National Library of Canada

Acquisitions and Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada

Direction des acquisitions et des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canadä

Performance of Ordinal Algorithms for Parallel Machine Scheduling Problems

By

Tracy Lin Chen

Supervised by

Professor Jeffery B. Sidney

Professor John C. Nash

Systems Science Programme

University of Ottawa

Tracy Lin Chen, Ottawa, Canada, 1994

National Library of Canada

Bibliothèque nationale du Canada

Acquisitions and Bibliographic Services Branch

Direction des acquisitions et des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

THE AUTHOR HAS GRANTED AN IRREVOCABLE NON-EXCLUSIVE LICENCE ALLOWING THE NATIONAL LIBRARY OF CANADA TO REPRODUCE, LOAN, DISTRIBUTE OR SELL COPIES OF HIS/HER THESIS BY ANY MEANS AND IN ANY FORM OR FORMAT, MAKING THIS THESIS AVAILABLE TO INTERESTED PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE IRREVOCABLE ET NON EXCLUSIVE PERMETTANT A LA BIBLIOTHEQUE NATIONALE DU CANADA DE REPRODUIRE, PRETER, DISTRIBUER OU VENDRE DES COPIES DE SA THESE DE QUELQUE MANIERE ET SOUS QUELQUE FORME QUE CE SOIT POUR METTRE DES EXEMPLAIRES DE CETTE THESE A LA DISPOSITION DES PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP OF THE COPYRIGHT IN HIS/HER THESIS. NEITHER THE THESIS NOR SUBSTANTIAL EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT HIS/HER PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE DU DROIT D'AUTEUR QUI PROTEGE SA THESE. NI LA THESE NI DES EXTRAITS SUBSTANTIELS DE CELLE-CI NE DOIVENT ETRE IMPRIMES OU AUTREMENT REPRODUITS SANS SON AUTORISATION.

ISBN 0-612-00523-2

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

Acknowledgement

I wish to express my gratitude to Professor Jeffery B. Sidney and Professor John C. Nash for their supervision and encouragement.

Contents

1. **Introduction** 1

   1.1 General Scheduling Problem 1

   1.2 The Parallel Machine Scheduling Problem 2

   1.3 Ordinal Algorithm 5

   1.4 Problem to be studied 9

   1.5 Thesis Outline 10

2. **Problem Definition and Experimental Design** 11

   2.1 Definitions and Notations 11

   2.2 Using Simulation 20

   2.3 Experiments to be Performed 21

   2.4 Methodology 23

      2.4.1 Choice of Environment and Program Language 25

      2.4.2 Choice of Random Number Generator 26

      2.4.3 Transformation to Other Distributions 28

      2.4.4 Choice of Parameters 29

      2.4.5 Choice of Sorting Codes 31

      2.4.6 Simulation Code 32

      2.4.7 Implementation of the Problem Generator 34

      2.4.8 Implementation of the Simulation 36

3. **Experimental Result and Analysis**....39

3.1 Choice of Experiments and Notations....39

3.2 Simulation Results....42

3.2.1 Behaviour of the Algorithms for Constant Numbers of Jobs....42

3.2.1.1 Average performance Ratio: Makespan....43

3.2.1.2 Average Performance Ratio: Mean Completion Time....44

3.2.2 Behaviour of the Algorithms for Different Numbers of Jobs....46

3.2.2.1 Makespan as a Function of n....46

3.2.2.2 Average Completion Time as a Function of n.... 49

4. **Conclusions and Further Research Directions**....51

Bibliography 53

List of Tables

1.1 Worst Case Performance Ratios for Existing Algorithms 8

3.1 Mean, Standard Deviation and CV of Experiments 41

3.2 Makespan with n=100 and $a_i = 1$ 42

List of Figures

2.1 The Structure of Simulation Program 24

2.2 Flow Chart of Pseudo-Random Number Generator 35

2.3 Flow Chart of Suitable Distributed Random Number Generator 36

2.4 Flow Chart of Simulation Run 38

Appendices

Appendix A : Tabular Results

A.1 Mean and Standard Deviation of the Ratio of M/LB

A.2 Mean and Standard Deviation of the Ratio of C/C(LPTo)

A.3 95% CI of the Ratio of M/LB

A.4 95% CI of the Ratio of C/C(LPTo)

A.5 95% CI of the Ratio of M(LPTa)/M(LPTo)

A.6 Mean and Standard Deviation of the Makespan M

A.7 Mean and Standard Deviation of the Average Completion Time C

A.8 Mean and Standard Deviation of the Ratio of M/LB, by # of Jobs

A.9 Mean and Standard Deviation of the Ratio of C/C(LPTo) , by # of Jobs

A.10 95% CI of the Ratio of M/LB, by # of Jobs

A.11 95% CI of the Ratio of C/C(LPTo) , by # of Jobs

A.12 95% CI of the Ratio of M(LPTa)/M(LPTo), by # of Jobs

Appendix B : Graphical Results

B.1 Makespan vs. 1/m for U(0,1)

B.2 Makespan vs. 1/m for U(19,20)

B.3 Makespan vs. 1/m for U(5,10)

B.4 The Ratio of M(LPTa)/LB vs. 1/m, by # of Jobs

B.5 The Ratio of M(LPTo)/LB vs. 1/m, by # of Jobs

B.6 The Ratio of M(P(m))/LB vs. 1/m, by # of Jobs

Appendix C : Simulation Program

Chapter 1

Introduction

1.1 General Scheduling Problem

Job-shop scheduling problems arise in almost all areas of human activity. They occur whenever there is a choice as to the order in which a number of tasks can be performed. A problem could involve: jobs in a manufacturing plant, aircraft waiting for landing clearances, or programs to be run at a computing centre. Regardless of the character of the particular tasks to be ordered, these problems may be described in the common language of job-shop scheduling theory.

We may view scheduling in several ways. First, scheduling is a decision-making function: it is the process of determining a schedule. Second, scheduling is a body of theory: it is a collection of principles, models, techniques, and logical conclusions that provide insight into the scheduling function. Scheduling helps the decision-maker to efficiently utilize resources, respond rapidly to demand, and conform closely to prescribed deadlines.

Many job-shop scheduling problems can be described by specifying four types of information:

1. The jobs and operations to be processed.
2. The number and the types of machines that comprise the shop.
3. Disciplines that restrict the manner in which assignment of jobs to time slots and machines can be made.
4. The criteria by which a schedule will be evaluated.
5. In this research, each job consists of one task, and so the term job will be used interchangeable with task..

1.2 The Parallel Machine Scheduling Problem

In general, scheduling requires both sequencing and resource allocation decisions. When there is only a single resource, as in the single-machine model, the allocation of that resource may be completely determined by the sequencing decision. Consequently there is no distinction between the sequencing and allocation parts of the problem.

The focus in this research is not on single machine systems, but on systems with two or more identical processors in parallel. All n jobs are assumed to be available for processing at time 0, and each job can be performed equally well on each machine. In this case, the allocation part of the problem is to decide which jobs to process on which machines. Once the allocation has been made, the individual machines may be optimally or sub-optimally sequenced. More specifically, we shall deal with the problem of minimizing the maximum job completion time, also called the makespan, for scheduling n jobs on m identical machines (this problem is denoted by $P||C_{max}$). This problem and its well-known relatives the partition problem and subset-sum problem, have fascinated many researchers for several decades. Many excellent algorithms have been developed, and almost any reasonable level of performance may be obtained in modest computer time using various approximation schemes. We shall also examine the implications of the algorithms under study for the problem of minimizing a secondary objective, the mean job completion time, which is equivalent to minimizing the sum of the completion times of all of the jobs.

