2006

Empirical study of dense schedule performance ratio on open-shop scheduling problem

Yan, Qingxiang

Empirical Study of Dense Schedule Performance Ratio on Open-shop Scheduling Problem

by

Qingxiang Yan

A thesis

presented to Lakehead University

in fulfillment of the

thesis requirements for the degree of
Master of Science
in
Mathematics

Thunder Bay, Ontario, Canada, 2006

Your file Votre référence
ISBN: 978-0-494-24066-3
Our file Notre référence
ISBN: 978-0-494-24066-3

Abstract

In this paper, we study properties of dense schedules for the open-shop problems and their average performance ratio. After using two sets of test problems, we show that the average performance ratio of dense schedules is actually much better than $(2-\frac{1}{m})$, the worst-case performance ratio in the conjecture. The results from randomly generated problems which have large sizes show that when the dimension of open-shop problems become larger, the average performance ratio is getting even smaller. Twelve heuristic algorithms to generate dense schedules are presented in Chapter 3 and the computational results of two sets of test problems are also provided.

Acknowledgments

I would like to thank Dr. Wendy Huang, Dr. Deli Li and Dr. Tianxuan Miao for all the time and effort they have spent on my education, and the kindness and friendship they have shown me.

Contents

Chapter 1

Introduction

1.1 Open-Shop Scheduling Problem

The open-shop problem may be stated as follows. There are n jobs J_1, J_2,...,J_n and m machines M_1, M_2,...,M_m. Every job has m operations, each of which has to be processed on a specified machine for a given duration time. The operation can be processed in any sequence, and as long as all the operations needed for the job are done, the job is done. We assume that at any time t, at most one job can be processed on each machine, and each job can be processed on at most one machine. In this paper, we only consider those cases that preemption is not allowed, that is, all operations must be processed without interruption. The objective is to find a schedule that minimizes the makespan C_{max}, the time from the beginning of the first operation to the end of the last operation.

1.2 Computational Complexity

1.2.1 *P* and *NP*

Practical experience tells us that some problems are easier to solve than others. To classify problems as "easy" or "hard", we should introduce Complexity Theory.

A computational problem can be viewed as a function h that maps each input x in some given domain to an output $h(x)$ in some given range. We consider algorithms that compute $h(x)$ for each input x. One of the main issues of complexity theory is to measure the performance of algorithms with respect to

computational time. To be more precise, for each input x we define the input length $|x|$ as the length of some on coding of x. Then we measure the efficiency of an algorithm by an upper bound $T(n)$ on the number of steps that the algorithm takes for any input x with $|x| = n$. Usually, it is difficult to calculate the precise form of T. For these reasons we will replace the precise form of T by its asymptotic order. Therefore, we say that $T(n) \in O(g(n))$ if there exist constants $c > 0$ and a nonnegative integer n_0 such that $T(n) \leq cg(n)$ for all integers $n \geq n_0$, where $g(n)$ is a function that has the same order as $T(n)$.

A problem is called **polynomially solvable** if there exists a polynomial p such that $T(|x|) \in O(p(|x|))$ for all inputs x for the problem, i.e. if there is a k such that $T(|x|) \in O(|x|^k)$.

The notion polynomially solvable depends on the encoding. We assume that all numerical data describing the problem are binary encoded. An algorithm is called **pseudopolynomial** if $T(n)$ is polynomial where n is the input length with respect to unary encoding. A problem is called **pseudopoly-nomially solvable** if there exists a pseudopolynomial algorithm which solves the problem.

A problem is called a **decision problem** if the output range is {yes, no}. We may associate with each scheduling problem a decision problem by defining a threshold k for the corresponding objective function f. This decision problem is: Does there exist a feasible schedule S such that $f(S) \leq k$?

The class of all decision problems which are polynomially solvable is denoted by P.

When a optimization problem is formulated as a decision problem there is an important asymmetry between those inputs whose output is "yes" and those whose output is "no". A "yes" -answer can be certified by a small amount of information: the feasible schedule S with $f(s) \leq k$. Given this certificate, the "yes" -answer can be verified in polynomial time. This is not the case for the "no" -answer. In general, let NP denote the class of decision problems where each "yes" input x has a certificate y, such that $|y|$ is bounded by a polynomial in $|x|$ and there is a polynomial-time algorithm to verify that y is a valid certificate for x. (For detail discussion of complexity theory see [6].)

1.2.2 *NP*-complete and *NP*-hard Problems

The principal notion in defining *NP* –completeness is that of a **reduction**. For two decision problems P and Q, we say that P reduces to Q (denoted $P \propto Q$) if there exists a polynomial-time computable function y that transforms inputs for P into inputs for Q such that x is a "yes" -input for P if and only if $y(x)$ is a "yes" -input for Q.

A decision problem Q is called *NP*-**complete** if $Q \in NP$ and, for all other decision problems $P \in NP$, we have $P \propto Q$.

The following lemma provides us a straightforward approach for proving new problems to be *NP*-complete.

Lemma 1. If P and Q belong to *NP*, P is *NP*-complete, and $P \propto Q$, then Q is *NP*-complete.

An optimization problem is called *NP*-**hard** if the corresponding decision problem is *NP*-complete.

For the open-shop problem that is described in Section 1.1, when $m = 2$, a polynomial time algorithm is proposed by Gonzalez & Sahni [7]. Recently, Pinedo [8] presented another simple dispatching rule: Longest Alternate Processing Time first (LAPT) which also solves this problem in polynomial time. However, from $m \geqslant 3$, many open shop scheduling problems are *NP*-complete. (Gonzalez & Sahni [7]).

Algorithm designers have developed several approaches to deal with *NP*-hard problems, such as the branch & bound algorithms (Brucker [1]). A branch-and-bound algorithm is based on the idea of intelligently enumerating all feasible solutions. Computational results show that these methods find optimal solutions in reasonable time for small to medium size problems.

1.3 Performance Ratio

Since most scheduling problems are *NP*-hard, it is usually difficult to find the optimum. As alternatives, in many practical situations, we try to find the approximated solutions that are guaranteed to have the objective value within a fixed percentage of optimal value. Algorithms that provide such solutions are approximation algorithms. For a scheduling problem that minimizes $F(\cdot) \geq 0$, an

algorithm H is considered an r-approximation algorithm ($r > 1$) if for all instances I of this problem, $F(H(I)) \leq rF(S^*(I))$, where $H(I)$ and $S^*(I)$ are solutions provided by algorithm H and the optimum of this problem, respectively. The smallest r is called **the best worst-case performance ratio** of algorithm H.

Chapter 2
Empirical Study of Dense Schedule Performance Ratio

Although the branch and bound method referred to in previous sections is computationally more efficient than simple exhaustive enumeration, for a large number of machines and jobs it still requires high computational time and effort. Therefore, most of the real-life problems are solved by **heuristic** methods. According to Peter Brucker[6], a heuristic method is an approach without formal guarantee of performance. Dense schedules can be used as heuristic solutions to open shop problems. In this Chapter we will put our focus on dense schedules and see how good a dense schedule can be.

2.1 Dense Schedule and Its Properties

2.1.1 Dense Schedule

In a schedule S of a open-shop scheduling problem, there might be an idle time interval Q from time b to time c (denoted as $Q[b,c)$) on machine M_i. Q is rational if all jobs, if any, that are needed to be processed on machine M_i after time c are being processed on other machines during any time in Q. A **dense schedule** is a schedule S in which all idle time intervals are rational.

2.1.2 Relationship between Dense Schedule and Optimal Schedule

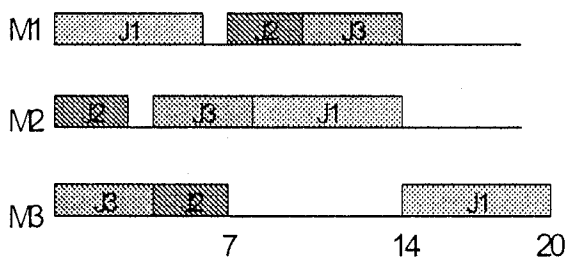
Let D be the set of all dense schedules to an open-shop problem, and let O be its optimal schedule set. We would like to see $D \cap O \neq \emptyset$, which would mean that we could concentrate only on studying

the set of dense schedules D since we would be able to find optimal schedules to any open-shop problem from its dense schedule set D. Unfortunately, this is not necessarily the case. For some open-shop problems, $D \cap O = \emptyset$. Consider the following example.

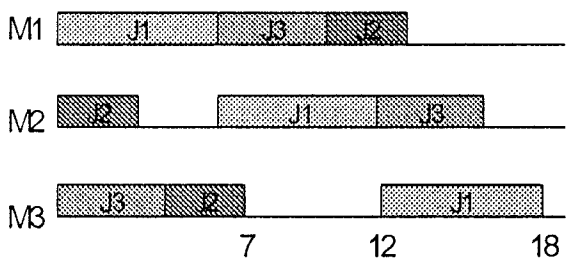
Example 1 Consider an open-shop problem with 3 machines and 3 jobs. The processing time of Job 1 is 6 for all operations, of Job 2 is 3 for all operations, and of Job 3 is 4 for all operations. We may use the following matrix P to express this information. Each entry P_{ij} is the processing time of the operation of Job j on Machine i.

$$p = \begin{bmatrix} 6 & 3 & 4 \\ 6 & 3 & 4 \\ 6 & 3 & 4 \end{bmatrix}$$

First of all, let us find all the possible dense schedules. Without loss of generality, we assign Job 1 on Machine 1, Job 2 on Machine 2 and Job 3 on Machine 3 at time 0. Following the definition of a dense schedule, the only schedule is shown in Graph 1 with the makespan of 20. (We could get other dense schedules by interchange the time-0 arrangement and their makespans are all 20).



Graph 1



Graph 2

A better schedule can be obtained in Graph 2, which is not dense (Since the idle time interval on M2 after Job2 is finished is not rational). This proves $D \cap O = \emptyset$ for this problem.

2.1.3 Performance Ratio of Dense Schedule

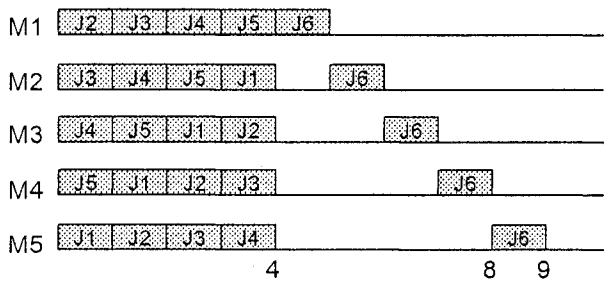
The concept of **dense schedule** was first introduced by Raczmany (see Barany and Fiala [9]), who

also indicated in 1982 that the makespan of a dense schedule to an open-shop scheduling problem is $C_{\max} \leq 2C_*$, where C_* is the optimum makespan. In other words, the performance ratio of a dense schedule is bounded above by 2. Later, Wein[2] and Chen and Strusevich[3] showed by example that this ratio could be $2 - \frac{1}{m}$. (See Example 2 below.) Chen and Strusevich[3] and Chen[4] have both presented this as a conjecture and proved the case when $m = 3$. This **conjectured performance ratio** has also been proved for $m = 5$ by Chen & Yu [5].

