 0,$$
$$\sum_{j \in n_2} \omega^{-1}(j) - \sum_{j \in n_2} \beta^{-1}(j) = 0.$$

As we have $n_1 + n_2 = n$ and $C_{\max} \geq C_1 \text{ or } C_2$; if we add up inequalities of (1-4) with inequality (1-5), then we obtain that

$$nC_{\max} \geq \sum_{j=1}^{n} l_j + n_2^2 + n_1^2 + n ; \quad (1\text{-}6)$$

As $n_1 + n_2 = n$, then we can know that

$$2(n_2^2 + n_1^2) - (n_1 + n_2)^2 = (n_1 - n_2)^2 \geq 0 ;$$

$$\Rightarrow 2(n_2^2 + n_1^2) \geq (n_1 + n_2)^2$$

$$\Rightarrow (n_2^2 + n_1^2) \geq \frac{n^2}{2}. \quad (1\text{-}7)$$

From (1-6) and (1-7), we may derive

$$C_{\max} \geq \frac{\sum_{i=1}^{n} l_j}{n} + \frac{n}{2} + 1.$$

It then follows that

$$C_{\max} \geq \left\lceil \frac{\sum_{i=1}^{n} l_j}{n} + 1 + \frac{n}{2} \right\rceil$$

$$\geq \left\lceil \frac{\sum_{i=1}^{n} l_j}{n} \right\rceil + \left\lceil \frac{n}{2} \right\rceil . \quad (1\text{-}8)$$

Therefore, the result is established.

4.2.2 General processing times

Let us now pass to the case where the processing times are general, and proceed as in the same lines.

Lemma 1: If $\omega_{opt}(I)$ denotes the optimal makespan for an instance I for the two-machine open shop problem with time delays, then we have that

$$\omega_{opt}(I) \geq \max(\quad p_{1j} + l_j + p_{2j}). \tag{2-1}$$

Proof: Let us consider job k in an optimal schedule. The earliest time it can be completed is p_{1k}. Due to its time delay l_k the earliest time it can be completed on the other machine then $p_{1k} + l_k + p_{2k}$. Therefore, the result follows immediately.

Lemma 2: If $\omega_{opt}(I)$ denotes the optimal makespan for an instance I for the two-machine open shop problem with time delays, then we have that

$$\omega_{opt}(I) \geq \max(\sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}). \tag{2-2}$$

Proof: This is clear that the best schedule we can generate is the one with no idle time on two machines. Since jobs are processed on both machines and that the makespan can be produced by either machine, then the least value is $\max(\sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j})$. Therefore the result follows immediately

Lemma 3: If $q_j = \min(p_{1j}, p_{2j})$ and $r_j = \max(p_{1j}, p_{2j})$, then if $\omega_{opt}(I)$ denotes the optimal makespan for an instance I for the two-machine open shop problem with time delays, then we have that

$$\omega_{opt}(I) \geq \left\lceil \sum_{j \in N} q_j (l_j + r_j - 1) / \sum_{j \in n} q_j \right\rceil + \left\lceil \sum_{j \in N} q_j / 2 \right\rceil. \qquad (2\text{-}3)$$

Proof: Let OS1 denote the set of sub-operations that are first processed on machine M1 and then on machine M2, respectively, and OS2 denotes the set of sub-operations that are processed first on M2 and then on M1. For OS1, taking the viewpoint of job splitting, we may assume that each job j contains q_j artificial jobs with unit processing times. We consider the following two cases:

Case 1: $p_{1j} \leq p_{2j}$

In this case, $q_j = p_{1j}$ and $r_j = p_{2j}$. Each job j contains q_j artificial jobs with unit processing times, so each artificial job J_{jk}. So, its operation on machine M1 is taken as the sub-operation in the *k-th* unit time slot of job j on M1. And its operation on M2 is taken as the sub-operation in the $r_j - J_{jk}^{-1} - 1 - th$ unit time slot of job j on M2. Where $k = 1,2...q_j$ and J_{jk}^{-1} denote the positions of the artificial job J_{jk} on job j, respectively; see Figure 4-2.

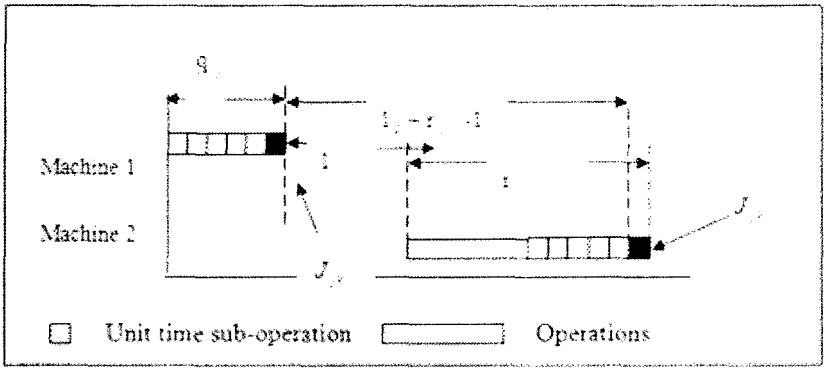


Figure 4-2: Case where $p_{1j} \leq p_{2j}$

These unit time sub-operations have a new common delay L_j.

$$L_j = J_{jk}^{-1} + l_j + r_j - J_{jk}^{-1} - 1 = l_j + r_j - 1 \qquad (2\text{-}4)$$

Case 2: $p_{1j} > p_{2j}$

In this case, $r_j = p_{1j}$ and $p_j = p_{2j}$. For each job j contains q_j artificial jobs with unit processing times, so each artificial job J_{jk}, its operation on M1 is taken as the sub-operation in the k-th unit time slot of job j on M1. And its operation on M2 is taken as the sub-operation in the $J_{jk}^{-1}-1$-th unit time slot of job j on M2. Where $k = 1,..., q_j$, see Figure 4-3.

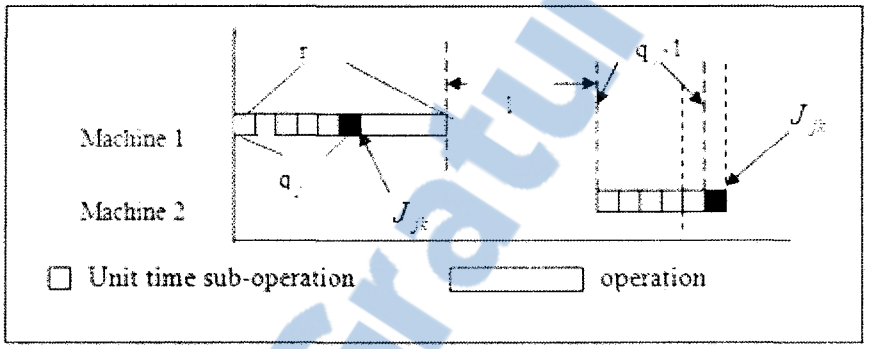


Figure 4-3: Case where $p_{1j} > p_{2j}$

$$L_j = J_{jk}^{-1} + \tau_j + l_j - J_{jk}^{-1} - 1 = l_j + r_j - 1, \tag{2-5}$$

where J_{jk}^{-1} denote the positions of the artificial job J_{jk} on job j. Similarly, we can obtain the same conclusion on OS2.

From Lemma 2, with $L_j = l_j + r_j - 1$ and $n = q_1 + q_2 + ... + q_n = \sum_{j \in n} q_j$, we then derive the following lower bound:

$$LB = \left\lceil \sum_{j \in N} q_j (l_j + r_j - 1) / \sum_{j \in n} q_j \right\rceil + \left\lceil \sum_{j \in N} q_j / 2 \right\rceil. \tag{2-6}$$

4.3 Heuristic approach

Let us observe that most of open shop problems are NP-hard, and their optimal

solutions are not always successfully obtained in reasonable time. In that case, we use heuristic algorithms to solve those problems. But at some point, we may not be satisfied with the solution of the current heuristic algorithm. We may then be interested to improve the quality of the solution. To do so, we may consider either the running time or the quality of the solution as the primary factor to improve. However, since the running time of most of heuristic algorithms is satisfactory, we mainly focus on the quality of the heuristic algorithm as the improvement criterion. Generally, this is evaluated through the quality of the worst-case solution.