More specifically, the problem P||C_{max} is defined as follows:

Problem instance: a set J={1,....,n} of n jobs, where job i has non-negative processing time a_i, and m processors (n ≥ m).

Question: Partition the job set into m subsets J_1,...,J_m, so as to minimize the maximum sum of processing times (makespan) of the jobs in any of the subsets, where J_j contains the jobs which are assigned for processing on machine j.

The importance of this problem cannot be underestimated, both for its theoretical value and its practical applications. Detailed discussion of P||C_{max} is therefore a feature of every introductory job-shop scheduling book. In complex real-world systems, parallelism is a common feature, and an understanding the parallel machine problem enables the master scheduler to deal more effectively with the parallel portions of a complex network of processors.

It is unfortunate that the problem P||C_{max} cannot be solved optimally (in a practical sense) if the number of jobs becomes too great. The reason is that this problem is in the class of NP-hard problems [GJ], for which the only known optimal algorithms grow exponentially in computational time as the size of the problem grows. Nobody has yet been able to find an "efficient" optimal algorithm for any NP-hard problem (efficient meaning an algorithm whose computational time grows as only a polynomial function of problem size), nor has anybody proved that such a efficient algorithm does not exist. However, the general practice is to develop computationally efficient heuristic algorithms (non-optimal but often close to optimal) for such NP-hard problems, to enable large problems to be reasonably well solved.

For such heuristic procedures, we often evaluate the quality of the algorithm in two ways:

(i) worst case performance ratio: for each instance of a problem, compute the ratio of the heuristic value of the objective function obtained for a problem to the optimal value for that problem. The worst case performance ratio for an algorithm is the maximum of this ratio over all problem instances.

(ii) computational complexity: the number of computations (in effect, the computer time) required to solve any instance of the problem. The computational complexity of a problem is given as a function of the problem size (technically, problem size is the length of the input, i.e., number of binary bits, needed to specify a problem instance, but for most practical purposes, it may be stated as some natural parameter(s) of the problem, e.g., number of jobs, number of machines, etc.).

There are several heuristic algorithms for minimizing the makespan on parallel machines when processing times are known, including the well-known longest processing time (LPT) algorithm, which will play a major role in this research. The LPT heuristic procedure first orders the jobs by non-increasing processing times, and then schedules the jobs in order, assigning each job to the machine with the least amount of processing already assigned to it. Thus, the LPT algorithm is easy, intuitive, and reasonably fast computationally (running time $O(n \log n)$ for fixed m, where there are n jobs) and was shown by Graham[G] to have a worst case performance ratio of $4/3-1/3m$, where m is the number of machines.

Much more work on the LPT algorithm followed that of Graham [G]. Since the LPT algorithm is being used for comparison in this thesis, it is worth mentioning some particular results relating to its performance. The worst case performance ratio of $4/3-1/3m$ was generalized by Coffman and Sethi [CS] in 1976 to a bound of $(k+1)/k - 1/km$, where the latest finishing job is assigned by LPT to a machine with at least k-1 other jobs. This latter bound was shown to be tight for $k=1$ and for $k>=3$. Finally, Chen [C] provided a modified tight bound for the case of $k=2$.

1.3 Ordinal Algorithms

The pragmatic world is full of decision making problems in the presence of uncertainty, and many disciplines have responded with different approaches. There exist aids to decision making under uncertainty in many disciplines. For example, political decision making approaches (e.g., election), psychological approaches (group decision making techniques such as Delphi), and a host of mathematical approaches such as decision theory, sensitively analysis, statistical modelling, etc. In particular, for many practical problems including scheduling problems, it is often the case that our ability to find good solutions is severely limited by the availability of only incomplete information with regard to the problem structure or data. The desired data, i.e., processing times, may be available in only inexact form, as estimates, 'guestimates' or even worse. This problem is well recognised, and much work has been done to deal with it, e.g., statistics, decision theory, sensitivity analysis, etc. In scheduling problems, it is often difficult to obtain accurate time estimates for jobs. However, it may be much cheaper and easier to establish which of two jobs is likely to be longer, i.e., to order the processing times rather than measuring the individual processing times themselves.

The ordinal data model is a new approach to dealing with inexact data. The approach has its drawbacks, e.g., there is no guarantee that ordinal data will be available when exact data isn't. However, it is not unreasonable to think that it might be, e.g., the decision that one processing time is less than another may very well be possible, while exact estimates may not (the cake takes longer to prepare than the stew, but exact times for either may be difficult to determine). As a new approach, there is very little literature available, and all of it is cited in the literature review in this paper. It should be noted that results will shortly be completed (by W.-P. Liu and J.B. Sidney) on the use of ordinal data for bin packing (where at least some data must be known exactly) and for minimizing the deviation of linearly packed weights from a target centre of gravity.

With this is mind, we shall examine the effectiveness of algorithms for scheduling parallel machines when the order of the processing times is known, but the actual times themselves are not known. Thus, without loss of generality we may assume that $a_1 \geq a_2 \geq .... \geq a_n$. We refer to such data as ordinal data. Algorithms which utilize only ordinal data rather than actual magnitudes will be called ordinal algorithms. Ordinal algorithms will be evaluated by their worst case performance ratios with respect to the actual (unknown) data. Clearly, an ordinal algorithm for $P||C_{max}$ is given by a partition of the jobs between the m machines. The orders (permutations) of the jobs on each machine are irrelevant for the makespan objective function, but will be taken into consideration later when the secondary objective of average job completion time is considered.

For example, suppose m=2, n=2, J={ 1,2 } and by assumption $a_1 \geq a_2$. The optimal value C_{max} is a_1, and this is achieved by the algorithm which places job i on machine i. The algorithm which places both jobs on the machine 1 has a worst case performance ratio of 2, and this occurs for the case of $a_1 = a_2$.

There exist optimal ordinal algorithms for a number of well-known problems. The shortest processing time rule optimally solves the single machine and the identical parallel machine scheduling problems when the objective is to minimize mean completion time [CMM]. The greedy algorithm finds the minimal spanning tree [L1]. Many greedy algorithms have a strong ordinal "flavour" to them, e.g., the minimal spanning tree algorithm picks the next edge to be the smallest unused edge which does not destroy feasibility. An ordinal algorithm for the two machine no-wait flow shop problem which gives excellent solution values has also been developed [A], where by "no-wait", we mean that there can be no time delay between the completion of the first operation and the start of the second operation of a job. Many other problems are poor candidates for the ordinal algorithm approach. For example, the problem of scheduling n jobs on a single machine to minimize the

number of late jobs is such an example: without the actual values of processing times and due dates, it is impossible to determine if a job is late or not.

While our main focus, the parallel machine scheduling problem $P||C_{max}$, has been widely studied (see for example [LS]), the only previous work on ordinal algorithms for the problem is [LS], where algorithms for the 2, 3, 4 and m machine problems were presented and analyzed for worst case performance ratios. We briefly describe this last work, as it provides the basis for the current research.

A problem instance is given by a set of n processing times and the number m of machines. Let M denote the value of makespan given by a heuristic ordinal algorithm for such a problem instance, and let OPT denote the corresponding optimal value. Thus for any problem given by n and m, the measure of the quality of an algorithm will be given by the worst case performance ratio max { M/OPT }, where the maximization is taken over all instances of the problem.