Example 2 Consider an open-shop problem with 5 machines and 6 jobs. The processing times of operations are shown below:

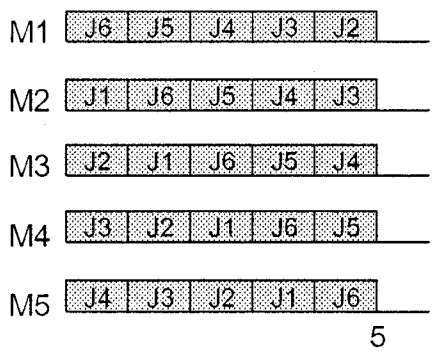
$$p = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

It is easy to check that the following schedule is dense, and the makespan C_{max} is 9.



Graph 3

Obviously, an optimal schedule can be given by the following and the makespan is 5.



Graph 4

This demonstrates that the performance ratio of a certain dense schedule can be $\frac{9}{5} = 2 - \frac{1}{5} = 2 - \frac{1}{m}$ for $m = 5$.

2.2 Average Performance of Dense Schedule

As referred to in Section 2.1.3, $(2 - \frac{1}{m})$ may be **the worst-case performance ratio** of a dense schedule. In this section, we will show that the average performance ratio of dense schedules is actually much better than $(2 - \frac{1}{m})$. Two sets of test problems are used to study the performance of dense schedules. The first set of problems consists of new benchmark problems given by Brucker et al. [1]. This set consists of 52 different problems. The second set of problems is randomly generated, which consists of 4300 different problems. The scheme of how these problems are generated will be explained in Section 2.2.2.1.

2.2.1 Study of New Benchmark Problems from Brucker

2.2.1.1 Introduction to New Benchmark Problems

As pointed out by Brucker et al. [1], those benchmarks from Taillard [10] are not really "hard" instances. Some measurements are defined and can be used to measure the "hardness" of instances.

1) LB: the trivial lower bound, i.e.

$$\text{LB} = \max(\{P_{J_j} | j = 1, \ldots, n\} \cup \{P_{M_i} | i = 1, \ldots, m\}),$$

Where P_{J_j} denotes the sum of processing times of the operations belonging to job J_j (i.e. $P_{J_j} = \sum_{i=1}^{m} P_{ij}$), $j = 1, \ldots, n$ and P_{M_i} denotes the sum of processing times of the operations which should be processed on machine M_i (i.e. $P_{M_i} = \sum_{j=1}^{n} P_{ij}$), $i = 1, \ldots, m$.

2) MIN: $MIN = \min(\{P_{J_j} | j = 1, \ldots, n\} \cup \{P_{M_i} | i = 1, \ldots, m\})$.

3) DIFF: MIN/LB.

4) WORKLOAD: reflects "average" workload on the machines for a schedule divided by lower bound LB, i.e.

$$\text{WORKLOAD} = \frac{\text{total processing time}}{m \cdot \text{LB}}.$$

If the WORKLOAD of an instance is close to 1, the processing times $\{P_{j_j}\}$ and $\{P_{M_i}\}$ are all close and within a small range, and the chance of finding a solution with $C_{\max}$-value close to LB will be rather small. On the other hand, if the WORKLOAD of an instance is small, there are only a few jobs or machines with processing times close to LB and the rest of them have processing times much smaller than LB, and one can expect to find a schedule with $C_{\max}$-value close or even equal to LB.

The new "hard" problems generated by Brucker et al. [1] are of dimension 3×3, 4×4, 5×5, 6×6, 7×7 and 8×8 with $LB = 1000$, $DIFF \in (0.9, 1)$ and $WORKLOAD \in (0.85, 1)$.

2.2.1.2 Procedure to Generate a Sample of Dense Schedules

We use the following procedure to find a sample for dense schedules of an open-shop problem. At

time 0, all the machines are free and all the jobs are available. For an open-shop problem with m machines and n jobs, there are different ways to assign jobs to machines. From each different way, we will sample one dense schedule. After fixing the arrangement at time 0, when one machine becomes available again, we randomly select a job from the jobs that need to be processed on this machine and are available at that time and assign it to this machine. When all the jobs are completed, we will get a dense schedule. Since there are $m!$ different time-0 arrangements, we could obtain $m!$ different dense schedules and calculate the performance ratio of each dense schedule.

2.2.1.3 Computational Results

We provide results for new benchmark problems with dimension up to 6×6.

- Problems: names of new benchmark problems provided by Brucker [1].
- DIFF: the DIFF of the problem.
- WKLD: the WORKLOAD the problem.
- Opt: the optimal makespan provided by Brucker [1].
- N.O.Ds: Number of dense schedules generated.
- Min-PR: the minimum performance ratio of performance ratios of all sample dense schedules.
- Max-PR: the maximum performance ratio of performance ratios of all sample dense schedules.
- Ave-PR: the average performance ratio of performance ratios of all sample dense schedules.
- Con-PR: the conjectured performance ratio or, **the worst case performance ratio** $(2-\frac{1}{m})$.

Table 1

Problems	DIFF	WKLD	Opt	N.O.Ds	Min-PR	Max-PR	Ave-PR	Con-PR
j3-per0-1	1.000	1.000	1127	6	1.05	1.23	1.1470	1.6667
j3-per0-2	1.000	1.000	1084	6	1.00	1.01	1.0043	1.6667
j3-per10-0	0.900	0.9617	1131	6	1.01	1.08	1.0426	1.6667
j3-per10-1	0.900	0.9630	1069	6	1.00	1.24	1.1311	1.6667
j3-per10-2	0.900	0.9493	1053	6	1.00	1.20	1.1168	1.6667
j3-per20-0	0.800	0.870	1026	6	1.05	1.27	1.1439	1.6667
j3-per20-1	0.800	0.886	1000	6	1.08	1.26	1.1837	1.6667
j3-per20-2	0.800	0.8997	1000	6	1.03	1.19	1.1098	1.6667
j4-per0-0	1.000	1.000	1055	24	1.00	1.15	1.0828	1.75
j4-per0-1	1.000	1.000	1180	24	1.00	1.27	1.1018	1.75
j4-per0-2	1.000	1.000	1071	24	1.01	1.27	1.1575	1.75
j4-per10-0	0.900	0.9413	1041	24	1.04	1.30	1.1449	1.75
j4-per10-1	0.900	0.9683	1019	24	1.01	1.29	1.1361	1.75
j4-per10-2	0.900	0.9433	1000	24	1.00	1.31	1.1206	1.75
j4-per20-0	0.800	0.8898	1000	24	1.05	1.27	1.1543	1.75
j4-per20-1	0.800	0.9087	1004	24	1.04	1.31	1.1829	1.75
j4-per20-2	0.800	0.9087	1009	24	1.01	1.19	1.1076	1.75
j5-per0-0	1.000	1.000	1042	120	1.01	1.31	1.1705	1.8
j5-per0-1	1.000	1.000	1054	120	1.00	1.35	1.2140	1.8
j5-per0-2	1.000	1.000	1063	120	1.03	1.29	1.1654	1.8
j5-per10-0	0.900	0.948	1004	120	1.02	1.32	1.1812	1.8
j5-per10-1	0.900	0.9294	1002	120	1.01	1.29	1.1799	1.8
j5-per10-2	0.900	0.934	1006	120	1.03	1.27	1.1447	1.8
j5-per20-0	0.800	0.895	1000	120	1.02	1.35	1.1356	1.8
j5-per20-1	0.800	0.8844	1000	120	1.00	1.29	1.1593	1.8
j5-per20-2	0.800	0.911	1012	120	1.00	1.25	1.1222	1.8
j6-per0-0	1.000	1.000	1056	720	1.03	1.28	1.1340	1.83
j6-per0-1	1.000	1.000	1045	720	1.02	1.32	1.1642	1.83

Table 1 (Continued.)

Problems	DIFF	WKLD	Opt	N.O.Ds	Min-PR	Max-PR	Ave-PR	Con-PR
j6-per0-2	1.000	1.000	1063	720	1.02	1.30	1.1429	1.83
j6-per10-0	0.900	0.944	1005	720	1.02	1.32	1.1494	1.83
j6-per10-1	0.900	0.9492	1021	720	1.02	1.27	1.1371	1.83
j6-per10-2	0.900	0.9463	1012	720	1.01	1.35	1.1476	1.83
j6-per20-0	0.800	0.911	1000	720	1.02	1.27	1.1213	1.83
j6-per20-1	0.800	0.9045	1000	720	1.00	1.26	1.1201	1.83
j6-per20-2	0.800	0.8797	1000	720	1.00	1.30	1.1169	1.83

From Table 1, we can see that for the new benchmark problems, the average performance ratios of dense schedules are very close to one, much less than the conjectured performance ratios. In some cases, the minimum performance ratios are equal to one, which means the corresponding dense schedules are optimal.

2.2.2 Study of Random Generated Problems

The previous section gave us a general idea of the performance of a dense schedule on small size problems. In this section, we want to study the performance of dense schedule on problems with larger dimension. Section 2.2.2.1 introduce the scheme of generating our problems with larger sizes. The results are presented in Section 2.2.2.2.

2.2.2.1 The Scheme of Generating Problems

Our goal is to generate more general problems that not only have large dimension but have various DIFF and WORKLOAD levels as well. One approach is assuming P_{ij}, the duration of operation is normal distributed with various means and variance. The following program is written in Matlab:

```
function p=new_generate(m,a,b)
    for i=1:m
```

```
for j=1:m
  p(i,j)=normrnd(a,b);
  if p(i,j)<0
    p(i,j)=0;
  end
 end
end
```

where m (number of machines and jobs), a (mean) and b (variance) are parameters and they can have different values in order to obtain different problems. We let m = 5,50,100,150; a = 100; b = 10,20,30,40,50,60,70,80,90,100,150,200 and have generated 100 problems for each combination (m,a,b) and obtained 4300 different problems in total.