In this section, first we present some worst-case results. Then, in the second step, an experimental study is conducted to compare two given heuristic algorithms.

4.3.1 Worst-case analysis

The worst case analysis is to simulate and analyze the bound that can be reached under the worst circumstance. However, sometimes the result may be overly pessimistic. So, it does not necessarily comply with real situations, but provides feasible theoretical upper bound on the result produced by the heuristic algorithm.

Theorem 1: [Strusevich 1999]

Let $C_{\max}(H)$ *and* $C_{\max}(S)$ *denote the makespan generated by heuristic H and optimal solution S, respectively. If m=2, and the time delays are symmetric, then there exists a heuristic H such that*

$$\frac{C_{\max}(H)}{C_{\max}(S)} \leq \frac{3}{2}.$$

Furthermore this bound is tight.

Theorem 2 [Rebaine, 2004]

Let $C_{\max}(H)$ *and* $C_{\max}(S)$ *denote the makespan generated by heuristic H and optimal solution S, respectively, for the case of unit-time operations. If m=2, and the*

time delays are symmetric, then

$$\frac{C_{\max}(H)}{C_{\max}(S)} \leq \frac{3}{2} - \frac{1}{2n}.$$

Furthermore, this bound is tight.

Theorem 3: [Rebaine and Strusevich, 1999]

Let $C_{\max}(H)$ *and* $C_{\max}(S)$ *denote the makespan generated by heuristic H and optimal solution S, respectively. If m=2, and the time delays are non symmetric and constant from M1 (M2) toM2 (M1), then*

$$\frac{C_{\max}(H)}{C_{\max}(S)} \leq \frac{8}{5}.$$

Furthermore, this bound is tight.

4.3.2 Experimental study with unit-time operations

In this section, we deal with the problem of unit-time operations and symmetric time delays. We present two simple heuristics and compare their performance throughout an experimental study.

Algorithm 1

The idea behind Algorithm 1 is to give the priority first to the jobs with the biggest time delays. It can be described as follows:

1. Rename the jobs such that that $l_1 \geq \ldots \geq l_n$;
2. For *(j=1; j≤n; j++)*
   Process the first operation of job *j* on the first available machine;
3. For *(j=1; j≤n; j++)*
   Process the second operations of jobs on the corresponding machine as soon as possible;

Algorithm 1

Algorithm 2

Basically, Algorithm 2 is similar to Algorithm 1, except that it avoids collisions as much as possible with jobs in the sense that, as long as it is possible, it finds time slots such that the two operations of a given job are processed at its exact time delay apart. The description of Algorithm 2 is as follows.

1. Rename the jobs such that $l_1 \geq \ldots \geq l_n$;
2. Process the first operation of job 1 on M1 (M2), respectively.
3. Process the second operations of jobs in corresponding positions such that the delays are exact between the two operations.
4. For *(j=3; j≤n; j++)*
   Process job *j* in available positions on M1 and M2 such that the delays are exact between the two operations, if the first operation of job *j* is processed on the last position of the current machine then schedule the other operation of job j on the first available position of the other machine;

Algorithm 2

Let us consider the following example to describe the execution process of Algorithm 1 and Algorithm 2, respectively.

Example 4-1

Consider an instance with two machines, 9 unit-time operation jobs, and the following symmetric time delays.

Job *j*	1	2	3	4	5	6	7	8	9
Time delay	6	6	5	4	3	3	2	2	2

Table 4-1: Instance with N=9

Figure 4-4 shows the running of Example 4-1 to Algorithm1.

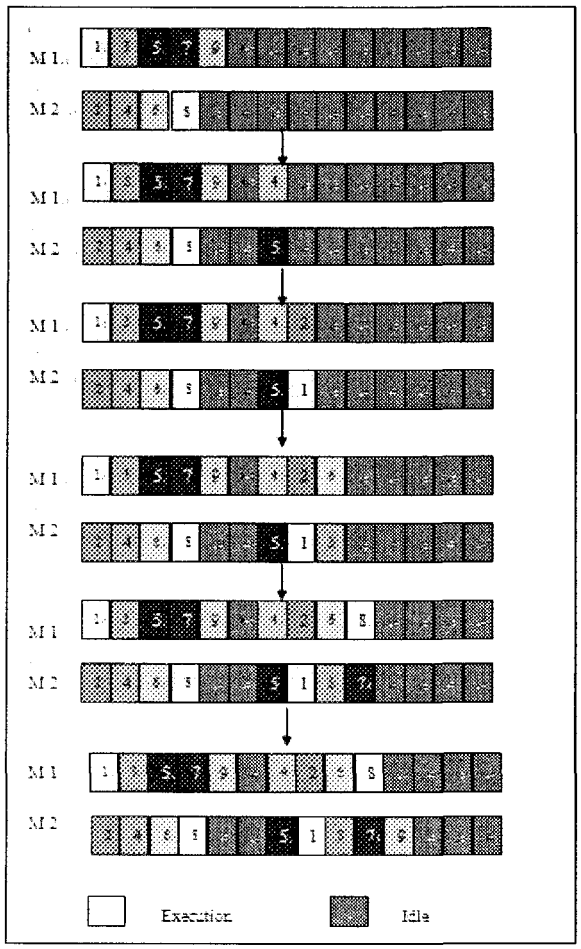


Figure 4- 4: Algorithm 1 through Example 4-1

Figure 4-5 shows the running of the same example 4-1 by Algorithm2.

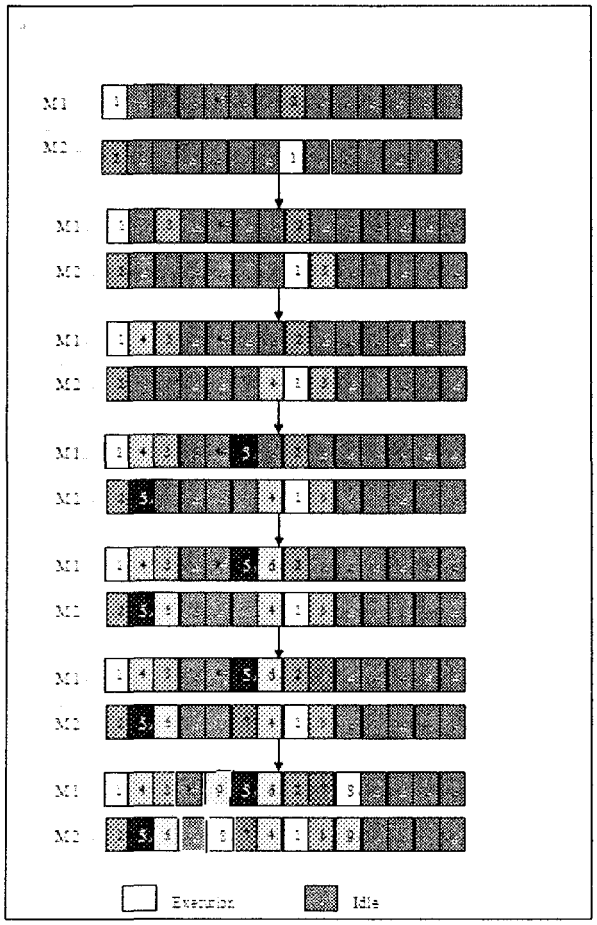


Figure 4-5: Algorithm 2 through Example 4-1

To proceed with the comparison between the two algorithms, we use the empirical approach. Within this approach, we need to collect a large number of data on which both algorithms are tested.