In [LS] ordinal algorithms for the cases of 2, 3 and 4 machines are presented. For the cases of 2 and 3 machines, the algorithms achieve the minimum possible worst case performance ratio over all problem instances. For example, for m=2 the worst case performance ratio is shown to be 4/3, and this ratio is tight. The tightness implies there are instances which yield the worst case performance ratio of 4/3. Moreover, no other ordinal algorithm (i.e., assignment of jobs to machines by rank) can produce a worst case performance ratio less than 4/3. For the 4 machine case, the algorithm has a worst case performance ratio very close to the theoretical lowest ratio. For the case of an arbitrary number m > 4 machines, an algorithm is presented that is guaranteed to produce a solution no worse than (5m-2)/3m times the optimal value. Finally if the number of machines m gets arbitrarily large, the best possible ordinal algorithm is shown to have worst case performance ratio of at least 1.52368. It

is shown that for m=18, 19 or m ≥ 21, the smallest worst case performance ratio for the m machine problems exceeds 3/2.

The table below summarizes the results in [LS]:

Problem Name	Number of Machines	Worst Case Performance Ratio	Ratio Tight?
$P(2)\|\|C_{max}$	2	4/3	yes
$P(3)\|\|C_{max}$	3	7/5	yes
$P(4)\|\|C_{max}$	4	13/9 *	?
$P(m)\|\|C_{max}$	m	(5m-2)/3m	Yes for $m=4^k$, $k \geq 2$
$P(m)\|\|C_{max}$	as m --> inf	≥ 1.52368	?

* Lowest worst case ratio known to be at least 10/7.

Table 1.1 : Worst Case Performance Ratios for Existing Algorithms
(Ratio of Heuristic to Optimal Solution Values)

1.4 Problem to Be Studied

In this thesis, we shall use simulation to examine the effectiveness of algorithms for the parallel machine scheduling problem which use only ordinal data. Computer simulations will be used to study the makespan and average completion time of the jobs. Simulations will be based on many different distributions, to facilitate the identification of the kind of data configurations that yield good (poor) results using ordinal algorithms.

A major shortcoming of the previous work just described [LS] is that worst case performance ratio is often very unrepresentative of "typical" performance of an algorithm, which for most problems is usually much better than this pessimistic ratio. In addition, while practitioners may not know exact values of processing times, they may have some idea as to their distributions, e.g., the distribution may be symmetrical, have a low coefficient of variation (CV), take on only two values, etc. The reason of the CV was considered a critical factor that a low CV would represent data which is "relatively" constant. For example, a CV of 0 would represent data for which all processing times are the same. In such a case, LPTa and LPTo are optimal, and there is no problem. However, a high CV can lead to a few large jobs producing large variation between time assignments to machines. Thus, while the worst case ratio may be high (say, for example, five-thirds), it may be that for data of the type being considered (for example, low coefficient of variation) the algorithm tends to perform very well. Thus, we would like to deal with several issues related to average performance ratio (rather than worst case performance ratio):

1. for what data configurations (i.e., data distributions) do these algorithms give good (poor) results, where good means competitive with the LPT algorithm using exact values of processing times.

2. how well do ordinal algorithms compare in performance with typical good heuristic algorithms which use full data (i.e., actual processing times)?
3. how does the effectiveness of the algorithms examined differ for different problem sizes (given by n and m), and for different data distributions?

1.5 Thesis Outline

Chapter 2 lays out the foundations of the study: definitions, notation, description of algorithms under study, theoretical considerations, description of the experiments to be performed, description of the simulation methodology being used, description of the software system developed for current and future studies. In Chapter 3 the simulation results are presented, discussed and analyzed. Chapter 4 contains general conclusions and suggestions for further study.

Chapter 2

Problem Definition and Experimental Design

2.1 Definitions and Notations

We now give the details of the algorithms under study and discuss the structure of the experiments to be performed. As mentioned in the previous chapter, we let M denote the value of makespan, (i.e., C_{max}) for the heuristic solution given by any of the algorithms for some problem, and OPT the corresponding optimal value. Thus, for any problem the measure of the quality of an algorithm will be given by the worst case performance ratio max{ M/OPT }, where the maximization is taken over all instances of the problem. We also use the secondary performance measure average job completion time, which is easily minimized (to be discussed below) for any assignment of jobs to machines.

The following notations are given :

M(LPTa) = makespan under algorithm LPTa
M(LPTo) = makespan under algorithm LPTo
M(P(m)) = makespan under algorithm P(m)
LB = lower bound for makespan problem
C(LPTa) = average job completion time under algorithm LPTa
C(LPTo) = average job completion time under algorithm LPTo
C(P(m)) = average job completion time under algorithm P(m)

The LPT algorithm has already been described. We shall use it in two forms:

1. LPTa denotes the LPT algorithm applied to the actual data a_i;
2. LPTo denotes the LPT algorithm applied using ordinal data: associated with the i-th smallest job is the (pseudo-)processing time i.

Algorithm LPTa

1. Construct an ordering of jobs (from large to small).
2. At each iteration assign the next job to the machine with the least amount of previously assigned processing time.

In the implementation of the LPTa algorithm, if there is a tie for the machine with the lowest time assignment, the next job is allocated to machine i, where i is the smallest possible index. The worst case performance of LPTa is (4/3 - 1/3m) as previously mentioned.

Algorithm LTPo

For the m machine case, we assign jobs 1,....,m to machines 1,....,m, then jobs m+1,....,2m to the machines in reverse order, jobs 2m+1 to 3m in the original order, and so on. This amount to applying LPTa to the ordinal data set { 1, ... , n } where i represents the i-th largest job.

The assignments given by algorithm LPTo for the case of m=3 are illustrated below for m=3.

Machine 1 : 1 6 7 12 13 .....
Machine 2 : 2 5 8 11 14 .....
Machine 3 : 3 4 9 10 15 .....

The worst case performance ratio of LPTo for the makespan problem is at least 5/3, as can be seen by applying LPTo to the data set { 1, 1/3, 1/3, 1/3, 1/3, 1/3, 1/3 }.

The ordinal algorithms from [LS] are described below, along with information about their worst case performance ratios. For each value of m, the associated algorithm is described by specifying the set of jobs assigned to each of the m machines. Before detailing the algorithms, a few comments are in order. The first instinct of the authors of [LS] was to use the algorithm LPTo as a good ordinal algorithm, but as illustrated previously, LPTo gives a worst case performance ration of 5/3, which is higher that the P(m) algorithms achieve. In fact, there appears little intuitive about the structure of the P(m) algorithms. This was the reality: the authors developed each one separately, and were unable to detect a pattern which would enable the three special algorithms P(2), P(3) and P(4), whose worst case performance is quite good, to be generalized into P(5), P(6), ... Thus, for i>4, the algorithm P_m has been used instead of a more specialized and better performing algorithm.

Algorithm P(2)

The machine assignments are:

M1: $\{1+3i \mid i \geq 0\}$

M2: $\{2+3i \mid i \geq 0\} \cup \{3+3i \mid i \geq 0\}$

The resulting schedule is illustrated below:

Machine 1 :	1		4		7		.....
Machine 2 :	2	3	5	6	8	9	.....

Algorithm P(2) gives a worst case performance ratio of 4/3, and no other algorithm can do better. For example, let α = completion time on M1. Then:

$$\alpha = a_1 + a_4 + a_7 + ... \leq a_1 + \frac{1}{3}(a_2 + a_3 + ...) = \frac{2}{3}a_1 + \frac{1}{3}b \leq \frac{4}{3}\max\{a_1, b/2\}.$$

Since $\max\{a_1, b/2\}$ is clearly a lower bound on makespan . it follows that the time allocation to M1 is less than 4/3 times this lower bound. A similar argument applies for machine 2. The worst case performance ratio of 4/3 is achieved by the instance given by processing times { 1, 1/3, 1/3, 1/3}.