2.2.2.2 Computational Results

We randomly choose only one dense schedule for each problem using the procedure discussed in Section 2.2.1.2 without fixing the time-0 arrangement, that is, we randomly select a job to be assigned on a free machine from the jobs that need to be processed on this machine and are available at that time (each job has the same probability to be selected) from time 0 till all the jobs are completed. The results are shown below:

- m: the number of machines and jobs.
- N.O.P: Number of problems with the same number of machines and jobs.
- I.O.DIFF: the interval of DIFF of the corresponding problems.
- I.O.WKLD: the interval of WORKLOAD of the corresponding problems.
- Min-PR: the minimum performance ratio of performance ratios of all (N.O.P.) generated dense schedules.
- Max-PR: the maximum performance ratio of performance ratios of all (N.O.P.) generated dense schedules.
- Ave-PR: the average performance ratio of performance ratios of all (N.O.P.) generated dense schedules.

• Con-PR: the conjectured performance ratio or, **the worst case performance ratio** $(2-\frac{1}{m})$.

Table 2

m	N.O.P.	I.O.DIFF	I.O.WKLD	Min-PR	Max-PR	Ave-PR	Con-PR
5	1200	0.152~0.940	0.354~0.975	1.0000	1.3425	1.0712	1.8
50	1200	0.140~0.928	0.329~0.971	1.0004	1.0646	1.0161	1.98
100	1200	0.138~0.925	0.318~0.952	1.0008	1.0344	1.0088	1.99
150	700	0.139~0.902	0.208~0.925	1.0018	1.0217	1.0074	1.99

Since the optimal makespans are unknown, we use LB instead of C_* while calculating the performance ratio. We can see from Table 2 that the performance of a dense schedule is not affected by the dimension of problem. It is easy to verify that $\frac{C_{max}}{LB} \geq \frac{C_{max}}{C_*}$, therefore, the actual performance ratio could be even better.

Based on the above consideration and computation, we could conclude that the average performance ratio of dense schedule is excellent. Therefore, it is reasonable to develop some heuristic approaches to generate dense schedule.

Chapter 3

Algorithms to Generate Dense Schedules

In this chapter we will develop 12 different heuristic algorithms to generate dense schedules based on different criteria to select machines and jobs at each step, and we call them machine-job heuristic algorithms (because in these algorithms, we select a machine first, and then assign a job based on the selected machine). Some notation will be introduced in Section 3.1.1. Three different criteria to select a machine and four criteria to select a job are presented and explained in subsequent Sections 3.1.2 and Section 3.1.3. The general procedure of these heuristic algorithms is described in Section 3.1.4.

3.1 Twelve Algorithms to Generate Dense Schedules

3.1.1 Some Important Notations

J_j: the j^{th} job, $j = 1, 2, \ldots, n$.

M_i: the i^{th} machine, $i = 1, 2, \ldots, m$.

makespan: a variable used to calculate $C_{\max}$ of a given problem.

P: the processing time matrix. Each entry P_{ij} is the processing time of the operation of J_j on M_i. (See Section 2.1.2)

$$P = \begin{bmatrix} P_{11} & P_{12} & \ldots & P_{1n} \\ P_{21} & P_{22} & \ldots & P_{2n} \\ & \ldots & \ldots & \\ P_{m1} & P_{m2} & \ldots & P_{mn} \end{bmatrix}.$$

S: the start time matrix. Each entry S_{ij} is the start time of the operation of J_j on M_i. This matrix is the solution of our algorithm and we can use this to construct a dense schedule easily. For example, for an open shop with

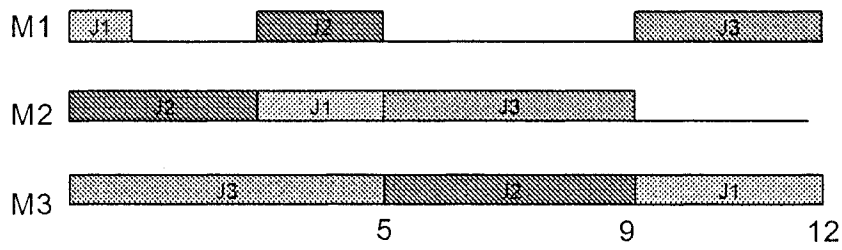
$$P = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix},$$

we may obtain a solution with:

$$S = \begin{bmatrix} 0 & 3 & 9 \\ 3 & 0 & 5 \\ 9 & 5 & 0 \end{bmatrix},$$

then the corresponding dense schedule is as follows:

	time 0	time 3	time 5	time 9
M_1	start J_1	start J_2		start J_3
M_2	start J_2	start J_1	start J_3	
M_3	start J_3		start J_2	start J_1



Graph 5

Also, at any time, the following variables are used to record the intermediate results.

AM: a $1 \times m$ vector, each entry AM_i shows the time needed for M_i to be available to process the next job;

AJ: similar to AM, is a $1 \times n$ vector. Each entry AJ_j shows the time needed for J_j to be available to be

processed on another machine;

RPM: a $1 \times m$ vector, each entry RPM_i shows the total remaining processing time (workload) of machine i. The initial value of RPM_i is the work-load for machine M_i,

$$RPM_i = \sum_{j=1}^{n} P_{ij} = P_{M_i};$$

RPJ: a $1 \times n$ vector, each entry RPJ_i shows the total remaining processing time of job j. The initial value of RPJ_i is the total processing time for job J_j,

$$RPJ_i = \sum_{i=1}^{m} P_{ij} = P_{J_j};$$

ROM: a $1 \times m$ vector, each entry ROM_i is the total number of operations on Machine i that have not been processed yet;

ROJ: a $1 \times n$ vector, each entry ROJ_i is the total number of operations of Job j that have not been processed yet;

total_operation: number of total operations that are not processed yet. Algorithms will stop if *total_operation* = 0.

3.1.2 Criteria to Select a Machine

a) Longest Remaining Processing Time Arrange the machines in the descending order of their remaining processing times and select one machine each time in this order.

b) Largest Number of Remaining Operations Arrange the machines in the descending order of their remaining numbers of operations that have not been done yet and select one machine each time in this order.

c) Natural Order Select one machine each time according to its index.

* When using Criteria a) or b), ties are broken by choosing the machine with the smallest index.

3.1.3 Criteria to Select a Job

1) Longest Remaining Processing Time Arrange the jobs in the descending order of their remaining processing times and select one job each time in this order.

2) Largest Number of Remaining Operations Arrange the jobs in the descending order of their remaining numbers of operations that have not been done yet and select one job each time in this order.

3) Shortest Processing Time (SPT) Arrange the jobs in the ascending order of the processing times of the operations that will be processed on the selected machine and select one job each time in this order.

4) Longest Processing Time (LPT) Arrange the jobs in the descending order of the processing times of the operations that will be processed on the selected machine and select one job each time in this order.

* Ties are broken by choosing the job with the smallest index.

3.1.4 General Procedure of Machine-Job Heuristic Algorithms

The steps of the machine-job heuristic algorithm are as follows:

Step 1. For a given open-shop problem, P, calculate initial RPM, RPJ, ROM and ROJ , and calculate *total_operation*. Set all entries of AM and AJ to zero, which means all machines and jobs are available at time 0. Set $t = 0$, *makespan* $= 0$ and $S = 0$.

Step 2. Put all the available machines ($AM_i = 0$) into machine-candidate-pool.

Step 3. Select a machine from the pool based on one of the above criteria introduced in Section 3.2.2. Suppose M_i has been chosen.

Step 4. Put all the available jobs ($AJ_j = 0$) that still need to be processed on M_i ($P_{ij} \neq 0$) into job-candidate-pool and select a job from it based on one of the above criteria introduced in Section 3.2.3. Suppose J_j has been chosen. If the job-candidate-pool is empty, then goto *Step 6.*

Step 5. Assign J_j on M_i, starting at time t and update AM, AJ, RPJ, RPM, ROJ, ROM, P, S and *total_operation* as follows:

$$
\begin{aligned}
AM(i) &= P(i,j) \\
AJ(j) &= P(i,j) \\
RPJ(j) &= RPJ(j) - P(i,j) \\
RPM(i) &= RPM(i) - P(i,j) \\
ROJ(j) &= ROJ(j) - 1 \\
ROM(i) &= ROM(i) - 1 \\
P(i,j) &= 0 \\
s(i,j) &= makespan \\
total\_operation &= total\_operation - 1
\end{aligned}
$$

Step 6. Remove M_i from machine-candidate-pool and repeat Step 3 to Step 5 until machine-candidate-pool is empty.

Step 7. Find the time t when at least one of the occupied machines will be available to accommodate the next job:

$$t = \min\{\text{all the non-zero entries of } AM\} .$$

update *AM*, *AJ* and *makespan* as follows:

$$
\begin{aligned}
AM(i) &= AM(i) - t : i = 1,\ldots m, AM(i) \neq 0 \\
AJ(j) &= AJ(j) - t : j = 1,\ldots n, AJ(j) \neq 0 \\
makespan &= makespan + t.
\end{aligned}
$$

Step 8. Repeat steps 3 to 7 until all the operations have been processed (until $total\_operation = 0$).

Step 9. Calculate $C_{\max}$ as follows:

$$
\begin{aligned}
t &= \max\{\text{all the non-zero entries of } AM\} \\
C_{\max} &= makespan + t.
\end{aligned}
$$

Machine-job algorithms are briefly presented as follows

BEGIN

Calculate *RPM*, *RPJ*, *ROM* and *ROJ* based on the criteria that will be used;

Calculate *total_operation*;

$t = 0$; *makespan* $= 0$;

While *total_operation* $\neq 0$

Use one of the three criteria to arrange the machine that are currently available;

Select one machine each time according to the constructed order,

DO

Use one of the four criteria to select a job from those jobs that are available currently and haven't been processed on the selected machine;

Assign the selected job on the selected machine, and update the corresponding variables as specified in *Step 5*;

END

IF *total_operation* $\neq 0$

$t = \min\{\text{all the non-zero entries of } AM\}$;

$AM(i) = AM(i) - t : i = 1,\ldots m, AM(i) \neq 0$;

$AJ(j) = AJ(j) - t : j = 1,\ldots n, AJ(j) \neq 0$;

makespan $=$ *makespan* $+ t$;

ELSE $t = \max\{\text{all the non-zero entries of } AM\}$;

$C_{\max}$ $=$ *makespan* $+ t$

END

END

3.2 Computational Results

The two sets of test problems used in Chapter 2 are used again to evaluate the performance of machine-job heuristic algorithms.