Both algorithms are coded in Visual C++ 6.0 and implemented on a personnel computer with 1.66 GHz Intel® Core™ Duo CPU and 1 GB memory on the MS Windows XP operating system.

At first, we generated a set of random initial time delays within [0, 10] to be tested on Algorithm 1 and Algorithm 2, respectively. The experiment includes 50 instances for each class, thus 400 instances in total. The second and fifth columns represent the processing time for Algorithm1 and Algorithm 2, the fourth and third columns represent the average makespan for Algorithm 1 and Algorithm 2, respectively. The results produced by Algorithm 1 and Algorithm 2 are denoted by A

(B), and we tested whether A<B, A=B and A>B, as indicated respectively in column 6, 7, and 8 in Table 4-2.

The number of jobs	Processing Time for Algorithm1	Processing Time for Algorithm2	Average makespan for Algorithm1	Average makespan for Algorithm2	A>B	A=B	A<B
5	0.8	0.8	9	9	0	50	0
10	1.1	1.1	11	11	0	50	0
15	1.2	1.3	12	12	6	24	20
20	1.2	1.3	15	14	25	15	0
30	1.4	1.5	21	17	50	0	0
50	1.6	1.6	51	34	50	0	0
100	1.8	1.9	98	79	50	0	0
200	2.1	2.2	187	163	50	0	0

Table 4-2: Results Produced by Algorithm 1 and Algorithm 2

When the size of the problem is smaller than or equal to 15, the quality of two algorithms is similar. However, when the size of the problem gets larger than 20, the results of Algorithm 2 is far better than that of Algorithm 1.

4.4 Meta-heuristic Algorithms

For some special cases, heuristic algorithms can have some satisfactory results, but it is not universal. In other words, the heuristic algorithm can have a good result for a special problem, but not for all problems. Obviously, for each special problem, it is very troublesome and difficult to find out its corresponding algorithm. Therefore, some general heuristic algorithms are very important, and one of them is the meta-heuristic algorithm, which is a very famous method for solving a very general class of computational problems. A review of literature introduces meta-heuristic algorithms for solving the open shop problem. Tabu search, introduced by Glover [1989,1990, 1997], is a local search approach designed for solving hard combinatorial optimization problems. More refined versions and a large number of successful applications to improve heuristic algorithms can be found as follows: Liaw has worked extensively on the open shop problem, proposing a tabu search algorithm

[1999a, 2003], simulated annealing [1999b], and hybrid genetic algorithm and search [2000]. Alcaide *et al.* [1997] present a tabu search algorithm for the minimum makespan of open shop problem. A promising hybrid (GA) heuristic approach for open-shop scheduling problems is published by Fang *et al.* [1994].

In this section, there will be an introduction on some general heuristic algorithms (meta-heuristic), which may be used as a framework to design algorithms for solving general NP-hard problems. Although the framework is the same, if we can still make adjustments to improve significantly the efficiency of the resulting algorithm.

The basic strategy of a meta-heuristic algorithm improvement (adjustement) includes the following criteria.

1. Stopping criterion

Stopping criterion does not depend on the algorithm details (framework). It is used to avoid unnecessary costs. Generally, stopping criteria are as follows:

a) The qualified result has been found; for example, the result is equal to the lower bound.

b) It takes too long. For example: the algorithm enters into a deadlock.

c) The possibility of the result improvement is too low.

2. Internal Structure: Each meta-heuristic algorithm has its own formwork which includes initial value, special parameters, neighborhood structure and so on. Intensification and diversification module is also the key point for the improvement of meta-heuristic algorithms.

3. Hybrid algorithm: If it is difficult for the meta-heuristic algorithms to break the shackles, combining the advantages of different algorithms to create a new hybrid algorithm is a prevailing practice.

In this dissertation, we mainly introduce the improvement of two meta-heuristic algorithms (tabu search and simulated annealing) for the two-machine open-shop problem with time delays.

4.4.1 Internal Structure

As far as meta-heuristic algorithms are concerned, despite the fact that they all have their own frameworks and parameters, they also share some common points. For example, the value ranges of most meta-heuristic algorithms are in whole domain, so the strategies of intensification and diversification both can be made use of to conduct further exploration and exploitation on value taking. Therein, the idea of intensification is to thoroughly explore more of the current solution in order to find the global best solution. The idea of diversification is to force to search the previously unexplored areas of the search space in order to avoid local convergence.

4.4.1.1 Simulated Annealing

Simulated annealing is a method that attempts to simulate the physical process of annealing. Annealing is where a material is heated and then cooled (as steel or glass) usually for softening and making the material less brittle. Simulated annealing, therefore, exposes a "solution" to "heat" and cools producing a better solution. Generally speaking, with the control parameter T gradually decreasing, algorithm converges to the set of the optimal solution. In theory, if the final temperature T_j is small enough, the optimal solution can be obtained. However, we cannot directly control CPU time during the execution of this algorithm. To do so, we used the iterative time L.

1. Basic algorithm

We set the iterative time =1000. The results of the experiment show that the transition from 1 to 400 was significantly improved. However, from 500 to1000 iterations, the transition did not improve the results. In terms of the experiment accuracy, in the following tests, we set the iteration time to 600, the initial temperature T_0 is 600, terminated temperature T_j is 0.01, and the cooling factor is 0.8.

```
Obtain an random sequence f(i);
Initial temperature T0 = 600; the iterative time x=1;
Repeat:
    Generate a random neighbor f(j) from f(i);
    Δf=f(j)-f(i);
    If Δf<=0 then accept the new sequence;
              Else
                    If Δf>0 then
                                get a random number h∈(0,1);
                                If h >exp(-Δf/T)
                                  Then accept the new sequence;
    If T>0.01 then T=T*0.8;
              Else x=600;
    x=x+1;
  Until x>=600;
```

Simulated Annealing Algorithm

In addition to the strategy of the algorithm, simulated annealing algorithm also includes *a calculation module* that can be used to calculate the value of *f(i)*.

1 . The order of *N* jobs sequence on M1 (M2) is S1 (S2), respectively.

T1 = the completion time of job S1;
T2 = the completion time of jobs in S2;

2 . If T2<T1 then choose the first job *x* ($x \in N$) of the processing sequence S2

If *x* has been processed on M1
Then T2=max {the completion time of job S1(*x*) + time delay of job *x*, T2}
+ the processing time of job S2(x);
Else T2= T2+ the processing time of job S2(*x*);

6. If T1<= T2 then choose the first job *y* ($y \in N$) of the processing sequence S1

If *y* has been processed on M2
Then T1=max {the completion time of job S2(y) + time delay of job y, T1}
+ the processing time of job S1(y);
Else T1= T1+ the processing time of job S1(y);

6. Repeat 5 and 6 until the processing of the jobs are finished
7. Return T=max (T1, T2).

Calculation module

Example 4.1

Let us illustrate the calculation module on a processing sequence:

	J_1	J_2	J_3	J_4
Processing time on M1	2	6	6	3
Processing time on M2	6	3	4	1
Time delay	2	3	5	3

Table 4-3: Processing times for an instance with 4 jobs

The aim of calculation module is to obtain the minimum total processing time of the current order of the processing sequence. For Example 4.1, we assume that the processing order has been received as follows:

We know that the processing sequence on M1 and M2 are, respectively, $J_2 \rightarrow J_3 \rightarrow J_1 \rightarrow J_4$, and $J_1 \rightarrow J_2 \rightarrow J_4 \rightarrow J_3$. Let T1 and T2 denote respectively the completion time of machine M1 and M2. Initially, we have T1 = 0, T2= 0. J_2 (J_1) are both taken as the first job of the unprocessed sequence on M1 (M2), respectively and when the jobs are completed, T1=6, T2=6. At this time, T1= T2 , then J_3, as the first job of the unprocessed sequence on M1, will be processed. Since J_3 is not processed yet on M2, the start time of J_3 on M1 is T1. Update T1 = T1 + the processing time of J_3 on M1 = 6+6 = 12. Now, since T1<T2, then J_2, as the first job of the unprocessed sequence on M2, will be processed. Since J_2 is being processed on M1, it must be taken into consideration whether its processing is completed or not. The start time of J_2 on M2 is max(T2, the completion time of J_2 on M1 + time delay l_2, so we get T2 = max (T2, the completion time of J_2 on M1 + time delay l_2) + the processing time of J_2 on M2, that is T2 = max (6, 6+3) +3=12. Now, since T1=T2=12, then J_1, as the first job of the unprocessed sequence on M1, will be processed.