Algorithm P(3)

The machine assignments are:

M1: $\{1+5i \mid i \geq 0\}$

M2: $\{2\} \cup \{5i \mid i \geq 0\} \cup \{8+5i \mid i \geq 0\}$

M3: $\{3,4\} \cup \{7+5i \mid i \geq 0\} \cup \{9+5i \mid i \geq 0\}$

The resulting schedule is shown below:

M1 :	1	6		11		16		21	...
M2 :	2	5	8	10	13	15	18	20	...
M3 :	3	4	7	9	12	14	17	19	...

Algorithm P(3) gives a worst case performance ratio of 7/5, and no other algorithm can do better.

Algorithm P(4)

The machine assignments are:

M1: $\{1+7i \mid i \geq 0\}$

M2: $\{2+14i \mid i \geq 0\} \cup \{7+14i \mid i \geq 0\} \cup \{11+14i \mid i \geq 0\}$

M3: $\{3\} \cup \{6+14i \mid i \geq 0\} \cup \{9+14i \mid i \geq 0\} \cup \{12+14i \mid i \geq 0\} \cup \{14+14i \mid i \geq 0\} \cup \{18+14i \mid i \geq 0\}$

M4: $\{4\} \cup \{5+14i \mid i \geq 0\} \cup \{10+14i \mid i \geq 0\} \cup \{13+14i \mid i \geq 0\} \cup \{17+14i \mid i \geq 0\}$

The resulting schedule is shown below:

M1 :	1 8	15 22	29 36	...
M2 :	2 7	11 16 21	25 30 35	...
M3 :	3 6	9 12 14 18 20	23 26 28 32 34	...
M4 :	4 5	10 13 17 19	24 27 31 33	...

Algorithm P(4) gives a worst case performance ratio of 13/9, and no other algorithm can do better than 10/7, which is only a bit more than one percent below 13/9.

Algorithm P_m

For m>=5 jobs are assigned to machine i as follows:

1. For $1 \leq i \leq \lfloor m/2 \rfloor$, $\{a_i\} \cup \{a_{2m+1-i+k(m+\lceil m/2 \rceil)}\}$, $k \geq 0$
2. For $\lfloor m/2 \rfloor + 1 \leq i \leq m$, $\{a_i\} \cup \{a_{2m+1-i+k(m+\lceil m/2 \rceil)}\} \cup \{a_{3m+1-i+k(m+\lceil m/2 \rceil)}\}$, $k \geq 0$

The assignments given by algorithm P(m) for the case of m=7 are illustrated in below:

Machine 1:	1	14		25		36		....
Machine 2:	2	13		24		35		....
Machine 3:	3	12		23		34		....
Machine 4:	4	11	18	22	29	33	40	....
Machine 5:	5	10	17	21	28	32	39	....
Machine 6:	6	9	16	20	27	31	38	....
Machine 7:	7	8	15	19	26	30	37	....

For any m, algorithm P_m gives a worst case performance ratio no greater than (5m-2)/3m. Note that algorithm P(3) is in fact P_m with m=3, i.e., P_3.

Once the above algorithms have been applied to obtain makespan, the jobs will have been partitioned into subsets for each machine. To minimize average job completion time subject to this partition, the jobs on each machine will be put into shortest processing time order, that is, from smallest to largest. In other words, LPT is used to partition jobs between machines, and then SPT is used to order each machine.

The following two lemmas establish some general properties which provide direction in the research for 'good' algorithms for $P||C_{max}$; proofs are given in [LS].

Lemma 1. Suppose that an algorithm for $P||C_{max}$ with m machines does not have the following two properties:

1. jobs 1 to m are assigned to different machines;
2. jobs i and 2m+1-i are assigned to the same machine.

Then the algorithm has a worst case performance ratio of at least 3/2.

Lemma 2. Suppose that an algorithm for $P||C_{max}$ with m machines and n jobs schedules n_i jobs on machine i. Then the worst case performance ratio is at least $\text{Max}\left(\{n_i\}/\lceil n/m \rceil\right)$.

As mentioned above, for any job assignment to machines given by an algorithm for $P||C_{max}$, the average job completion time can be minimized subject to the above assignment of jobs to machines by placing the jobs on each machine in shortest processing time order (SPT). This will enable us to compare algorithmic performance on the secondary measure, average completion time of jobs.

An additional tool we shall use in evaluating the quality of makespan solutions is the easily calculated lower bound $LB = \max\{a_1, a_m+a_{m+1}, b/m\}$, where b is the sum of job processing times, $b = \sum_i a_i$. For the average job completion time objective, one of the algorithms we shall use will actually yield the minimum, and this minimum will be used for comparison.

To summarise, for each problem we will compare the makespans produced by three algorithms with each other and with a lower bound. In addition, we shall compare the average job completion times produced when the machine assignments produced by the algorithms are placed in shortest processing time order, and compare the values so obtained with the optimal (which one of the three algorithms produces). The three algorithms which will be compared for each random problem generated are:

1. the longest processing time (LPT) algorithm applied to actual data, LPTa
2. the LPT algorithm applied to ordinal data, LPTo
3. the appropriate ordinal algorithm from [LS], as determined by the number of machines, namely P(2), P(3), P(4) or P_m. The notation P(m) will be used to denote any of these algorithms.

In each case, the SPT algorithm will be applied after the jobs have been assigned to machines in order to minimize the secondary performance measure average job completion time. We note that the LPTo algorithm above yields the actual minimum over all schedules for average job completion time, and this serves as our lower bound on average job completion time. The kinds of simulation experiments reported in this research typify those done to compare various scheduling algorithms, see for example [CMM].

As an example, suppose m=3, n=10 and the job times are 75, 67, 45, 41, 27, 26, 23, 15, 7, 3. Then the three algorithms above yield the solutions below, where each machine is in SPT order.

The assignments of jobs using the LPTa algorithm and then reordering the jobs in each machine by SPT are:

Machine 1 : 3 7 26 75

Machine 2 : 15 27 67

Machine 3 : 23 41 45

In the above table, the jobs allocated to each machine have been ordered in shortest processing time order. Thus, mean completion time equals :

$$[3+ (3+7) + (3+7+26) + ... + (23+41+45)]/10 = 52.2$$

The total processing time is 111 on machine 1, 109 on machine 2, and 109 on machine 3. Thus the makespan of the LPTa algorithm is 111. The average job completion time of the jobs is 52.2.

The assignments of jobs using the LPTo algorithm are:

Machine 1 : 23 26 75

Machine 2 : 15 27 67

Machine 3 : 3 7 41 45

The total processing time is 124 on machine 1, 109 on machine 2, and 96 on machine 3. Thus the makespan of the LPTo algorithm is 124. The average job completion time of the jobs is 52.2.

The assignments of jobs using the P(3) algorithm are:

Machine 1 : 26 75

Machine 2 : 3 15 27 67

Machine 3 : 7 23 41 45

The total processing time is 101 on machine 1, 112 on machine 2, and 116 on machine 3. Thus the makespan of the P(3) algorithm is 116. The average job completion time of the jobs is 52.9.

The lower bound of this example is LB = 109.67 = max {75, 41+45, 329/3}.

Thus, for the data given above, the makespan of algorithm LPTa is closest to the lower bound, next comes P(3), and finally LPTo.

For P(3), the ratio of M to LB is 116/109.67 = 1.0577, which is well within the theoretical upper bound of 7/5 for this algorithm. The average completion time of 52.2 for LPTo is (as stated previously) optimal, with LPTa equal to the optimal and P(3) within 2 percent of the optimal for this example.