3.2.1 Results for New Benchmark Problems from Brucker

- Problems: names of new benchmark problems provided by Brucker [1].
- DIFF: the DIFF of the problem.
- WKLD: the WORKLOAD of certain problem.
- Min-PR: the minimum performance ratio of performance ratios of all 12 generated dense schedules.
- Max-PR: the maximum performance ratio of performance ratios of all 12 generated dense schedules.
- Ave-PR: the average performance ratio of performance ratios of all 12 generated dense schedules.
- Con-PR: the conjectured performance ratio or, **the worst case performance ratio** $(2-\frac{1}{m})$.

Performance ratios that are in italics are calculated using LB since optimal C_*s are unknown.

Table 3

Problems	DIFF	WKLD	Min-PR	Max-PR	Ave-PR	Con-PR
j3-per0-1	1.000	1.000	1.0470	1.2325	1.0934	1.6667
j3-per0-2	1.000	1.000	1.0000	1.0120	1.0035	1.6667
j3-per10-0	0.900	0.9617	1.0080	1.0849	1.0348	1.6667
j3-per10-1	0.900	0.9630	1.0000	1.2002	1.0913	1.6667
j3-per10-2	0.900	0.9493	1.0066	1.1956	1.1348	1.6667
j3-per20-0	0.800	0.870	1.0478	1.2661	1.1590	1.6667
j3-per20-1	0.800	0.886	1.0760	1.2140	1.1890	1.6667
j3-per20-2	0.800	0.8997	1.0320	1.1880	1.1049	1.6667
j4-per0-0	1.000	1.000	1.0000	1.0123	1.0062	1.75
j4-per0-1	1.000	1.000	1.0169	1.2729	1.1199	1.75

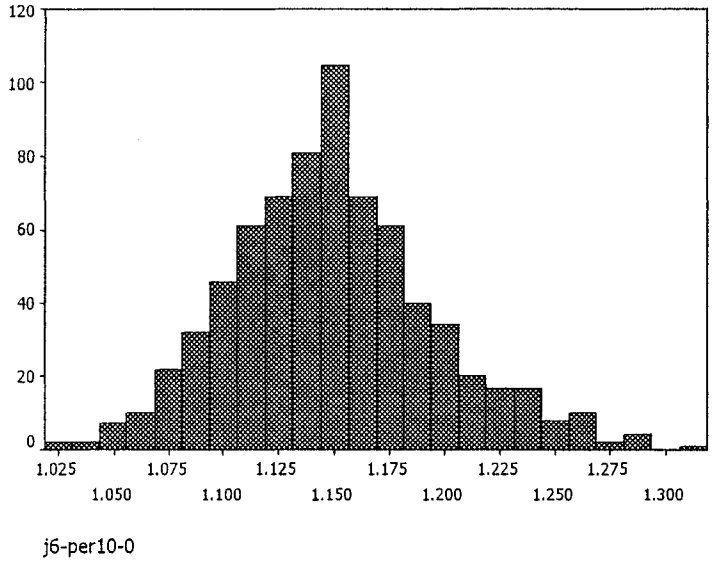
Table 3 (continued)

Problems	DIFF	WKLD	Min-PR	Max-PR	Ave-PR	Con-PR
j4-per0-2	1.000	1.000	1.0140	1.1858	1.1307	1.75
j4-per10-0	0.900	0.9413	1.0365	1.2968	1.1150	1.75
j4-per10-1	0.900	0.9683	1.0854	1.1590	1.1213	1.75
j4-per10-2	0.900	0.9433	1.0000	1.1290	1.0803	1.75
j4-per20-0	0.800	0.8898	1.0560	1.2730	1.1676	1.75
j4-per20-1	0.800	0.9087	1.0777	1.3078	1.1480	1.75
j4-per20-2	0.800	0.9087	1.0704	1.1298	1.1071	1.75
j5-per0-0	1.000	1.000	1.1190	1.1737	1.1622	1.8
j5-per0-1	1.000	1.000	1.0531	1.1082	1.0594	1.8
j5-per0-2	1.000	1.000	1.1016	1.1571	1.1328	1.8
j5-per10-0	0.900	0.948	1.0926	1.2849	1.1889	1.8
j5-per10-1	0.900	0.9294	1.0549	1.1996	1.1415	1.8
j5-per10-2	0.900	0.934	1.0706	1.1948	1.1483	1.8
j5-per20-0	0.800	0.895	1.0420	1.2620	1.1432	1.8
j5-per20-1	0.800	0.8844	1.0670	1.1870	1.1602	1.8
j5-per20-2	0.800	0.911	1.0000	1.2381	1.1217	1.8
j6-per0-0	1.000	1.000	1.0777	1.1828	1.1102	1.833
j6-per0-1	1.000	1.000	1.0651	1.2201	1.1384	1.833
j6-per0-2	1.000	1.000	1.1044	1.1750	1.1541	1.833
j6-per10-0	0.900	0.9440	1.1025	1.2488	1.1488	1.833
j6-per10-1	0.900	0.9492	1.1107	1.1900	1.1440	1.833
j6-per10-2	0.900	0.9463	1.0543	1.1877	1.1264	1.833
j6-per20-1	0.800	0.9045	1.0640	1.1540	1.1083	1.833
j6-per20-2	0.800	0.8797	1.0320	1.1500	1.0900	1.833

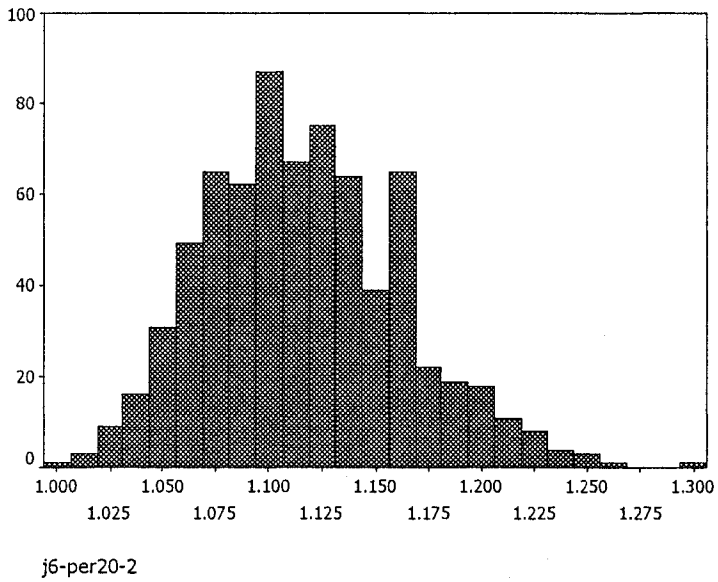
Table 3 (continued)

Problems	DIFF	WKLD	Min-PR	Max-PR	Ave-PR	Con-PR
j7-per0-0	1.000	1.000	*1.1100*	*1.2600*	*1.1918*	*1.857*
j7-per0-1	1.000	1.000	1.0749	1.1886	1.1285	1.857
j7-per0-2	1.000	1.000	1.0407	1.1809	1.1516	1.857
j7-per10-0	0.900	0.9584	*1.1200*	*1.2710*	*1.1739*	*1.857*
j7-per10-1	0.900	0.9440	1.0510	1.2200	1.1303	1.857
j7-per10-2	0.900	0.9509	1.0708	1.2473	1.1602	1.857
j7-per20-0	0.800	0.8786	1.0290	1.1080	1.0713	1.857
j7-per20-1	0.800	0.9307	1.0547	1.1592	1.1208	1.857
j7-per20-2	0.800	0.9253	1.0578	1.2273	1.1542	1.857
j8-per0–1	1.000	1.000	*1.1220*	*1.2610*	*1.2193*	*1.875*
j8-per0-2	1.000	1.000	*1.1470*	*1.3210*	*1.2226*	*1.875*
j8-per10-0	0.900	0.9695	*1.0960*	*1.2550*	*1.1680*	*1.875*
j8-per10-1	0.900	0.9449	*1.0950*	*1.1690*	*1.1223*	*1.875*
j8-per10-2	0.900	0.9504	*1.0880*	*1.1890*	*1.1388*	*1.875*
j8-per20-0	0.800	0.9050	1.0460	1.2250	1.1103	1.875
j8-per20-1	0.800	0.8801	1.0350	1.1330	1.0744	1.875
j8-per20-2	0.800	0.9130	*1.0910*	*1.2330*	*1.1346*	*1.875*

The results from Table 3 show that the average performance ratios of dense schedules generated by using the heuristic algorithms are not necessarily better than those in Table 1, since the performance ratios in Table 1 are already very close to 1. Following graphs are Histograms of performance ratios of 720 random generated dense schedules for Problem j6-per10-0 and Problem j6-per20-2. They could help us learn the general type of the distribution of dense schedules' performance ratio.



Graph 6



Graph 7

3.2.2 Results for Randomly Generated Problems

- m: the number of machines and jobs.
- N.O.P: Number of problems with same number of machines and jobs..
- I.O.DIFF: the interval of DIFF of the corresponding problems.
- I.O.WKLD: the interval of WORKLOAD of the corresponding problems.
- Min-PR: the minimum performance ratio of performance ratios of all (12 × N.O.P.) generated dense schedules.
- Max-PR: the maximum performance ratio of performance ratios of all (12 × N.O.P.) generated dense schedules.
- Ave-PR: the average performance ratio of performance ratios of all (12 × N.O.P.) generated dense schedules.
- Con-PR: the conjectured performance ratio or, **the worst case performance ratio** $(2-\frac{1}{m})$.

Table 4

m	N.O.P.	I.O.DIFF	I.O.WKLD	Min-PR	Max-PR	Ave-PR	Con-PR
5	1200	0.152~0.940	0.354~0.975	1.0000	1.1567	1.0698	1.8
50	1200	0.140~0.928	0.329~0.971	1.0016	1.0458	1.0132	1.98
100	1200	0.138~0.925	0.318~0.952	1.0024	1.0322	1.0092	1.99
150	700	0.139~0.902	0.208~0.925	1.0067	1.0144	1.0067	1.99

In this table the average performance ratios have been improved a little compared to those in Table 2.

Conclusion

We have studied the average performance ratio of dense schedules for open-shop problems. The computational experiments show that the average performance of dense schedules as the solutions of open-shop problems is much better than we expected. We have presented 12 heuristic algorithms to generate dense schedules. We will work on open-shop problems with job release time and study the average performance of dense schedules for open-shop problems with random processing time as further research.

Bibliography

[1] Brucker, P., Hurink, J., Jurisch, B., & Wostmann, B. (1997). A Branch & Bound Algorithm for the Open-shop Problem. *Discrete Applied Mathematics, 76,* 43-59.

[2] Wein, J. M., (1991). *Algorithms for Scheduling and Network Problems.* Combridge, USA: MIT.