Since J_1 has been processed on M2, we get that that T1= max(T1, the completion time of J_1 on M2 + time delay l_1) + the processing time of J_1 on M1, that is T1=12+2=14. Now, since T1>T2 then J_4, as the first job of the unprocessed sequence on M1, will be processed. Since J_4 is not processed, we get that T2=T2 + the processing time of J_4 on M1, that is T2 = 12+1 = 13. Again, since T1 > T2, then J_3, as the first job of the unprocessed sequence on M2, will be processed. Since J_3 has been processed on M1, we get T2= max (T2, the completion time of J_3 on M1 + time delay l_3) + the processing time of J_3 on M2, that is T2 = 12+5+4 = 21. Now, since T1 < T2, then J_4, as the first job of the unprocessed sequence on M1, will be processed. Because J_4 has been processed on M2, we get T1= max (T1, the completion time of J_4 on M2 + time delay l_4) + the processing time of J_4 on M1, that is T1 = 13+3+3 = 19. All the tasks on M1 and M2 have been now processed. We therefore get, for the current job sequence, C_{max} = max (T1, T2) = 21.

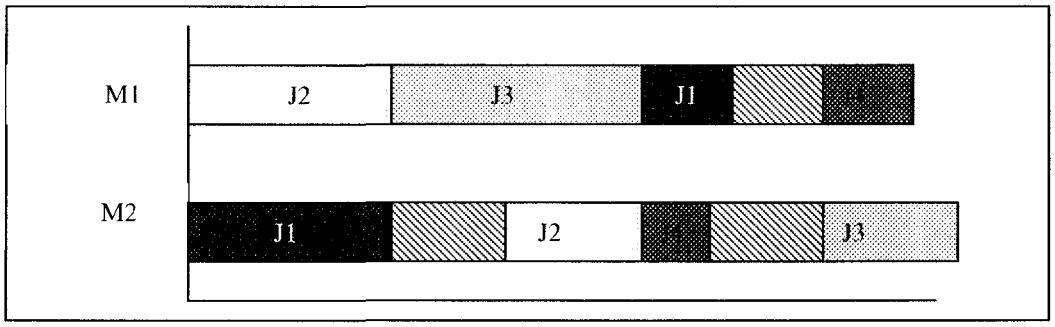


Figure 4-6: solution produced by the Calculation module

Calculation module and algorithm strategy are two relatively independent modules, but there are data exchanges between them. However, if the data types of the transmitted data are consistent, the changes of algorithm strategy will not produce whatsoever influences on the calculation module. Therefore, the same calculation module can be applied to different meta-heuristic algorithm strategies.

The experimental study we conducted was run in Visual C++ 6.0 and implemented on a 1.66 GHz Intel® Core™ Duo CPU and 1 GB memory on the MS Windows XP operating system personal computer.

The conducted experiment witnessed 6 stages, where the sizes of problem successively are 5, 10, 20, 50, 100, and 200, as is demonstrated in Table 4-6. For each size, 10 sets of data were selected at random in [1,100] and involved in the corresponding stage, with the results for the set of all instances in the cooling factor=0.8; The second column represents the average time of execution. The average, the best and the worst makespan are illustrated by the third, fourth and fifth column of Table 4-3, respectively.

The number of jobs	Average time	Average makespan	Best makespan	Worst makespan
5	0.4	186.3	178	209
10	0.4	347.8	319	367
20	0.4	601.7	584	620
50	0.5	1346.7	1298	1386
100	0.6	2596.4	2512	2623
200	0.7	5199.2	5101	5410

Table 4-3: Cooling Factor = 0.8

2. Improving the computational experiments

In terms of different problems, simulated annealing algorithm needs to make corresponding adjustments. Generally, simulated annealing algorithm includes three factors: the initial temperature, the cooling factor, and a diversification and intensification approach.

a. Initial temperature

In the light of the traits of simulated-annealing algorithm, when temperature T drops slowly, the system will accept the inferior solution by the probability of exp $(-\Delta E/T)$ to escape from local optimal solution. That is to say, when $T\rightarrow\infty$, exp

$(-\Delta E/T) \rightarrow 1$. At this time, the system almost can accept all possible variations. But the results of experiment show that when the processing times (delay times) of all jobs are smaller than 100, the difference of the results is not obvious, while the initial time is 600 or 10000. For this reason, in this thesis we set up the initial temperature $T_0 = 600$.

b. Cooling factor

In the annealing process, the new temperature $T_1 = \alpha * T_0$, where α is called the cooling factor. The temperature-decreasing speed would affect the efficiency of the algorithm. If the temperature-decreasing speed is too fast, the algorithm's accepting rate of inferior solutions will be too low, which will make it very easy for the algorithm to fall into local minimum. If the temperature-lowering speed is too slow, the accepting rate of the inferior solution is too high. The result of algorithm is that it is difficult to achieve a stable equilibrium. Figure 4-6 and 4-7 display the output for a typical simulated annealing case run with $\alpha = 0.01$ or $\alpha = 0.99$, respectively.

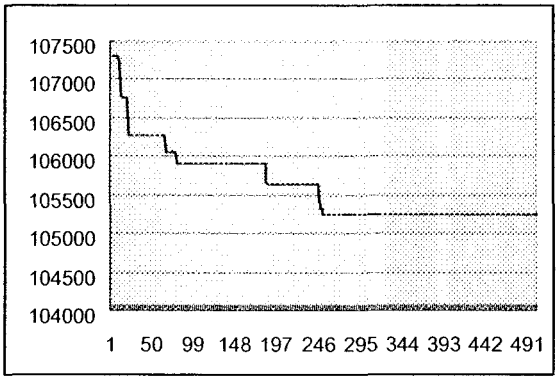


Figure 4-6: Cooling Factor = 0.01

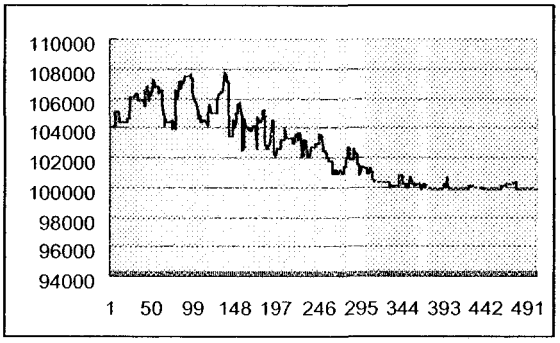


Figure 4- 7: Cooling Factor = 0.99

In order to find a best cooling factor, we get a test. The comparison of the same initial data under different cooling factors shows when the cooling factor is 0.95; we can obtain a better solution. Table 4-3, 4-4, 4-5 and 4-6 present the results on the same bodies with the cooling factor being 0.8, 0.9. 0.95, 0.99, respectively; the conducted experiment witnessed 6 stages, where the sizes of problem successively are 5, 10, 20, 50, 100, and 200, For each size, 10 sets of data were selected at random in [1,100] and involved in the corresponding stage. The second column represents the average time of execution. The average, the best, and the worst makespan are illustrated by the third, fourth and fifth column of Table 4-4, 4-5, and 4-6, respectively.