2.2 Using Simulation

The two main objectives of the thesis are to examine the performance of ordinal algorithms for the problem $P||C_{max}$ and to develop a software system to facilitate examination of the performance of other ordinal algorithms. All the algorithms will be evaluated for makespan by their performance ratios with respect to the lower bound and with respect to each other. In addition, we examine how well the algorithms perform on the secondary measure of performance, expected job completion time. In particular, we anticipate that the worst case performance ratios are not always a good indicator of average performance, and also that the algorithms that minimize worst case performance ratio may not control average job completion time very well.

Using the PASCAL language, a software system was developed consisting of a simulation shell to control the simulation experiments, and the simulator itself. The output was carefully designed to allow convenient (automated) use of available statistical software for analysis of the results. A typical run of the simulator for an m-machine n-job problem involves generation of a large number of random problems, heuristic solution of each problem by the Longest Processing Time (LPT) algorithm , LPTa, solution of an ordinal version of each problem (in which we substitute the rank of a_i in place of its value) by LPTo and by the appropriate ordinal algorithm given in [LS]. The actual values of a_i are used with the solutions found by ordinal methods to compute comparisons of the quality of solutions, i.e., makespan and average completion time, between the three algorithms. All objective function values are compared with a lower bound calculated for the problem. In addition, for each solution to $P||C_{max}$, the jobs on each machine are ordered by non-decreasing processing time to obtain the best average completion time for the particular assignment of jobs to machines, and the three algorithms are compared with respect to average job completion time. As previously mentioned, the second algorithm, namely

the LPTo algorithm, will yield the optimal value for the average job completion time, so that our comparisons of performance for average job completion time will be made against the optimal.

While the control shell for the simulation software is relatively simple, we divide the simulator itself into two main segments, one to generate pseudo-random problems and the other to run them and output results. The output segment produces, for each problem, the makespan and average job completion time for three algorithms (LPTa using actual data, LPTo using ordinal data, and the ordinal algorithm P(m) from [LS]), and also computes a lower bound for the makespan problem.

2.3 Experiments to Be Performed

A single simulation run is insufficient to examine the performance of algorithms. We require a set of results based on the same problem size and the same distribution of job times. More than one simulation run can be executed in our simulation model, i.e., the number of repetitions is greater than one. For each repetition of each problem size and job-time distribution, seven numbers will be computed for use in comparisons:

1. the makespan for the LPTa algorithm, M(LPTa);
2. the makespan for the LPTo algorithm, M(LPTo);
3. the makespan for the P(m) algorithm M(P(m));
4. the average job completion time for the LPTa algorithm, C(LPTa);
5. the average job completion time for the LPTo algorithm, C(LPTo);
6. the average job completion time for the P(m) algorithm, C(P(m));
7. a lower bound on the optimal makespan : LB = MAX { a_1 , a_m+a_{m+1} , b/m }.

For each problem investigated, the simulation output will include:

(1) data describing the method by which data was generated (to permit replication of the experiment): ***n*** (number of jobs), ***m*** (number of machines), ***disttype*** (distribution used), ***rep*** (number of repetitions of the experiment), ***seed*** (for pseudo-random number generator);

(2) for each repetition, the makespan given by each algorithm used (3 numbers, one for each algorithm), the calculated lower bound (1 number), and the average job completion time when the jobs are divided between the machines according to the makespan algorithm used, and if these are sequenced on the machines in the shortest processing time order (3 numbers, one for each algorithm).

Using standard statistical tools, the makespans of the ordinal algorithms, LPTo and P(m), will be compared with each other, and with the makespan of the LPTa algorithm and the lower bound. The comparisons with the LPTa results will indicate how well the ordinal algorithms compare with a typical good algorithm which uses actual data, and the comparison with the lower bound will give an indication of the quality of the solutions obtained.

Additionally, the average completion times for the various algorithms will be computed, and compared, to indicate the implications of the various makespan algorithms for this secondary performance measure.

2.4 Methodology

Computer simulation and statistical analysis of the results of these simulations are the main tools used to answer the questions:

1. for what data configurations (i.e.. data distributions) do these algorithms give good (poor) results, where good means competitive with the LPT algorithm using exact values of processing times.
2. how well do ordinal algorithms compare in performance with typical good heuristic algorithms which use full data (i.e., actual processing times)?
3. how does the effectiveness of the algorithms examined differ for different problem sizes (given by n and m), and for different data distributions?

The structure of the simulation program is given in Figure 2.1.

For the three kind of algorithms, LPTa, LPTo and P(m), various representative problem sizes, given by the values of the number of jobs, n, and the number of machines, m, will be examined for data drawn from various distributions:

1. Uniform distribution U(a,b), $0 \leq a \leq b$.
2. Truncated normal distribution N(μ, σ), truncated from below at 0. μ and σ must be positive numbers.
3. Shifted exponential E(λ, shift) : both d and the shift must be greater than zero. The probability density function is given by

$$f(t) = \lambda e^{-\lambda(t-shift)}$$

A user-specified distribution is permitted to allow the software to be used in later simulations. This feature was not applied in the current research.

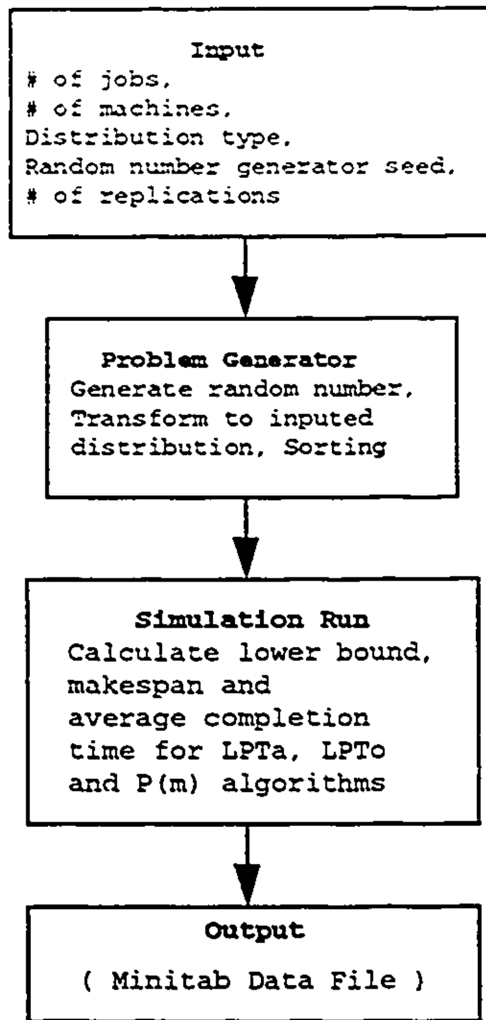


Figure 2.1 The Structure of the Simulation Program

2.4.1 Choice of Environment and Program Language

In this thesis research, the PASCAL programming language is used to construct the simulation programs. There are several advantages of choosing PASCAL:

PASCAL is a well-known programming language that most computer-literate workers are able at least to read and understand.

PASCAL is widely available on computers: the Borland Turbo Pascal compilers are inexpensive and, in particular, are installed on several University of Ottawa networks, making them convenient for this research.

Because PASCAL is a strongly-typed programming language, each data structure such as a variable or array must be declared before it is used. This reduces programming errors, especially those that result when a named data structure is used in different ways in different places in the program.

There are some disadvantages of our choice, however, such as the limited working memory of Turbo PASCAL. Data structures are normally not allowed to use more than 64K bytes of memory. This limitation will not affect our simulation runs since we do not need to use that much working memory.