[3] Chen, B., & Strusevich, V. A. (1993). Approximation Algorithms for Three Machine Open Shop Scheduling. *ORSA Journal on Computing, 5,* 321-326.

[4] Chen, B. (1994). Worst-case Performance of Scheduling Heuristics. Amsterdam: [s n].

[5] Chen, X. H., & Yu, W. C. (2000-12). Upper-bound of Performance Ratio of Dense Schedule for Open-shop. *Journal of East China University of Science and Technology, vol 26, No.6,* 670-673.

[6] Brucker, P. (1995). *Scheduling Algorithms.* Springer.

[7] Gonzalez, T., & Sahni, S. (1976). Open Shop Scheduling to Minimize Finish Time. *Journal of the Association for Computing Machinery, 23,* 665-679

[8] Pinedo, M. (1995). *Scheduling: Theory, Algorithms, and Systems.* New Jersey: Prentice-Hall, Englewood Cliffs.

[9] Barany, I. & Fiala, T. (1982). Nearly Optimum Solution of Multimachine Scheduling Problems. *Szigma Mathematika Kozgazdasagi Folyoirat, 15,* 177-191.

[10] Taillard, E (1989). Benchmarks for Basic Scheduling Problems. ORWP89/21, Lausanne.

Appendix

All the computations in this thesis are done by MATLAB for windows, Version 6.5.0 Release 13 on Intel Pentium Processor 1600MHz. Followings are all the Matlab codes used.

1. Generate m! Random Dense Schedules for a Benchmark Problem (Used in Section 2.2.1.2)

1.1 Main Function

```
function [makespan_n,s_n,PR]=RandomGenerate_SAMPLE1(p,optimal)
% given the operation matrix p, output the starting time matrix s_n
% and makespan.
LB=max(max(sum(p),max(sum(p,2))));
MIN=min(min(sum(p),min(sum(p,2))));
DIFF=MIN/LB;
[m,n]=size(p); % m is the # of machines, n is the # of jobs.
WORKLOAD=(sum(sum(p)))/(m*LB);
s_norepeat=[];
makespan_n=0;
number=1;
```

```
PR=0;
for i1=1:6
  for i2=1:6
    for i3=1:6
      for i4=1:6
        for i5=1:6
          for i6=1:6
            job_pool=[i1,i2,i3,i4,i5,i6];
            if nosame(job_pool)==1
              temp=p;
              s=[];
              convert;
              total_operation=sum(sum(operation,1));
              AM=zeros(1,m);
              AJ=zeros(1,n);
              t=0;
              makespan=0;
              for x1=1:6
                m_to_be_assigned=x1;
                job_to_be_assigned=job_pool(1,x1);
                random_assign;
              end
              t=min(nonzeros(AM));
              makespan=makespan+t;
              while total_operation~=0
                  AM=time_lapse(AM,t);
                  AJ=time_lapse(AJ,t);
                  zeros_AM=find_zero(AM);
                  m_candidates=zeros_AM;
```

```
                for i=1:size(zeros_AM,2)
                  new_m_candidates=eliminate_zeros(m_candidates);
                  m_candidates=new_m_candidates;
                  random_no=floor(rand(1)*size(m_candidates,2)+1);
                  m_to_be_assigned=m_candidates(random_no);
                  m_candidates(random_no)=0;
                  zeros_AJ=find_zero(AJ);
                  nonzeros_pij=find(p(m_to_be_assigned,:));
                  if isempty(nonzeros_pij)==1
                   nonzeros_pij=0;
                  end
                  same_job=find_same(zeros_AJ,nonzeros_pij);
                  if same_job~=0
                   job_candidates=same_job;
                   job_to_be_assigned=job_candidates
(floor(rand(1)*size(job_candidates,2)+1));
                    random_assign;
                 end
                end
                if all_zero(AM)==0
                 t=min(nonzeros(AM));
                end
                if total_operation==0
                 t=max(nonzeros(AM));
                end
                zeros_AJ=0;
                zeros_AM=0;
                nonzeros_pij=0;
                same_job=0;
```

```
                makespan=makespan+t;
              end
              s_norepeat(:,:,number)=s;
              if optimal~=0
               PR(number,1)=makespan/optimal;
              else PR(number,1)=makespan/LB;
              end
              number=number+1;
              makespan_norepeat(number,1)=makespan;
            end
            s=[];
            p=temp;
           Cmax(times)=makespan;
         end
       end
     end
   end
 end
end
Cmax;
LB;
MIN;
DIFF;
WORKLOAD;
MaxCmax=max(Cmax);
MinCmax=min(Cmax);
```

1.2. Subfunctions Used in the above Main Function

1.2.1 Function to Caculate AM and AJ When Time t Has Lapsed

time_lapse:

```
function b=time_lapse(a,t)
% this function is used to caculate AM and AJ when time t has lapsed
m_tl=size(a,2);
for i_tl=1:m_tl
  if a(i_tl)~=0
    b(i_tl)=a(i_tl)-t;
  else b(i_tl)=0;
  end
end
```

1.2.2 Function to Check Whether There Are Same Entries in Vector p

nosame:

```
function ans=nosame(p)
% if there are no similar entry in vector p, returns 1
ans=1;
for i_ns=1:size(p,2)
  for j_ns=(i_ns+1):size(p,2)
    if p(i_ns)==p(j_ns)
      ans=0;
      break
    end
  end
end
```

1.2.3 Function to Convert Matrix p into An 0-1 Matrix

```
convert:
%covert matrix p to an 0-1 matrix
for i=1:m
  for j=1:n
    if p(i,j)==0
      operation(i,j)=0;
      s(i,j)=-1;
    else operation(i,j)=1;
    end
  end
end
```

1.2.4 Codes to Assign Job j on Machine i

```
random_assign:
% this m-file is to assign job j on machine i.It will do the following
% calculations:
i=m_to_be_assigned;
j=job_to_be_assigned;
AM(i)=p(i,j);
AJ(j)=p(i,j);
opration(i,j)=0;
s(i,j)=makespan;
total_operation=total_operation-1;
p(i,j)=0;
```

1.2.5 Function to Find Same Entries in Matrix a and b

find_same:

```
function same=find_same(a,b)
%this function is to find same entries in 2 matrix a and b.(each entry
%in a or b appears only once) a and b are both row matrixes.
m_fs=size(a,2);
n_fs=size(b,2);
i_fs=1;
same(1)=0;
for x_fs=1:m_fs
  for y_fs=1:n_fs
    if a(x_fs)==b(y_fs)
      same(i_fs)=a(x_fs);
      i_fs=i_fs+1;
    end
  end
end
```

1.2.6 Function to Return the Index of Non-zero Entries in Matrix a

find_zero:

```
function x=find_zero(a)
%this function will return the index of non-zero entries in matrix a.here
%a is an n*m matrix
[m_fz,n_fz]=size(a);
i_fz=1;
x=1;
for x_fz=1:m_fz
  for y_fz=1:n_fz
    if a(x_fz,y_fz)==0
```

```
      x(i_fz)=(x_fz-1)*n_fz+y_fz;
      i_fz=i_fz+1;
    end
  end
end
```

1.2.7 Function to Check If All Entries in a is 0

all_zero:

```
function result=all_zero(a)
%this function is to check if all entries in a is 0, if yes, return
%1,otherwise return 0
x_az=size(a,2);
result=1;
for i_az=1:x_az
  if a(i_az)~=0
    result=0;
    break
  end
end
```

2. Randomly Generate an Open-shop Problem (Used in Section 2.2.2.1)

```
function p=new_generate(m,mean,var)
% p=new_generate(m,mean)
for i=1:m
  for j=1:m
    p(i,j)=normrnd(mean,var);
```

```
    if p(i,j)<0
      p(i,j)=0;
    end
  end
end
LB=max(max(sum(p),max(sum(p,2))))
MIN=min(min(sum(p),min(sum(p,2))))
DIFF=MIN/LB
WORKLOAD=(sum(sum(p)))/(m*LB)
```

3. Randomly Generate One Dense Schedule for an Open-shop Problem (Used in Section 2.2.2.2)

3.1 Main Function

```
function [makespan_one,PR]=RandomGenerateone(p)
% given the operation matrix p, output the starting time matrix s_one and
% makespan.
s_one=[];
makespan_one=0;
PR=0;
LB=max(max(sum(p),max(sum(p,2))));
MIN=min(min(sum(p),min(sum(p,2))));
DIFF=MIN/LB;
[m,n]=size(p);
WORKLOAD=(sum(sum(p)))/(m*LB);
```

```
s=[];
convert;
total_operation=sum(sum(operation,1));
AM=zeros(1,m);
AJ=zeros(1,n);
t=0;
makespan=0;
while total_operation~=0
   AM=time_lapse(AM,t);
   AJ=time_lapse(AJ,t);
   zeros_AM=find_zero(AM);
   m_candidates=zeros_AM;
   for i=1:size(zeros_AM,2)
     new_m_candidates=eliminate_zeros(m_candidates);
     m_candidates=new_m_candidates;
     random_no=floor(rand(1)*size(m_candidates,2)+1);
     m_to_be_assigned=m_candidates(random_no);
     m_candidates(random_no)=0;
     zeros_AJ=find_zero(AJ);
     nonzeros_pij=find(p(m_to_be_assigned,:));
     if isempty(nonzeros_pij)==1
      nonzeros_pij=0;
     end
     same_job=find_same(zeros_AJ,nonzeros_pij);
     if same_job~=0
      job_candidates=same_job;
      job_to_be_assigned=job_candidates(floor(rand(1)*size(job_candidates,2)+1));
      random_assign;
     end
```

```
    end
    if all_zero(AM)==0
      t=min(nonzeros(AM));
    end
    if total_operation==0
      t=max(nonzeros(AM));
    end
    zeros_AJ=0;
    zeros_AM=0;
    nonzeros_pij=0;
    same_job=0;
    makespan=makespan+t;
end
PR=makespan/LB;
s_one=s
makespan_one=makespan;
```

3.2 Subfunction Used in the above Main Function

3.2.1 Function to Eliminate Zero Entries in Coloum Vector a

eliminate_zeros:

```
function f=eliminate_zeros(a)
% this function is to eliminate zero entries in coloum vector a
nonzeros_a=find(a);
n_ez=size(nonzeros_a,2);
for i_ez=1:n_ez
  f(i_ez)=a(nonzeros_a(i_ez));
```

```
end
```