While the cooling factor = 0.95, which varies from 0.8 to 0.99, the average makespan is improved by 0.2%, 0.0.9% and 1.6%, respectively. The best and worst solutions are improved by respectively 0.01 % (0.005%, 0.5%) and 0.4 % (0.25%, 5.9%).

The number of jobs	Average time	Average makespan	Best makespan	worst makespan
5	0.4	186.3	178	209
10	0.4	347.8	319	367
20	0.4	601.4	584	610
50	0.5	1345.3	1296	1389
100	0.6	2592.3	2511	2623
200	0.7	5191.3	5097	5400

Table 4- 4: Cooling Factor = 0.9

The number of jobs	Average time	Average makespan	Best makespan	Worst makespan
5	0.4	186.3	178	209
10	0.4	347.8	319	367
20	0.4	601.4	584	610
50	0.5	1345.2	1296	1386
100	0.6	2591.3	2511	2622
200	0.7	5186.5	5094	5386

Table 4-5: Cooling Factor = 0.95

The number of jobs	Average time	Average makespan	Best makespan	Worst makespan
5	0.4	186.6	178	220
10	0.4	349.5	322	376
20	0.4	611.3	584	667
50	0.5	1393.2	1320	1476
100	0.6	2648.3	2521	2698
200	0.7	5273.6	5122	5709

Table 4-6: Cooling Factor = 0.99

c.Intensification and diversification procedures

In the annealing process, with temperature T dropping, particles tend towards being ordered. However, in this process, because a portion of particles may be more active than the others, their values taken may be much more than those of the others. Therefore, intensification module lessens the possibility for this portion of particles to be chosen by expanding the scale of choosing particles, to accelerate the drop of temperature. However, in order to avoid a deadlock, we can use diversification module to recall temperature and adjust the cooling factor to redo the temperature reduction.

c.1. Intensification

We set up parameter Maxtime to be the maximum number for the solutions to be taken. In the temperature reduction, the number is increased and the probability for the emergence of acquisition solutions lowered. Intensification module can be described as follows:

```
Function intensification (i, j)
{
Time=0;
While (time< Maxtime )
    Generate a random neighbor f(j) from f(i);
    Δf= f(j)-f(i);
    If  Δf<=0 then accept the new sequence (f*= f(j); f (i)= f(j)
    Else
    If  Δf>0 then
        Get a random number h∈(0,1);
        If h< exp(- Δf/T) then accept the new sequence (f*= f(j); f (i)= f(j);
Time=time-1;
Return f*;
```

Intensification Strategy

Table 4-7 presents the results for the set of all instances in which the cooling factor is 0.95 and the intensification module added; the conducted experiment witnessed 6 stages, where the sizes of problem successively are 5, 10, 20, 50, 100, and 200, For each size, 10 sets of data were selected at random in [1,100] and involved in the corresponding stage. The second column represents the average time of execution. The average, the best, and the worst makespan are illustrated by the third, fourth and fifth column, respectively.

When only intensification strategy is taken into consideration, the improved algorithm is shown in Table 4-7. Table 4-5 and 4-7 present the results on the same bodies with different search strategies, respectively. The average makespan is improved by 1%. The best solutions are improved respectively by 1%. However, most of the worst solutions are significantly improved, and even some of them have further worsened.

The number of jobs	average time	average makespan	Best makespan	Worst makespan
5	0.7	184.7	178	189
10	0.7	338.6	319	342
20	1.0	601.2	584	612
50	1.3	1329.2	1296	1386
100	1.6	2558.6	2511	2648
200	2.3	5097.8	5046	5347

Table 4-7: Intensification Module Added

c.2. Diversification procedure

Set up another cooling factor β, after being run, Intensification module recalls a certain temperature, expands exploring space and reduces the probability for emergence of inferior solutions. Diversification module can be described as follows:

```
While (T_c<T_e)

{     Phase=1; g*=f(i); T=T_c;
While (phase<maxphase)
{
S*=intensification (i, j);
If (g*== S*) then {phase= phase-1 ;}
              Else {g*= S*; phase=1 ;}
T=T *  α ;}
T_c=T_c  *  β ;}
```

Diversification Strategy

Table 4-8 presents the results for the set of all instances in the cooling factor is 0.95 and the intensification module added; the conducted experiment witnessed 6 stages, where the sizes of problem successively are 5, 10, 20, 50, 100, and 200, For each size, 10 sets of data were selected at random in [1,100] and involved in the corresponding stage. The second column represents the average time of execution. The average, the best, and the worst solutions are illustrated by the third, fourth and fifth column, respectively.

When only diversification strategy is taken into consideration, the improved

algorithm is shown in Table 4-8. Table 4-5 and 4-8 present the results on the same bodies with different search strategies, respectively. The average makespan is improved by 0-1%. The best and worst solutions are improved by respectively 0.7 % and 2 %.

The number of jobs	average time	average makespan	Best makespan	Worst makespan
5	0.4	184.9	178	189
10	0.5	338.6	319	352
20	0.6	601.2	584	607
50	0.7	1329.2	1296	1346
100	0.8	2558.6	2511	2589
200	1.0	5134.8	5058	5276

Table 4-8: Diversification Module Added

c.3. Intensification And Diversification Procedures

When intensification and diversification strategy are taken into consideration, Table 4-9 presents the results for the set of all instances in the cooling factor is 0.95 and the intensification and diversification module added; the conducted experiment witnessed 6 stages, where the sizes of problem successively are 5, 10, 20, 50, 100, and 200, For each size, 10 sets of data were selected at random in [1,100] and involved in the corresponding stage. The second column represents the average time of execution. The average, the best, and the worst makespan are illustrated by the third, fourth and fifth column, respectively.

While the cooling factor is 0.95 and the intensification and diversification module is added, Table 4-5 and 4-9 present the results on the same bodies with different search strategies, respectively. The average makespan is improved by 2%, respectively. The best and worst solutions are improved by respectively 1.4 % and 4.2 %. The improvement of the simulated annealing approach has an obvious effect.

The number of jobs	Average time	Average makespan	Best makespan	Worst makespan
5	0.7	179.8	178	181
10	0.7	337.3	319	342
20	1.1	595.8	584	598
50	1.3	1311.5	1296	1321
100	1.7	2517.7	2511	2523
200	2.4	5068.7	5023	5167

Table 4-9: Intensification and Diversification Module Added

4.4.1.2 Tabu search

The basic principle of TS is based on classical Local Search methods (LS) improvement techniques and to overcome local optimal by crossing boundaries of feasibility. In this section, our goal is to improve the TS algorithm through an adjusting search strategy.

1. Basic algorithm

The basic version of the tabu search algorithm can be described as follows:

```
Obtain a random initial sequences f(i)
Clear up the Tabu list;
Repeat
    Select a new minimum sequence f(j) in the neighborhood of f(i);
    If f(j)<best_to_far then
    begin
    f(i)=f(j);
    let f(j) take place of the oldest sequence in the Tabu list;
    best_to_far:=f(j);
    end else
            begin
            if f(j) is not in the Tabu list then
               f(i)=f(j);
               f(j) take place of the oldest sequence in the Tabu list;
            end;
until (termination-condition);
```

Basic Tabu Search Process

The basic idea is to calculate all possible sequences of the current neighborhood and find the one with the best makespan. The cardinality of the neighborhood of each sequence is $N(N-1)$. Hence, as the size of the problem increases, the running time of the algorithm becomes difficult to accept. In addition, when the iterative time is bigger than 300, the possibility that the best sequence changes is very small. So in next section, we use a more flexible structure which is as follows:

1. Strategy of neighborhood search

   a. Generate $\frac{n}{2}$ permutations for the correct sequence through $\frac{n}{2}$ different random swaps;

   b. Calculate the value of every sequence;

   c. Arrange the sequence in decreasing order of the makespan and insert them into the candidate list.