2.4.2 Choice of Random Number Generator

The two major parts of the simulation program are the random number generator and the simulation "run". For each repetition of the simulation for fixed m and n, we must first generate a suitable value for the time required for each of the n jobs. We also must rank these n values. The value and rank of these random numbers are assumed to be the exact value and rank of the job processing times on the machines we wish to schedule.

The paradox of random number generation presented by John von Neumann in [WH] is:

> The random number sequences that computers produce are not truly random at all, since true randomness depends upon having a random process available, such as tossing a coin. Because such a process is not available within a computer, [we] use a pseudo-random process to produce sequences of numbers that appear to be random and mimic well the behaviour that is expected to be true for random sequences.

Such a process is not only easier to use than a truly random process, but it also produces numbers for which the generation can be repeated if necessary. In testing a program, for example, it is troublesome if every run is necessarily different because of different random numbers. Pseudo-randomness lets us use the same numbers over and over again for testing but use different ones for each production run.

The ideal random number generator:

- should be repeatable across computing environments, so that when moving a program from one machine to another the same numbers will be generated;
- should be written in a high-level language so that the code can port from one machine to another one without difficulty;
- should be efficient in computing effort.

The algorithm presented by Wichmann and Hill meets all criteria for an effective pseudo-random number generator [WH]. Moreover, it has a long period. That is, it does not repeat itself until very many (approximately 10^{12}) numbers have been generated. This generator was also available in Turbo PASCAL, putting it ahead of some other choices.

Like most pseudo-random number generators, the Wichmann-Hill algorithm produces numbers that are approximately uniformly distributed on the unit interval. We must transform the uniform [0,1] pseudo-random numbers into other distributions:

1. Uniform (a,b).
2. Truncated normal (μ, σ^2), truncated at 0.
3. Shifted Exponential (λ, shift).

2.4.3 Transformation to other distributions

(1) Uniform distribution on interval [a, b]:

The following transformation of unit uniform pseudo-random numbers produces pseudo-random numbers on the interval [a,b]. Let s_i, i=1,2,...,n, represent the uniform [0,1] pseudo-random numbers. Let x_i, i=1,2,....,n, represent the numbers distributed uniformly on [a,b]. Each s_i has a corresponding number x_i by inverse transformation:

$$x_i = a + (b-a)s_i.$$

(2) Normal distribution with parameters μ and σ, with truncation at zero:

The following transformation of unit uniform pseudo-random numbers produces (pseudo) random selections from the normal distribution (using the Box-Müller method [JD]), where the mean μ and the standard deviation σ are both positive. Let s_i, i=1,2,...,n, represent the uniform [0,1] pseudo-random numbers. Let y_i, i=1,2,....,n, represent a number drawn from the standard normal distributed number N (0,1). Let x_i, i=1,2,....,n, represent the number distributed as N(μ,σ). If $x_i < 0$, reject it and regenerate a new x_i.

With two independent random variables R_1 and R_2,

$$R_1 = 2s_i - 1 \quad and \quad R_2 = 2s_{i+1} - 1$$
$$U = R_1^2 + R_2^2$$
$$If\ U < 1$$
$$Then\ y_i = \{-2\ln U\}^{1/2} R_1 U^{1/2}$$
$$y_{i+1} = \{-2\ln U\}^{1/2} R_2 U^{1/2}$$
$$x_i = \mu + \sigma y_i$$

If $x_i < 0$, then regenerate x_i.

Note that the truncation is needed because our job times cannot be negative, but this truncation causes the distribution of times that results to have a mean and standard deviation slightly different from μ and σ.

(3) Shifted exponential distribution:

The following transformation of unit uniform pseudo-random numbers produces shifted exponential random numbers with parameters d and shift, where both of the parameters are positive numbers. The reason for using the shifted exponential distribution instead of the 'pure' exponential distribution is to match the mean and standard deviation to that of the uniform distribution. Therefore, the shifted exponential is comparable to the uniform distribution and Normal distribution under the same mean and standard deviation.

Let s_i, $i=1,2,...,n$, represent the uniform [0,1] pseudo-random numbers. Let x_i, $i=1,2,...,n$, represent the numbers we want. Each s_i has a corresponding number x_i.

$$x_i = shift + ln((1-s_i)^{-1})/\lambda$$

2.4.4 Choice of parameters

It was felt that the coefficient of variation might be a significant factor in the behaviour of the algorithms, as well as the general shape of the distributions. Therefore, three types of distributions were used : the uniform, the truncated normal, and the shifted exponential. For each distribution, three sets of parameters were used (for a total of nine different distributions), in such a way that each type of distribution was run with approximately equivalent values of mean and standard deviation.

As mentioned before, three data distributions are examined: uniform U (a.b): truncated normal N (μ, σ^2) and shifted exponential E (λ,shift). In order to compare the three algorithms under study, we have matched mean and standard deviation for these three distribution types.

For examples, let μ_1 and σ_1 equal to the mean and standard deviation of uniform distribution; let μ_2 and σ_2 equal to the mean and standard deviation of Normal distribution; and let μ_3 and σ_3 equal to the mean and standard deviation of shifted Exponential distribution.

Then for Uniform distribution U(a,b)

$\mu_1 = (a+b)/2$

$\sigma_1^2 = (b-a)^2/12$

Normal distribution N(μ,σ)

set $\mu_2 = \mu_1$, and $\sigma_2 = \sigma_1$. Except for one scenario to be mentioned later, the truncated normal will for all practical purposes be identical to the normal in our work.

Shifted exponential E (λ,shift)

The probability density function is:

$$f(t) = \lambda\, e^{-\lambda(t-shift)}$$

Then

$$\begin{aligned}\mu_3 &= \int_{shift} t f(t) dt \\ &= shift + 1/\lambda\end{aligned}$$

$$
\begin{aligned}
\sigma_3^2 &= \int_{shift}^{\infty} (t-\mu_3)^2 f(t)\,dt \\
&= 1/\lambda^2
\end{aligned}
$$

For $\sigma_1^2 = \sigma_3^2$, we have

$$
\lambda^2 = \frac{\sqrt[4]{12}}{\sqrt{b-a}}
$$

For $\mu_1 = \mu_3$, we have

$$
shift = \frac{b+a}{2} - \frac{\sqrt{3}}{6}(b-a)
$$

In the program, the sample means and sample standard deviations of the data for each distribution are calculated for comparison with their theoretical values.

2.4.5 Choice of Sorting Codes

Once the (pseudo)-random draws from a distribution are generated, the numbers drawn are the actual values of job processing times on the machines. The LPTa algorithm uses data to calculate the makespan and average completion time of jobs, but the ordinal algorithms LPTo and P(m) use the ranks of these numbers rather than their values. In any event, we sort the randomly generated job processing times to provide this ranking .

There are several techniques for sorting arrays, such as bubble sort, linear insertion sort, quicksort, shellsort, heapsort and interpolation sort. Based on the efficiency of these sorting algorithms, such as the running time, data file size, stability and computer stack requirement, quicksort was chosen to be the sorting algorithm in this simulation model.

2.4.6 Simulation Code

We have already seen that the first part of the simulation process, and of our program, is the 'Problem Generator' that generates data corresponding to or mimicking a real system of interest. The other part of our program is the 'Simulation Run' which runs and tests the simulated model under different requirements or environments. In both parts of our program, it is important that we save information that allows for analysis and conclusions. We will perform our analysis using standard statistical software (Minitab and Stata). The major advantage of separating the simulation program into several parts is that we can isolate any errors made. Thus, the simulation program consists of several individual programs, which eases the task of program checking and testing. All the individual programs can be linked together by using a batch file or execution shell.