4. Codes for 12 Heuristic Algorithms (Used in Chapter 3)

4.1 Criteria a) to Select a Machine + Criteria 1) to Select a job

4.1.1 Main Function

```
function makespan=A1(p)
% given the operation matrix p, output the makespan.
temp1=p;
[m,n]=size(p);
s=[];
convert;
total_operation=sum(sum(operation,1));
AM=zeros(1,m);
AJ=zeros(1,n);
t=0;
RPJ=zeros(1,n);
RPM=zeros(1,m);
makespan=0;
for i=1:n
  for j=1:m
    RPJ(i)=RPJ(i)+p(j,i);
  end
end
for i=1:m
  for j=1:n
    RPM(i)=RPM(i)+p(i,j);
```

```
  end
end
maxRPJ=max(RPJ);
maxRPM=max(RPM);
while total_operation~=0
    AM=time_lapse(AM,t);
    AJ=time_lapse(AJ,t);
    zeros_AM=find_zero(AM);
    RPM_order=sortdescending(RPM,zeros_AM);
    for x_mj=1:size(RPM_order,2)
      zeros_AJ=find_zero(AJ);
      nonzeros_pij=find(p(RPM_order(x_mj),:));
      if isempty(nonzeros_pij)==1
       nonzeros_pij=0;
      end
      same_job=find_same(zeros_AJ,nonzeros_pij);
      if same_job~=0
       job_to_be_assigned=find_max(RPJ,same_job);
       assign_A1;
      end
    end
    AM;
    s;
    AJ;
    if all_zero(AM)==0
     t=min(nonzeros(AM));
    end
    if total_operation==0
     t=max(nonzeros(AM));
```

```
    end
    zeros_AJ=0;
    zeros_AM=0;
    nonzeros_pij=0;
    same_job=0;
    RPM_order=0;
    makespan=makespan+t;
end
p=temp1;
makespan;
maxRPJ;
maxRPM;
```

4.1.2 Subfunctions Used in the above Function

a) Function to Sort the Entries According the Ascending Order

```
sortdescending:
function order=sortascending(a,b)
% given any 1*q matrixa,and an index set b, return the
% order of the values from the smallest to the largest in b.
q=size(b,2);
order=b;
for y1=1:(q-1)
  for x1=1:q-y1
    if a(b(x1))>a(b(x1+1));
      temp=a(b(x1));
      a(b(x1))=a(b(x1+1));
      a(b(x1+1))=temp;
```

```
      temp=order(x1);
      order(x1)=order(x1+1);
      order(x1+1)=temp;
    end
  end
end
```

b) Function to Find the Maximum Entry of a Given Matrix

find_max:

```
function max_index=find_max(a_fm,b_fm)
%this function is to return the index of the maximum entry of matrix a(b)
n_fm=size(b_fm,2);
max_index=b_fm(1);
for x_fm=1:n_fm-1
  if a_fm(b_fm(x_fm))<a_fm(b_fm(x_fm+1))
    max_index=b_fm(x_fm+1);
  else
    temp=a_fm(b_fm(x_fm+1));
    a_fm(b_fm(x_fm+1))=a_fm(b_fm(x_fm));
    a_fm(b_fm(x_fm))=temp;
  end
end
```

C) Codes to Assign Job j on Machine i

assign_A1:

```
% this m-file is to assign job j on machine i.It will do the following
% calculations:
i=RPM_order(x_mj);
```

```
j=job_to_be_assigned;
AM(i)=p(i,j);
AJ(j)=p(i,j);
RPJ(j)=RPJ(j)-p(i,j);
RPM(i)=RPM(i)-p(i,j);
opration(i,j)=0;
s(i,j)=makespan;
total_operation=total_operation-1;
p(i,j)=0;
```

4.2 Criteria a) to Select a Machine + Criteria 2) to Select a Job

4.2.1 Main Function

```
function makespan=A2(p)
% given the operation matrix p, output the makespan.
temp1=p;
[m,n]=size(p);
s=[];
convert;
total_operation=sum(sum(operation,1));
AM=zeros(1,m);
AJ=zeros(1,n);
t=0; % t is the current time
RPJ=zeros(1,n);
RPM=zeros(1,m);
```

```
ROJ=zeros(1,n);
for i=1:n
  for j=1:m
    ROJ(i)=ROJ(i)+operation(i,j);
  end
end
makespan=0;
for i=1:n
  for j=1:m
    RPJ(i)=RPJ(i)+p(j,i);
  end
end
for i=1:m
  for j=1:n
    RPM(i)=RPM(i)+p(i,j);
  end
end
maxRPJ=max(RPJ);
maxRPM=max(RPM);
while total_operation~=0
    AM=time_lapse(AM,t);
    AJ=time_lapse(AJ,t);
    zeros_AM=find_zero(AM);
    RPM_order=sortdescending(RPM,zeros_AM);
    for x_mj=1:size(RPM_order,2)
      zeros_AJ=find_zero(AJ);
      nonzeros_pij=find(p(RPM_order(x_mj),:));
      if isempty(nonzeros_pij)==1
        nonzeros_pij=0;
```

```
      end
      same_job=find_same(zeros_AJ,nonzeros_pij);
      if same_job~=0
       job_to_be_assigned=find_max(ROJ,same_job);
       assign_A2;
      end
    end
    AM;
    s;
    AJ;
    if all_zero(AM)==0
     t=min(nonzeros(AM));
    end
    if total_operation==0
     t=max(nonzeros(AM));
    end
    zeros_AJ=0;
    zeros_AM=0;
    nonzeros_pij=0;
    same_job=0;
    RPM_order=0;
    makespan=makespan+t;
end
p=temp1;
makespan;
maxRPJ;
maxRPM;
assign_A2:
```

4.2.2 Subfunction Used in the above Main Function

a) Codes to Assign Job j on Machine i

```
% this m-file is to assign job j on machine i.It will do the following
% calculations:
i=RPM_order(x_mj);
j=job_to_be_assigned;
AM(i)=p(i,j);
AJ(j)=p(i,j);
ROJ(j)=ROJ(j)-1;
RPM(i)=RPM(i)-p(i,j);
opration(i,j)=0;
s(i,j)=makespan;
total_operation=total_operation-1;
p(i,j)=0;
```

4.3 Criteria a) to Select a Machine + Criteria 3) to Select a Job

4.3.1 Main Function

```
function makespan=A3(p)
% given the operation matrix p, output the makespan.
temp1=p;
[m,n]=size(p);
s=[];
convert;
total_operation=sum(sum(operation,1));
AM=zeros(1,m);
AJ=zeros(1,n);
```

```
t=0;
RPJ=zeros(1,n);
RPM=zeros(1,m);
makespan=0;
for i=1:n
  for j=1:m
    RPJ(i)=RPJ(i)+p(j,i);
  end
end
for i=1:m
  for j=1:n
    RPM(i)=RPM(i)+p(i,j);
  end
end
maxRPJ=max(RPJ);
maxRPM=max(RPM);
while total_operation~=0
   AM=time_lapse(AM,t);
   AJ=time_lapse(AJ,t);
   zeros_AM=find_zero(AM);
   RPM_order=sortdescending(RPM,zeros_AM);
   for x_mj=1:size(RPM_order,2)
     zeros_AJ=find_zero(AJ);
     nonzeros_pij=find(p(RPM_order(x_mj),:));
     if isempty(nonzeros_pij)==1
       nonzeros_pij=0;
     end
     same_job=find_same(zeros_AJ,nonzeros_pij);
     if same_job~=0
```

```
      job_to_be_assigned=find_min(p(RPM_order(x_mj),:),same_job);
      assign_A3;
     end
   end
   AM;
   s;
   AJ;
   if all_zero(AM)==0
    t=min(nonzeros(AM));
   end
   if total_operation==0
    t=max(nonzeros(AM));
   end
   zeros_AJ=0;
   zeros_AM=0;
   nonzeros_pij=0;
   same_job=0;
   RPM_order=0;
   makespan=makespan+t;
end
p=temp1;
makespan
maxRPJ
maxRPM
```

4.3.2 Subfunctions Used in the Above Main Function

a) Function to Find the Index of the Minimum Entry in Matrix a

find_min:

```
function min_index=find_min(a_fm,b_fm)
%this function is to return the index of the minimum entry of matrix a(b)
n_fm=size(b_fm,2);
min_index=b_fm(1);
for x_fm=1:n_fm-1
  if a_fm(b_fm(x_fm))>a_fm(b_fm(x_fm+1))
    min_index=b_fm(x_fm+1);
  else
    temp=a_fm(b_fm(x_fm+1));
    a_fm(b_fm(x_fm+1))=a_fm(b_fm(x_fm));
    a_fm(b_fm(x_fm))=temp;
  end
end
```

b) Codes to Assign Job j on Machine i

assign_A3:

```
% this m-file is to assign job j on machine i.It will do the following
% calculations:
i=RPM_order(x_mj);
j=job_to_be_assigned;
AM(i)=p(i,j);
AJ(j)=p(i,j);
RPM(i)=RPM(i)-p(i,j);
opration(i,j)=0;
s(i,j)=makespan;
total_operation=total_operation-1;
p(i,j)=0;
```

4.4 Criteria a) to Select a Machine + Criteria 4) to Select a Job

4.4.1 Main Function

```
function makespan=A4(p)
% given the operation matrix p, output the makespan.
temp1=p;
[m,n]=size(p);
s=[];
convert;
total_operation=sum(sum(operation,1));
AM=zeros(1,m);
AJ=zeros(1,n);
t=0;
RPJ=zeros(1,n);
RPM=zeros(1,m);
makespan=0;
for i=1:m
  for j=1:n
    RPM(i)=RPM(i)+p(i,j);
  end
end
for i=1:n
  for j=1:m
    RPJ(i)=RPJ(i)+p(j,i);
  end
end
maxRPJ=max(RPJ);
```

```
maxRPM=max(RPM);
while total_operation~=0
   AM=time_lapse(AM,t);
   AJ=time_lapse(AJ,t);
   zeros_AM=find_zero(AM);
   RPM_order=sortdescending(RPM,zeros_AM);
   for x_mj=1:size(RPM_order,2)
     zeros_AJ=find_zero(AJ);
     nonzeros_pij=find(p(RPM_order(x_mj),:));
     if isempty(nonzeros_pij)==1
      nonzeros_pij=0;
     end
     same_job=find_same(zeros_AJ,nonzeros_pij);
     if same_job~=0
      job_to_be_assigned=find_max(p(RPM_order(x_mj),:),same_job);
      assign_A4;
    end
   end
   if all_zero(AM)==0
    t=min(nonzeros(AM));
   end
   if total_operation==0
    t=max(nonzeros(AM));
   end
   zeros_AJ=0;
   zeros_AM=0;
   nonzeros_pij=0;
   same_job=0;
   RPM_order=0;
```

```
    makespan=makespan+t;
end
p=temp1;
makespan
maxRPJ
maxRPM
```