2. Tabu size is half of all jobs.

3. This process is repeated 500 times.

The flexible tabu search algorithm can be described as follows:

```
Initialize a random sequence S; clear up the Tabu list
Result=f(s);
Best= result;
Time =0;
Repeat
   Generate n/2 random swaps to obtain a set of candidate sequences;
   Arrange candidate swaps to Tabu list V* by order;
   f(s) =min f (V*);
   If f(s) is best _to _far then
         Begin
         Result=f(s);
         Best= result;
         Let f(s) take place of the oldest sequence in the Tabu list;
         End
   Else
        If f(j) is not in the Tabu list then
            Begin
              Result=f(s);
              Let f(s) take place of the oldest sequence in the Tabu list;
            End;
    Time =time +1;
    Update Tabu list;
 Until Time >>00;
Result=min (result, best);
```

Flexible Tabu Search Process

In addition to the specific strategy of the algorithm, tabu search includes the same calculation module with simulated annealing algorithm (see Section 4.4.1). First, let us compare the two structures as follows:

The conducted experiment witnessed 6 stages, where the sizes of the problem successively are 5, 10, 20, 50, 100, and 200, For each size, 10 sets of data were generated at random in [1,100]. The second column represents the average running time. The average, the best, and the worst makespan are illustrated by the third, fourth and fifth column, respectively.

The algorithm was coded in Visual C++ 6.0 and tested on a personnel computer with a 1.66 GHz Intel® Core™ Duo CPU on the MS Windows XP operating system.

The number of jobs	Average time	Average makespan	Best makespan	Worst makespan
5	1.5	184.6	178	186
10	6.5	334.6	319	342
20	30.4	602.3	592	606
50	103	1337.2	1320	1378
100	3250	2587.3	2513	2615
200	>5000	.....	.....	.....

Table 4-10: Results for the Basic Tabu Search

The number of jobs	Average time	Average makespan	Best makespan	Worst makespan
5	1.1	184.7	178	189
10	4. 5	335.6	319	345
20	20.4	601.3	590	608
50	60.5	1345.2	1320	1389
100	200. 4	2592.3	2511	2623
200	800.2	5188.8	5097	5400

Table 4-11: Results for the Flexible Tabu Search

We have found, from the comparison of the two structures, that, in general, the overall quality of the solutions of the flexible tabu search is lower than that of the results of the basic tabu search. This indicates the absence of stability of both algorithms, and there is a clear gap between the best makespan and the worst makespan of the two algorithms. However, the flexible tabu search is time consuming, especially when the number of jobs is more than 200. Consequently, apart from reserving the advantage of the flexible tabu search that it does not take much time, both the quality and the stability of its solutions needs further improvements. Compared with the flexible structure, the quality of solution has been a little affected, but when N>100, the running time of the basic TS algorithm is higher. So, we chose the more flexible structure to the preliminary.

2. Improvement experiments

In terms of different problems, tabu search algorithm needs to make corresponding adjustments. Generally, tabu search algorithm includes three factors: the initial sequence, the size of the tabu list, and a diversification and intensification procedures.

a. Initial sequence

Tabu search repeats choosing the best neighborhood never visited from the neighborhoods of the current sequence. Therefore, if the initial sequence is relatively good, then it is likely to make the quality of solution better. So, we use either the current initial sequence or the result of a greedy algorithm as the initial sequence of tabu search.

Step 1: $t_1=0; t_2=0$ (t_1 and t_2 are the completion times of the last processed job on M1 and M2, respectively);

Step 2: if ($t_1 \le t_2$) the next job is processed on M1 else on M2;

Step 3: On M1 (M2), the job in the unprocessed sequence with the smallest waiting time is processed. With equal waiting times, the longest job is preferred for processing;

Step 4: According to step 3, the smallest job will be processed on M1 (M2).

Step 5: Update t_i and repeat Step 2 to Step 5

A Greedy Algorithm

The conducted experiment witnessed 6 stages, where the sizes of problem successively are 5, 10, 20, 50, 100, and 200, For each size, 10 sets of data were selected at random in [1,100] and involved in the corresponding stage. The second column represents the average time of execution. The average, the best, and the worst solutions are illustrated by the third, fourth and fifth column, respectively.

The algorithm was coded in Visual C++ 6.0 and the tested on a computer with a 1.66 GHz Intel® Core™ Duo CPU on the MS Windows XP operating system.

The number of jobs	Average time	Average makespan	Best makespan	Worst makespan
5	1.1	184.7	178	189
10	4.6	335.5	319	345
20	20.5	602.1	590	605
50	61	1346.4	1320	1391
100	201	2590.1	2512	2619
200	802	5186.5	5097	5403

Table 4-12: Initial Sequence for the Random Sequence

The number of jobs	Average time	Average makespan	Best makespan	Worst makespan
5	1.1	184.6	178	188
10	4.6	334.9	319	342
20	20.5	603.4	590	608
50	61	1345.8	1320	1394
100	201	2589.3	2512	2617
200	802	5190.6	5097	5409

Table 4-13: Initial Sequence Produced By the Greedy Algorithm

Compared with Table 12 and 13, it is indicated from the experiment result that in terms of the current algorithm, there is no necessary links between the initial value and the result quality. Therefore, the initial sequences of the following algorithms will uniformly be taken by means of random algorithms.

b. Size of the tabu list

A tabu list is used to store tabu candidate swaps to avoid repeating exploring the same neighborhood to enter a deadlock state. Therefore, if the tabu list is too short, obviously it is not enough to avoid the repetition. However, if the tabu list is too long, it is likely that some neighborhoods miss being visited due to being tabooed. Simultaneously, the size of the tabu list and the number of jobs are closely related.

It is indicated from the experimental results, with the processing scale changing, that the size of the tabu list needs appropriately expanding. Therefore, for the following tabu search algorithms, the size of the tabu lists is $N/2$.

c. Intensification module and diversification module

Tabu search repeats by choosing the best neighborhood never visited from the neighborhoods of the current sequence. There exist two problems with this strategy:

- The quality of random candidate sequences cannot be guaranteed.
- The other drawback is its lack of flexibility, deadlocks are likely to occur.

Therefore, intensification module can be used to accelerate and strengthen the exploration. However, to avoid a deadlock, the diversification module needs to be used to timely pioneer new exploring fields.

c.1. Intensification procedure

The neighborhood of the selected sequence is further explored and this process is repeated.

1 Record the current sequence;
2 Insert job k in other n-1 jobs to produce a set of new sequence; $k \in \{1,2,...,n\}$;
3 Find the shortest sequence within this set;
4 Repeat 1-3 until the number of the shortest sequences is equal to $\frac{n}{2}$;

Intensification Strategy

The conducted experiment witnessed 6 stages, where the sizes of problem successively are 5, 10, 20, 50, 100, and 200. For each size, 10 sets of data were generated at random in [1,100]. The second column represents the average time of execution. The average, the best, and the worst makespan are illustrated by the third, fourth and fifth column, respectively.

When only intensification strategy is taken into consideration, the improved algorithm is shown in Table 14. Table 4-12 and 4-14 present the results on same

bodies with different search strategies, respectively; the average makespan is improved by 2%, respectively. The best solutions are improved respectively by 0-2%. The parts of the worst solutions are improved by 0-3%, respectively. However, the rest of them decreased by 1-2%, respectively. Because the frequency of the iteration is fixed, the running time has increased by 20%.

The number of jobs	average time	average makespan	Best makespan	Worst makespan
5	1.7	183.6	178	188
10	7.0	331.7	319	346
20	30	598.9	586	605
50	88.5	1330.6	1304	1408
100	264	2569.9	2512	2596
200	995	5148.2	5037	5432

Table 4-14: Results for the Intensification Module

c.2. Diversification procedure

If the result of exploring neighborhoods has not been improved for long, the algorithm may enter into a deadlock state. For this reason, a new sequence is randomly selected as the initial sequence, and a new exploration starts over again.