The problem generator consists of:

1. Initializing system parameters:
   - number of jobs, represented by ***n***;
   - number of machines, represented by ***m***;
   - number of repetitions, represented by ***rep***;
   - random number generating seed, represented by ***seed***;
   - data configuration type, represented by ***disttype***, where 'U' for Uniform, 'G' for truncated Normal and 'E' for shifted Exponential;

- parameters for data configuration type, such as interval *a* and *b* for uniform, μ and σ for truncated normal and λ and ***shift*** for shifted exponential:

2. Generating pseudo-random numbers
3. Generating suitably distributed pseudo-random draws from the distribution, based on the Wichmann-Hill uniform generator
4. Sorting these random numbers

The values so obtained are assumed to be the job processing times on machines.

The simulation run consists of :

1. Initializing simulation variables;
2. Calculating the longest processing time and the average completion time of jobs by using LPTa algorithm, M(LPTa) and C(LPTa);
3. Calculating the longest processing time and the average completion time of jobs by using LPTo algorithm, M(LPTo) and C(LPTo);
4. Calculating the longest processing time and the average completion time of jobs by using P(m) algorithms, M(P(m)) and C(P(m));
5. Calculating the lower bound for makespan problem, LB.

Note that steps 2, 3, and 4 each generate two "result" numbers, and step 5 generates one, for a total of seven.

2.4.7 Implementation of the Problem Generator

The problem generator was programmed in PASCAL using the algorithms mentioned in Section 2.4.6.

1. pseudo-random numbers:

One of the system parameters is the random number generating seed. which is initialized at the beginning of the program. All the n pseudo-random numbers are generated based on this seed. This is transformed into three seed values needed by the Wichmann-Hill uniform pseudo-random number generator. This generator has been isolated in the code segment ***whrng.pas***. but the three seed values are global variables to our calling program. ***make.pas***.

The flow chart of the pseudo-random number generating algorithm is shown in figure 2.2.

2. Transformation of uniform pseudo-random numbers to desired distributions

We transform the uniform pseudo-random numbers on the unit interval using the methods discussed above in Section 2.4.2. A flow chart is presented in Figure 2.3.

3. Sorting

An implementation of Quicksort was adapted from code initially obtained from TUG. the Turbo Pascal Users' Group.