4.4.2 Subfunction Used in the above Main Function

a) Codes to Assign Job j on Machine i

```
assign_A4:
% this m-file is to assign job j on machine i.It will do the following
% calculations:
i=RPM_order(x_mj);
j=job_to_be_assigned;
AM(i)=p(i,j);
AJ(j)=p(i,j);
RPM(i)=RPM(i)-p(i,j);
opration(i,j)=0;
s(i,j)=makespan;
total_operation=total_operation-1;
p(i,j)=0;
```

4.5 Criteria b) to Select a Machine + Criteria 1) to Select a Job

4.5.1 Main Function

```
function makespan=B1(p)
% given the operation matrix p, output the makespan
temp1=p;
```

```
[m,n]=size(p);
s=[];
convert;
total_operation=sum(sum(operation,1));
AM=zeros(1,m);
AJ=zeros(1,n);
t=0; % t is the current time
RPJ=zeros(1,n);
ROM=zeros(1,m);
RPM=zeros(1,m);
makespan=0;
for i=1:n
  for j=1:m
    RPJ(i)=RPJ(i)+p(j,i);
  end
end
for i=1:m
  for j=1:n
    ROM(i)=ROM(i)+operation(i,j);
  end
end
while total_operation~=0
    AM=time_lapse(AM,t);
    AJ=time_lapse(AJ,t);
    eros_AM=find_zero(AM);
    ROM_order=sortdescending(ROM,zeros_AM);
    for x_mj=1:size(ROM_order,2)
      zeros_AJ=find_zero(AJ);
      nonzeros_pij=find(p(ROM_order(x_mj),:));
```

```
      if isempty(nonzeros_pij)==1
       nonzeros_pij=0;
      end
      same_job=find_same(zeros_AJ,nonzeros_pij);
      if same_job~=0
       job_to_be_assigned=find_max(RPJ,same_job);
       assign_B1;
      end
    end
    AM;
    s;
    AJ;
    if all_zero(AM)==0
     t=min(nonzeros(AM));
    end
    if total_operation==0
     t=max(nonzeros(AM));
    end
    zeros_AJ=0;
    zeros_AM=0;
    nonzeros_pij=0;
    same_job=0;
    ROM_order=0;
    makespan=makespan+t;
end
p=temp1;
makespan;
RPJ=zeros(n);
RPM=zeros(m);
```

```
for i=1:n
  for j=1:m
    RPJ(i)=RPJ(i)+p(j,i);
  end
end
for i=1:m
  for j=1:n
    RPM(i)=RPM(i)+p(i,j);
  end
end
maxRPJ=max(max(RPJ));
maxRPM=max(max(RPM));
```

4.5.2 Subfunction Used in the above Main Function

a) Codes to Assign Job j on Machine i

```
assign_B1:
% this m-file is to assign job j on machine i.It will do the following
% calculations:
i=ROM_order(x_mj);
j=job_to_be_assigned;
AM(i)=p(i,j);
AJ(j)=p(i,j);
RPJ(j)=RPJ(j)-1;
ROM(i)=ROM(i)-1;
opration(i,j)=0;
s(i,j)=makespan;
total_operation=total_operation-1;
```

```
p(i,j)=0;
```

4.6 Criteria b) to Select a Machine + Criteria 2) to Select a Job

4.6.1 Main Function

```
function makespan=B2(p)
% given the operation matrix p, output the makespan.
temp1=p;
[m,n]=size(p);
s=[];
convert;
total_operation=sum(sum(operation,1));
AM=zeros(1,m);
AJ=zeros(1,n);
t=0;
ROJ=zeros(1,n);
ROM=zeros(1,m);
makespan=0;
for i=1:n
  for j=1:m
    ROJ(i)=ROJ(i)+operation(i,j);
  end
end
for i=1:m
  for j=1:n
    ROM(i)=ROM(i)+operation(i,j);
  end
```

```
end
while total_operation~=0
   AM=time_lapse(AM,t);
   AJ=time_lapse(AJ,t);
   zeros_AM=find_zero(AM);
   ROM_order=sortdescending(ROM,zeros_AM);
   for x_mj=1:size(ROM_order,2)
     zeros_AJ=find_zero(AJ);
     nonzeros_pij=find(p(ROM_order(x_mj),:));
     if isempty(nonzeros_pij)==1
      nonzeros_pij=0;
     end
     same_job=find_same(zeros_AJ,nonzeros_pij);
     if same_job~=0
      job_to_be_assigned=find_max(ROJ,same_job);
      assign_B2;
     end
   end
   AM;
   s;
   AJ;
   if all_zero(AM)==0
    t=min(nonzeros(AM));
   end
   if total_operation==0
    t=max(nonzeros(AM));
   end
   zeros_AJ=0;
   zeros_AM=0;
```

```
    nonzeros_pij=0;
    same_job=0;
    ROM_order=0;
    makespan=makespan+t;
end
p=temp1;
makespan;
RPJ=zeros(n);
RPM=zeros(m);
for i=1:n
  for j=1:m
    RPJ(i)=RPJ(i)+p(j,i);
  end
end
for i=1:m
  for j=1:n
    RPM(i)=RPM(i)+p(i,j);
  end
end
maxRPJ=max(max(RPJ));
maxRPM=max(max(RPM));
```

4.6.2 Subfunction Used in the above Main Function

a) Codes to Assign Job j on Machine i

```
assign_B2:
% this m-file is to assign job j on machine i.It will do the following
% calculations:
```

```
i=ROM_order(x_mj);
j=job_to_be_assigned;
AM(i)=p(i,j);
AJ(j)=p(i,j);
ROJ(j)=ROJ(j)-1;
ROM(i)=ROM(i)-1;
opration(i,j)=0;
s(i,j)=makespan;
total_operation=total_operation-1;
p(i,j)=0;
```

4.7 Criteria b) to Select a Machine + Criteria 3) to Select a Job

4.7.1 Main Function

```
function makespan=B3(p)
% given the operation matrix p, output the makespan.
temp1=p;
[m,n]=size(p);
s=[];
convert;
total_operation=sum(sum(operation,1));
AM=zeros(1,m);
AJ=zeros(1,n);
t=0; % t is the current time
RPJ=zeros(1,n);
RPM=zeros(1,m);
ROM=zeros(1,m);
makespan=0;
for i=1:n
```

```
  for j=1:m
    RPJ(i)=RPJ(i)+p(j,i);
  end
end
for i=1:m
  for j=1:n
    RPM(i)=RPM(i)+p(i,j);
  end
end
for i=1:m
  for j=1:n
    ROM(i)=ROM(i)+operation(i,j);
  end
end
maxRPJ=max(RPJ);
maxRPM=max(RPM);
while total_operation~=0
  AM=time_lapse(AM,t);
  AJ=time_lapse(AJ,t);
  zeros_AM=find_zero(AM);
  ROM_order=sortdescending(ROM,zeros_AM);
  for x_mj=1:size(ROM_order,2)
    zeros_AJ=find_zero(AJ);
    nonzeros_pij=find(p(ROM_order(x_mj),:));
    if isempty(nonzeros_pij)==1
      nonzeros_pij=0;
    end
    same_job=find_same(zeros_AJ,nonzeros_pij);
    if same_job~=0
```

```
      job_to_be_assigned=find_min(p(ROM_order(x_mj),:),same_job);
      assign_B3;
    end
  end
  if all_zero(AM)==0
   t=min(nonzeros(AM));
  end
  if total_operation==0
   t=max(nonzeros(AM));
  end
  zeros_AJ=0;
  zeros_AM=0;
  nonzeros_pij=0;
  same_job=0;
  RPM_order=0;
  makespan=makespan+t;
end
p=temp1;
makespan;
maxRPJ;
maxRPM;
```

4.7.2 Subfunctions Used in the above Main Function

a) Codes to Assign Job j on Machine i

```
assign_B3:
% this m-file is to assign job j on machine i.It will do the following
% calculations:
i=ROM_order(x_mj);
```

```
j=job_to_be_assigned;
AM(i)=p(i,j);
AJ(j)=p(i,j);
ROM(i)=ROM(i)-1;
opration(i,j)=0;
s(i,j)=makespan;
total_operation=total_operation-1;
p(i,j)=0;
```

4.8 Criteria b) to Select a Machine + Criteria 4) to Select a Job

4.8.1 Main Function

```
function makespan=B4(p)
% given the operation matrix p, output the makespan.
temp1=p;
[m,n]=size(p);
s=[];
convert;
total_operation=sum(sum(operation,1));
AM=zeros(1,m);
AJ=zeros(1,n);
t=0; % t is the current time
RPJ=zeros(1,n);
RPM=zeros(1,m);
ROM=zeros(1,m);
makespan=0;
for i=1:n
  for j=1:m
    RPJ(i)=RPJ(i)+p(j,i);
```

```
  end
end
for i=1:m
  for j=1:n
    RPM(i)=RPM(i)+p(i,j);
  end
end
for i=1:m
  for j=1:n
    ROM(i)=ROM(i)+operation(i,j);
  end
end
maxRPJ=max(RPJ);
maxRPM=max(RPM);
while total_operation~=0
   AM=time_lapse(AM,t);
   AJ=time_lapse(AJ,t);
   zeros_AM=find_zero(AM);
   ROM_order=sortdescending(ROM,zeros_AM);
   for x_mj=1:size(ROM_order,2)
     zeros_AJ=find_zero(AJ);
     nonzeros_pij=find(p(ROM_order(x_mj),:));
     if isempty(nonzeros_pij)==1
      nonzeros_pij=0;
     end
     same_job=find_same(zeros_AJ,nonzeros_pij);
     if same_job~=0
      job_to_be_assigned=find_max(p(ROM_order(x_mj),:),same_job);
      assign_B4;
```

```
      end
    end
    AM;
    s;
    AJ;
    if all_zero(AM)==0
     t=min(nonzeros(AM));
    end
    if total_operation==0
     t=max(nonzeros(AM));
    end
    zeros_AJ=0;
    zeros_AM=0;
    nonzeros_pij=0;
    same_job=0;
    RPM_order=0;
    makespan=makespan+t;
end
p=temp1;
makespan;
maxRPJ;
maxRPM;
```

4.8.2 Subfunctions Used in the above Main Function

a) Codes to Assign Job j on Machine i

assign_B4:

% this m-file is to assign job j on machine i.It will do the following

```
% calculations:
i=ROM_order(x_mj);
j=job_to_be_assigned;
AM(i)=p(i,j);
AJ(j)=p(i,j);
ROM(i)=ROM(i)-1;
opration(i,j)=0;
s(i,j)=makespan;
total_operation=total_operation-1;
p(i,j)=0;
```