1	Save the best sequence;
2	Generate at random an initial sequence;
3	Regenerate $\frac{N}{2}$ random permutations;
4	Repeat the strategy of intensification;

Diversification Strategy

The conducted experiment witnessed 6 stages, where the sizes of problem successively are 5, 10, 20, 50, 100, and 200. For each size, 10 sets of data were generated at random from [1,100]. The second column represents the average time of execution. The average, the best, and the worst makespan are illustrated by the third, fourth and fifth column, respectively.

When only the diversification strategy is taken into consideration, the improved algorithm is shown in Table 15. Table 4-12 and 4-15 present the results on the same

bodies with different search strategies, respectively; the average makespan is improved by 1-3%. The best and worst solutions are improved respectively by 0-1% and 3 %. The running time has not obviously changed.

The number of jobs	average time	average makespan	Best makespan	Worst makespan
5	1.2	182.6	178	184
10	4.8	332.6	319	335
20	22.1	596.8	586	597
50	64.2	1339.6	1308	1398
100	209	2543.4	2512	2598
200	821	5169.4	5076	5312

Table 4-15: Results for the Diversification Module

c.3. Intensification and diversification procedure

When intensification and diversification strategies are taken into consideration, the improved algorithm is shown in Table 4-16. The conducted experiment witnessed 6 stages, where the sizes of the problem successively are 5, 10, 20, 50, 100, and 200. For each size, 10 sets of data were generated at random from [1,100]. The second column represents the average execution time. The average, the best, and the worst makespan are illustrated by the third, fourth and fifth column, respectively.

Table 4-12 and 4-16 present the results on the same bodies with different search strategies, and the average makespan is improved by 5 %. The best and worst solutions are improved respectively by 0-2 % and 5 %. However, the number of iterations increased quite significantly. So, the running time goes up by 20%.

The number of jobs	Average time	Average makespan	Best makespan	Worst makespan
5	1.7	179	178	180
10	7.1	321	319	323
20	30.3	593.7	584	597
50	89.5	1299.7	1296	1345
100	267.4	2513.2	2511	2515
200	1000.8	5012.2	4965	5124

Table 4-16: Results for Intensification and Diversification Module

Compared with Simulated Annealing algorithm (Table 4-9), the average makespan of tabu search is improved by 1-2%. The worst solutions improved by 3%. But the "cost" of the running time is beyond comparison. In other words, the result of the tabu search algorithm is more accurate and stable with the increase of the size of problem. But, the running time may be unbearable. So, we hope to find an algorithm that has both of the two advantages. This is the hybrid algorithms (tabu search and simulated annealing) that we present in the next section.

4.4.2 A Hybrid Algorithm

In this section, we propose a hybrid algorithm based on the concepts borrowed from tabu search and simulated annealing to solve the two-machine open shop problem with time delays. This algorithm, called tabu-simulated-annealing, is a combination of the tabu search algorithm with the cooling rule of simulated annealing.

4.4.2.1 Basic idea of the hybrid algorithm

The advantages of simulated annealing (SA) are simple and rapid and the ability to provide reasonably good solutions for same problem. The weaknesses of simulated annealing are that it is likely to enter a deadlock state and different solutions might be obtained for one problem and greatly differentiated (low stability). The advantages of tabu search algorithm are that it can avoid deadlock through the tabu list, and it has high stability where there are small differences between the best

and the worst situations. It is not difficult to find the complementarities of the two algorithms. Based on this point, we have built a new algorithm based on the combination of the advantages of the two algorithms. The new algorithm was investigated in two stages.

- **First stage**

To use the simulated annealing to choose *k* candidate swaps in the memory list. Where k is a constant and along with the size of problem increased, k can be advisable improved. This stage has two advantages: the range of candidate swaps is global search and the size of memory list is smaller.

- **Second stage**

1. Adjust the temperature through the cooling factor for gradually reducing the accepting probability of inferior result.
2. Through a tabu list, enhance the range of candidate swaps (out of deadlock).

The pseudo code is described as below:

```
I=random sequence;
f*= f(i);
Initial temperature T*= T0; Terminated temperature Tj *= Tj;
Cooling factor α*=α;
Repeat:
  For i=1 to 5
  Begin
    Generate a random neighbor f (j) from f (i);
    Δf=f (j) –f (i);
    If Δf<=0 then accept the new sequence to memory list
    Else
        If Δf >0 then
          Get a random number h∈(0,1);
          If h<exp (–Δf/T)
          Then accept the new sequence to memory list
    End;
    T =α T;
    To select a best in memory list
     If prohibited and not best so far to repeat to select the other
      Else the result→ (f*= f (j); f (i) =f (j))
```

A hybrid Algorithm

In addition to the specific strategy of the algorithm, the hybrid algorithm includes the same calculation module with simulated annealing algorithm. (See calculation module in Section 4.4.1)

4.4.2.2. Experimental Results With The Hybrid Algorithm

At first, we generate a random initial sequence. Initial temperature T_0 is 600. Terminated temperature T_j is 0.01. The cooling factor is 0.95; the size of the tabu list is N/2. The conducted experiment witnessed 4 stages, where the sizes of the problem successively are 20, 50, 100, and 200. For each size, 10 sets of data were generated at random from [1,100]. The second column represents the average time of execution. The average, the best, and the worst makespan are illustrated by the third, fourth and fifth column, respectively.

The number of jobs	Average time	Average makespan	Best makespan	Worst makespan
20	2.3	594.2	584	597
50	2.5	1300.6	1296	1348
100	2.5	2511.6	2511	2517
200	2.6	5013.2	4965	5129

Table 4-17: Results Produced by the Hybrid Algorithm

We may observe through the comparison of Table 4-9 and Table 4-16 with Table 4-17, the new hybrid algorithm combines the advantages of the tabu search and simulated annealing algorithms and to a certain extent makes up for their disadvantages. The gap between the mean value of the solution of the tabu search and that of the new algorithm is $< 0.1\%$, but its running time is greatly reduced. In contrast with simulated annealing algorithm, the difference of the best and the worst situations is $< 0.4\%$, and the stability of the solutions has been significantly improved. Meanwhile, it is indicated from the tests that besides only improving the framework of an algorithm, it is also a quite effective way to jump out of the current framework to combine the advantages of some other algorithms.

General Conclusion

The problem we have studied in this thesis is the two-machine open shop problem with time delays. The open shop scheduling problem has a much larger solution space than the job shop or flow shop scheduling problems because, as for the open shop model, no restrictions are placed on the processing order on the jobs. But little attention is paid to it by researchers and practitioners, primarily because of the limitation of traditional applications. In recent years, with technology innovation and scientific management, the open shop scheduling model began to come under an increasing attention, as it can be seen from the growth rate of relevant studies published. This demonstrates that more importance will be attached to this problem in the future.

After an introductory chapter, the problems under study are presented in Chapter 2, which is an overall description for scheduling problems. At first, the basic concepts of some necessary knowledge contain various scheduling models, Gantt chart, three field notations, and so on. Second, due to the fact that most of scheduling problems are NP-hard, a brief introduction is made for some basic concepts of the complexity theory, which is followed by an overview on how to tackle the resolution of a scheduling problem, along with a highlighted introduction to the worst-case analysis and probabilistic analysis. At last, we went on a detailed description on some common algorithms (such as tabu search, simulated annealing and branch and bound algorithms).

Chapter 3 describes the open-shop problem without time delays. We presented the Gonzalez-Sahni algorithm and the LAPT algorithm to solve the $O_2 \parallel C_{\max}$ problem. We designed a new way of stating the LAPT algorithm and proved also its optimality. Furthermore, we compared the two algorithms and undertook an experimental study.

Chapter 4 describes the open shop problem with time delays in two major parts. The first part presents several lower bounds. In the second part, we reviewed the literature and discussed heuristic algorithms and meta-heuristic algorithms. For the

heuristic approach, we derived better results in some special cases. We can use the analytic approach and empirical approach to measure the quality of algorithm. For the meta-heuristic approach, although the framework is the same, we can still make adjustments of the basic strategy to improve significantly the efficiency of the resulting algorithm that include stopping criteria, internal structure (framework) and hybrid algorithm. For the stopping criteria, we have mainly verified the role of the lower bounds in the meta-heuristic approach. For the internal structure, we gradually improved the quality of several algorithms an experimental experiment. Regarding the simulated Annealing algorithm, we started with a randomly generated initial solution, and the cool factor is respectively 0.8, 0.9, 0.95, and 0.99. The experimental results show that when the cool factor is 0.95, generally, the result of the algorithm is much better. When the intensification and diversification strategies are used in simulated annealing, the result of the algorithm is more accurate and stable. As for the tabu search algorithm, the basic algorithm is time consuming, we designed a more flexible neighborhood structure and the result show that its efficiency improved significantly particularly for large size inputs. When the intensification and diversification strategies are used, the improvement of the algorithm is drastic. However, when the instances of problem get larger, the running time of the algorithm gets more and more prohibited. Finally, a hybrid algorithm is proposed, based on the concepts borrowed from tabu search and simulated annealing methods to solve the two-machine open shop problem with time delays. We generated a random initial sequence. The cooling factor is 0.95 and the size of the tabu list is 5. The experimental results show that the result is more accurate than simulated annealing, and the running time is less than that of the tabu search algorithm.

The ideal of meta-heuristic algorithm in our view should be that, on the one hand, the accuracy of the results is higher, while simultaneously, on the other hand, the running time is less time consuming. However, as for the two-machine open shop problem with time delays, most of the original meta-heuristic algorithms can at most achieve one of above two performances. From the above experimental results, the

tabu search algorithm has an advantage in accuracy and stability front, whereas the simulated annealing algorithm has an advantage in the running time front. However, when intensification and diversification strategies are added, the defects of the algorithms are greatly remedied while keeping their original features. Additionally, it is also a good orientation for the algorithm improvement to combine algorithms like tabu search and simulated annealing whose advantages are complementary to each other into a new hybrid algorithm. Therefore, intensification and diversification procedures and hybrid algorithms is an important orientation for the improvement of meta-heuristic algorithms and also where breakthrough will be.

Rferences

[Alcaide *et al.*, 1997] Alcaide, D, Sicilia, J., Vigo, D. Heuristic approaches for the minimum makespan open shop problem. *Journal of the Spanish Operational Research Society,* 1997, Vol. 5, pp. 283–296.

[Blazewicz *et al.*, 2004] Blazewicz, J., Brauner, N., and Finke, G. Scheduling with Discrete Resource Constraints, *In Lueng, 2004 chapter 23.*

[Danzig Fulkerson and Johnson 1995] Dantzig, G.B., Fulkerson, D.R., Johnson, M. On a Linear-Programming, Combinatorial Approach to the Traveling-Salesman Problem, *Operations Research,* 1995, pp. 58-66.

[Fang *et al.* 1994] Fang, H.L., Ross, P., Corne, D. A Promising Hybrid GA/Heuristic Approach for Open-Shop Scheduling Problems. *ECAI,* 1994, pp. 590-594.

[Gantt,1916] Henry Laurence Gantt Work, Wages, and Profits, second edition, *Engineering Magazine Co,* New York, 1916.

[Graham et al., 1979] Graham,R.L., Lawler, E.L., Lenstra, J.K., and Rinnoy Kan, A.H.G. Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics,* 1979, Vol 5, pp. 287-326.

[Glover, 1986] Glover, F. Future Paths for Integer Programming and Links to Artificial Intelligence. *Comput. & Ops. Res.* 1986, Vol. 13, No.5, pp. 533-549.

[Glover, 1989] Glover F. Tabu search—Part I. *ORSA Journal on Computing 1989,* Vol. 1, No. 3, pp. 190-206.

[Glover, 1990] Glover F. Tabu search—Part II. *ORSA Journal on Computing* 1990, Vol. 2, pp. 4–32.

[Glover, 1997] Glover, F. and M. Laguna. *Tabu Search.* Kluwer, Norwell, MA, 1997.

[Gonzalez and Sahni 1976] Gonzalez T., Sahni, S. Open shop scheduling to minimize finish time, *Journal of the Association for Computing Machinery,* 1976, Vol. 23, pp. 665–679.

[Hansen, 1986] Hansen. P. The Steepest Ascent, Mildest Descent Heuristic for Combinatorial Programming. *Congrès sur les Méthodes Numériques en Optimisation, Capri, Italie,* 1986.

[Holland ,1975] Holland, JH. Adaptation in Natural and Artificial Systems, *University of Michigan Press,* Ann Arbor, 1975.

[Johnson, S.M,1954] Johnson, SM. Optimal two- and three-stage production schedules with set-up times included. *Naval Research Logistics Quarterly,* 1954, Vol.1, Issue.1, pp. 61-68.

[Kirkpatrick et al., 1983] Kirkpatrick, S., Gelatt, CD., Vecchi, MP. Optimization by Simulated Annealing. *Science,* 1982, Vol. **220** (4598), pp. 671–680.

[Liaw 1999a] Liaw, CF. A Tabu search algorithm for the open shop scheduling problem. *Computers and Operations Research,* 1999, Vol. 26, pp. 109–126.

[Liaw 1999b] Liaw, CF. Applying simulated-annealing to the open shop scheduling problem, Working paper, *Department of Industrial Engineering and Management, Chaoyang University of Technology,* Taiwan, 1999.

[Liaw 2000] Liaw, CF. A hybrid genetic algorithm for the open shop scheduling problem, *European Journal of Operational Research* , 2000, pp. 28–42.

[Liaw 2003] Liaw, CF. An efficient Tabu search approach for the two-machine preemptive open shop scheduling problem, *Computers and Operations Research, 2003,* Vol. **30**, pp. 2081–2095.

[Metropolis, 1953] Metropolis, N., Rosenbluth, AW., Rosenbluth, Teller, N.M., A. H., Teller, E. 1953. Equation of state calculations by fast computing machines. *J. Chemical*

Physics, Vol. 21, pp. 1087-1091.

[Fibich *et al*, 2005] Fibich P, Matyska L., Rudova H. Model of Scheduling Problem. In Proceedings of the Workshop on Exploring Planning and Scheduling for Web Services, *Grid and Autonomic Computing. 1. vyd. Pittsburgh: AAAI Press* Technical Report, 2005, pp. 235-242.

[Pinedo 1995] Pinedo, M. Scheduling: *Theory, Algorithms, and Systems,* Prentice-Hall, Englewood Cliffs, NJ, 1995.

[Pinedo and Schrage, 1982] Pinedo M, Schrage L Stochastic shop scheduling: a survey. *In:* Dempster MAH, Lawler EL, Lenstra JK, Rinnooy Kan AHG (eds.) *Deterministic and Stochastic Scheduling.* Riedel, Dordrecht, 1982, pp. 181–196.

[Rebaine and Strusevich, 1999] Rebaine, D, Strusevich, V.A. Two-machine open shop scheduling with special transportation times, *The Journal of the Operational Research Society,* 1999, Vol. 50, No. 7, pp 756-764.

[Rebaine, 2004] Rebaine, D. Scheduling the two-machine open shop problem with non-symmetric time delays. In *Congress ASAC 2004,* Quebec, Canada.

[Strusevich 1999] Strusevich, VA, van de Waart, AJA, Dekker, R. A 3/2 Algorithm for Two-Machine Open Shop with Route-Dependent Processing Times. *Journal of Heuristics,* 1999, Vol. 5, pp. 5-28.

[Yu 1996] Yu, W. The two-machine flow shop with delays and the one machine total tardiness problem, *Ph.D. thesis, Department of Mathematics and Computing, Eindhoven University of Technology,* The Netherlands, 1996.