4.9 Criteria c) to Select a Machine + Criteria 1) to Select a Job

4.9.1 Mian Function

```
function makespan=C1(p)
% given the operation matrix p, output the makespan
temp1=p;
[m,n]=size(p);
s=[];
convert;
total_operation=sum(sum(operation,1));
AM=zeros(1,m);
AJ=zeros(1,n);
t=0; % t is the current time
RPJ=zeros(1,n);
RPM=zeros(1,m);
makespan=0;
for i=1:n
  for j=1:m
```

```
    RPJ(i)=RPJ(i)+p(j,i);
  end
end
for i=1:m
  for j=1:n
    RPM(i)=RPM(i)+p(i,j);
  end
end
maxRPJ=max(RPJ);
maxRPM=max(RPM);
while total_operation~=0
    AM=time_lapse(AM,t);
    AJ=time_lapse(AJ,t);
    zeros_AM=find_zero(AM);
    for x_mj=1:size(zeros_AM,2)
      zeros_AJ=find_zero(AJ);
      nonzeros_pij=find(p(zeros_AM(x_mj),:));
      if isempty(nonzeros_pij)==1
       nonzeros_pij=0;
      end
      same_job=find_same(zeros_AJ,nonzeros_pij);
      if same_job~=0
       job_to_be_assigned=find_max(RPJ,same_job);
       assign_C1;
      end
    end
    if all_zero(AM)==0
     t=min(nonzeros(AM));
    end
```

```
    if total_operation==0
      t=max(nonzeros(AM));
    end
    zeros_AJ=0;
    zeros_AM=0;
    nonzeros_pij=0;
    same_job=0;
    RPM_order=0;
    makespan=makespan+t;
end
p=temp1;
makespan;
maxRPJ;
maxRPM;
```

4.9.2 Subfunctions Used in the above Main Function

a) Codes to Assign Job j on Machine i

```
assign_C1:
% this m-file is to assign job j on machine i.It will do the
% following calculations:
i=zeros_AM(x_mj);
j=job_to_be_assigned;
AM(i)=p(i,j);
AJ(j)=p(i,j);
RPJ(j)=RPJ(j)-p(i,j);
opration(i,j)=0;
s(i,j)=makespan;
```

```
total_operation=total_operation-1;
p(i,j)=0;
```

4.10 Criteria c) to Select a Machine + Criteria 2) to Select a Job

4.10.1 Main Function

```
function makespan=C2(p)
% given the operation matrix p, output the makespan
temp1=p;
[m,n]=size(p);
s=[];
convert;
total_operation=sum(sum(operation,1));
AJ=zeros(1,n);
t=0;
RPJ=zeros(1,n);
RPM=zeros(1,m);
ROJ=zeros(1,n);
for i=1:n
  for j=1:m
    ROJ(i)=ROJ(i)+operation(i,j);
  end
end
makespan=0;
for i=1:n
  for j=1:m
    RPJ(i)=RPJ(i)+p(j,i);
  end
end
```

```
for i=1:m
  for j=1:n
    RPM(i)=RPM(i)+p(i,j);
  end
end
maxRPJ=max(RPJ);
maxRPM=max(RPM);
while total_operation~=0
    AM=time_lapse(AM,t);
    AJ=time_lapse(AJ,t);
    zeros_AM=find_zero(AM);
    for x_mj=1:size(zeros_AM,2)
      zeros_AJ=find_zero(AJ);
      nonzeros_pij=find(p(zeros_AM(x_mj),:));
      if isempty(nonzeros_pij)==1
       nonzeros_pij=0;
      end
      same_job=find_same(zeros_AJ,nonzeros_pij);
      if same_job~=0
       job_to_be_assigned=find_max(ROJ,same_job);
       assign_C2;
      end
    end
    if all_zero(AM)==0
     t=min(nonzeros(AM));
    end
    if total_operation==0
     t=max(nonzeros(AM));
    end
```

```
    zeros_AJ=0;
    zeros_AM=0;
    nonzeros_pij=0;
    same_job=0;
    RPM_order=0;
    makespan=makespan+t;
end
p=temp1;
makespan;
maxRPJ;
maxRPM;
```

4.10.2 Subfunctions Used in the above Main Function

a) Codes to Assign Job j on Machine i

```
assign_C2:
% this m-file is to assign job j on machine i.It will do the following
% calculations:
i=zeros_AM(x_mj);
j=job_to_be_assigned;
AM(i)=p(i,j);
AJ(j)=p(i,j);
ROJ(j)=ROJ(j)-1;
opration(i,j)=0;
s(i,j)=makespan;
total_operation=total_operation-1;
p(i,j)=0;
```

4.11 Criteria c) to Select a Machine + Criteria 3) to Select a Job

4.11.1 Main Function

```
function makespan=C3(p)
%C3 M(nature)-J(SPT)
LB=max(max(sum(p),max(sum(p,2))));
MIN=min(min(sum(p),min(sum(p,2))));
DIFF=MIN/LB;
[m,n]=size(p);
WORKLOAD=(sum(sum(p)))/(m*LB);
s=[];   % preassign space for s.
convert; % generate the 0-1 matrix
total_operation=sum(sum(operation,1));
AM=zeros(1,m);
AJ=zeros(1,n);
t=0; % t is the current time
makespan=0;
RPJ=zeros(n);
RPM=zeros(m);
for i=1:n
  for j=1:m
    RPJ(i)=RPJ(i)+p(j,i); % calculate RPJ
  end
end
for i=1:m
  for j=1:n
    RPM(i)=RPM(i)+p(i,j); % calculate RPM
  end
```

```
end
maxRPJ=max(max(RPJ));
maxRPM=max(max(RPM));
for i=1:m
  for j=1:n
    operation_pool((i-1)*n+j)=p(i,j);
  end
end
a=[1:m*n];
operation_order=sortascending(operation_pool,a);
while total_operation~=0
    AM=time_lapse(AM,t);
    AJ=time_lapse(AJ,t);
    zeros_AM=find_zero(AM);
    zeros_AJ=find_zero(AJ);
    times=size(zeros_AM,2);
    j=1;
    while j<=times
       find=0;
       i=1;
       while find==0&&i<=m*n
          job_no=mod(operation_order(i),n);
          if job_no==0
           job_no=n;
           machine_no=operation_order(i)/n;
          else
           machine_no=floor(operation_order(i)/n)+1;
          end
         if operation(machine_no,job_no)~=0
```

```
&&all_zero(machine_no==zeros_AM)==0&&all_zero(job_no==zeros_AJ)==0
            AM(machine_no)=p(machine_no,job_no);
            AJ(job_no)=p(machine_no,job_no);
            operation(machine_no,job_no)=0;
            s(machine_no,job_no)=makespan;
            total_operation=total_operation-1;
            find=1;
           else i=i+1;
           end
        end
        if find==0;
         break
        end
        j=j+1;
        zeros_AM=find_zero(AM);
        zeros_AJ=find_zero(AJ);
    end
    if all_zero(AM)==0
     t=min(nonzeros(AM));
    end
    if total_operation==0
     t=max(nonzeros(AM));
    end
    zeros_AJ=0;
    zeros_AM=0;
    makespan=makespan+t;
end
makespan;
maxRPJ;
```

```
maxRPM;
```

4.12 Criteria c) to Select a Machine + Criteria 4) to Select a Job

4.12.1 Main Function

```
function makespan=C4(p)
%C4 M(nature)-J(LPT)
LB=max(max(sum(p),max(sum(p,2))));
MIN=min(min(sum(p),min(sum(p,2))));
DIFF=MIN/LB;
[m,n]=size(p);
WORKLOAD=(sum(sum(p)))/(m*LB);
s=[];
convert;
total_operation=sum(sum(operation,1));
AM=zeros(1,m);
AJ=zeros(1,n);
t=0;
makespan=0;
RPJ=zeros(n);
RPM=zeros(m);
for i=1:n
  for j=1:m
    RPJ(i)=RPJ(i)+p(j,i); % calculate RPJ
  end
end
for i=1:m
  for j=1:n
    RPM(i)=RPM(i)+p(i,j); % calculate RPM
```

```
  end
end
maxRPJ=max(max(RPJ));
maxRPM=max(max(RPM));
for i=1:m
  for j=1:n
    operation_pool((i-1)*n+j)=p(i,j);
  end
end
a=[1:m*n];
operation_order=sortdescending(operation_pool,a);
while total_operation~=0
   AM=time_lapse(AM,t);
   AJ=time_lapse(AJ,t);
   zeros_AM=find_zero(AM);
   zeros_AJ=find_zero(AJ);
   times=size(zeros_AM,2);
   j=1;
   while j<=times
      find=0;
      i=1;
      while find==0&&i<=m*n
         job_no=mod(operation_order(i),n);
         if job_no==0
          job_no=n;
          machine_no=operation_order(i)/n;
         else
          machine_no=floor(operation_order(i)/n)+1;
         end
```

```
            if operation(machine_no,job_no)~=0
&&all_zero(machine_no==zeros_AM)==0&&all_zero(job_no==zeros_AJ)==0
              AM(machine_no)=p(machine_no,job_no);
              AJ(job_no)=p(machine_no,job_no);
              operation(machine_no,job_no)=0;
              s(machine_no,job_no)=makespan;
              total_operation=total_operation-1;
              find=1;
            else i=i+1;
            end
        end
        if find==0
          break
        end
        j=j+1;
        zeros_AM=find_zero(AM);
       zeros_AJ=find_zero(AJ);
    end
    if all_zero(AM)==0
      t=min(nonzeros(AM));
    end
    if total_operation==0
      t=max(nonzeros(AM));
    end
    zeros_AJ=0;
    zeros_AM=0;
    makespan=makespan+t;
end
makespan;
```

```
maxRPJ;
maxRPM;
```

4.12.2 Subfunction Used in the above Main Function

a) Function to Sort the Entries in Matrix a According to the Descending Order

```
function order=sortdescending(a,b)
% given any 1*q matrix a,and an index set b, return the order of the values from the largest to
% the smallest in b.
q=size(b,2);
order=b;
same=0;
for y1=1:(q-1)
  for x1=1:q-y1
    if a(b(x1))==a(b(x1+1))&a(b(x1))~=0
      same=same+1;
    end
    if a(b(x1))<a(b(x1+1));
      temp=a(b(x1));
      a(b(x1))=a(b(x1+1));
      a(b(x1+1))=temp;
      temp=order(x1);
      order(x1)=order(x1+1);
      order(x1+1)=temp;
    end
  end
end
if same~=0
```

```
  same;
end
```