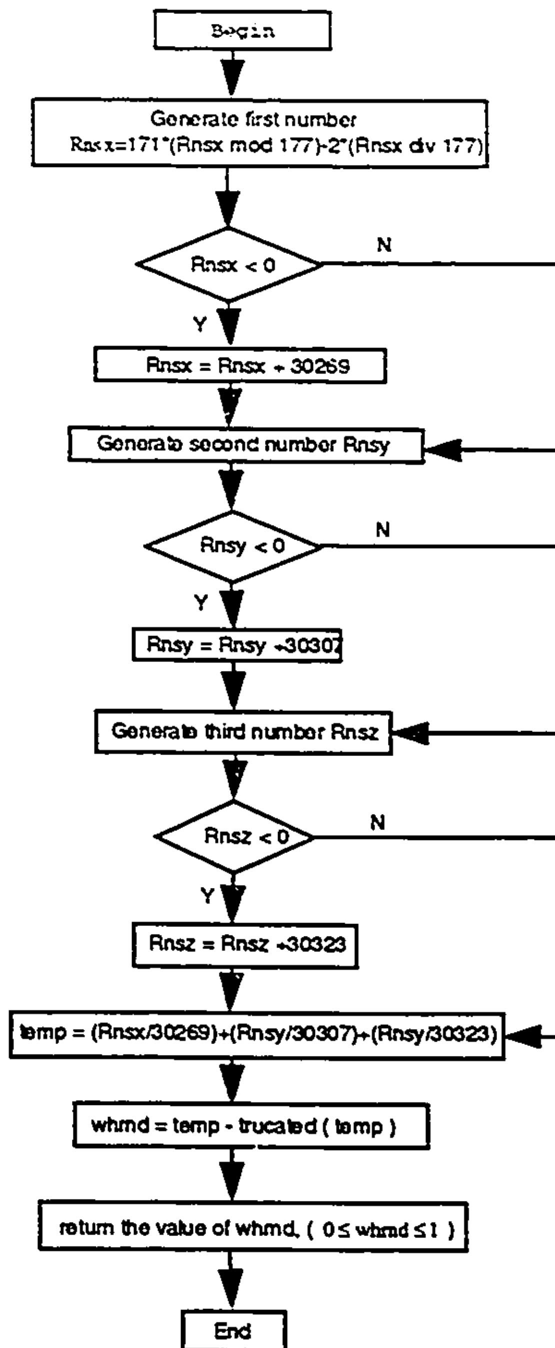


Figure 2.2 Flow Chart of Pseudo-Random Number Generator

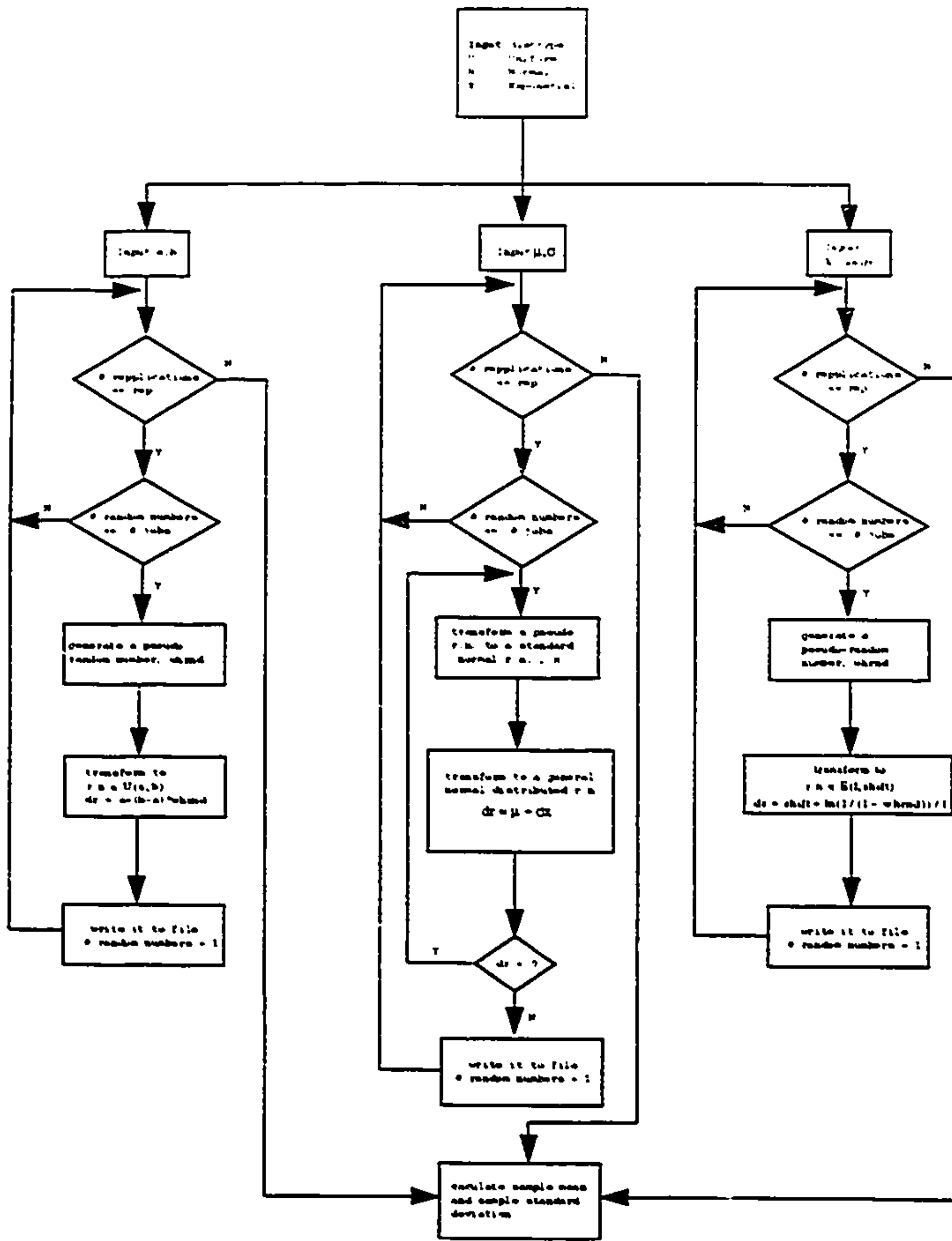


Figure 2. 3 Flow Chart of the Tansformation of the Random Numbers

2.4.8 Implementation of the Simulation

The various algorithms in this study were programmed as directly as possible in PASCAL, along with the lower bound of the makespan problem. There are seven data values output for each single simulation replicate as mentioned above. Since there are ***rep*** replications of each setting of the number of machines, number of jobs, and distribution type (all starting from a single value of a seed for pseudo-random number generation), there will be 7****rep*** numbers to store for each "run" we set in motion.

The statistical analysis is based on these data to examine the performance of all the actual algorithm and the ordinal algorithms. To make the task of analysis easier, we output the results in the form that can be directly executed within one of the Minitab or Stata statistical packages. That is, our PASCAL "write" statements included appropriate syntax to set up and execute commands for these packages. This saves us work when errors or new ideas cause us to alter our simulation. It does, however, require discipline in our choice of variable names for our output data. We also spent a considerable effort to include comment lines in the output files to allow us track our work.

The flow chart of the simulation run is shown in Figure 2.4.

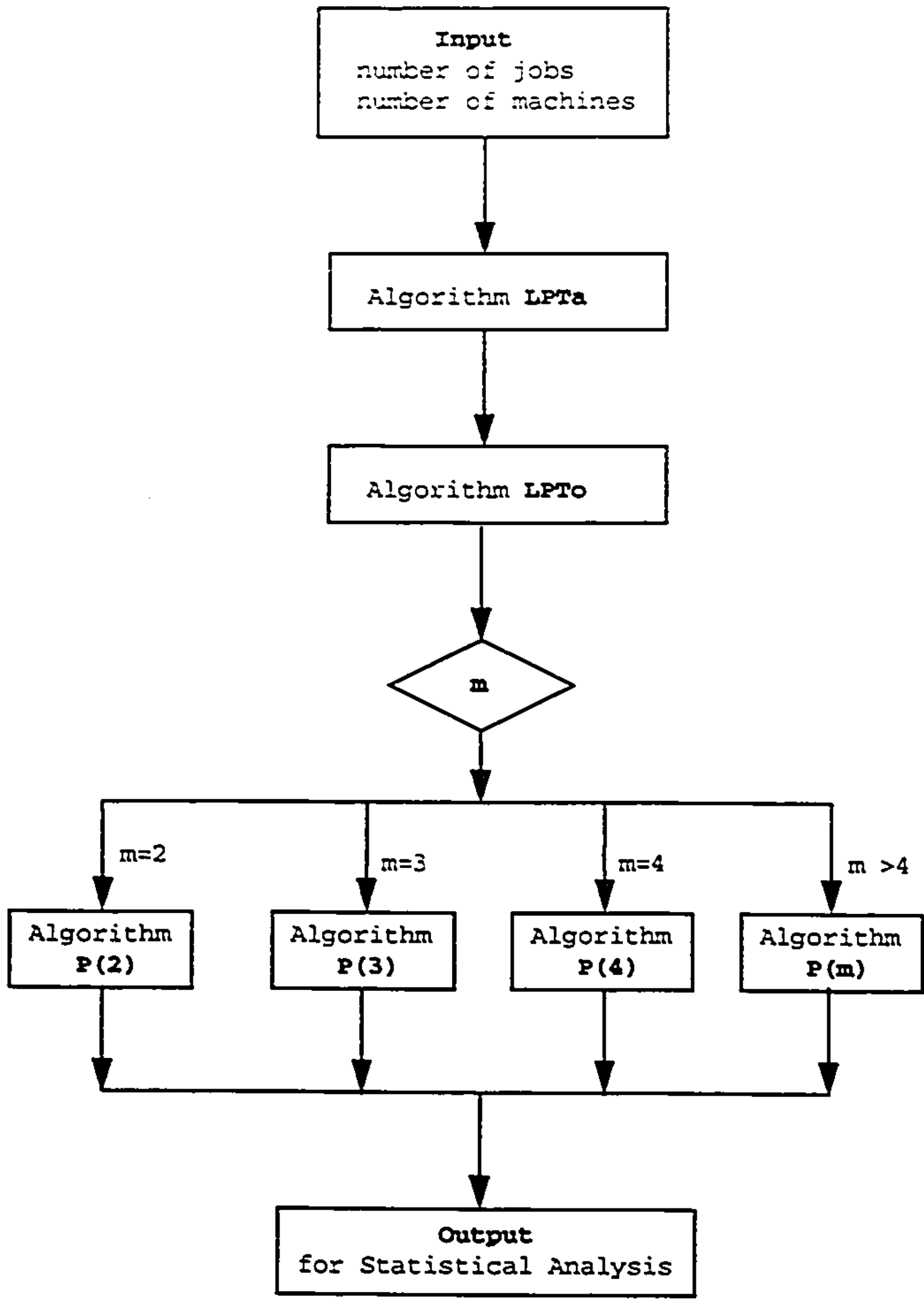


Figure 2. 4 Flow Chart of Simulation Run

Chapter 3

Experimental Result and Analysis

3.1 Choice of Experiments and Notations

The basic considerations in deciding what experiments to perform in this first investigation of the empirical properties of ordinal algorithms for the parallel machine makespan problem were:

1. Use of distributions with fundamentally different shapes. The uniform and truncated normal represent symmetrical (near symmetrical) distributions, while the shifted exponential represents a highly skewed distribution.
2. Matching of coefficients of variation. As mentioned, it was felt that the coefficient of variation might be a significant factor in the performance quality of the ordinal algorithms. Each of the three distribution types was run with the same mean and standard deviation, therefore the same coefficient of variation.

The three coefficients of variation (CV) used were .5774, .1925, and .0148, the first having been selected because it is the largest one achievable with the uniform distribution (The CV for the uniform distribution is $\frac{1}{\sqrt{3}}(\frac{b-a}{b+a})$. Since $0 \le a < b$ in the current research, the maximum CV is $\frac{1}{\sqrt{3}}=$ 0.5774, and this occurs for a=0.). In retrospect, it would have probably been useful to try some distributions with still greater coefficients of variation, and this is proposed for future work. Table 3.1 shows the values of the parameters of each distribution used to obtain the desired coefficients of variation. In the case of the truncated normal, we matched the coefficient of variation of the non-truncated normal. For all but the first case (CV = .5774), this should make virtually no difference.

since the truncated tail is more than three standard deviations from the mean. However, for the case of .5774, the true coefficient of variation for the truncated normal is a bit less than .5774.

For number of machines m = 2, 3, 4, 5, and 10, all nine distributions were used, for a total of 45 experiments. In all cases, the number of jobs was set equal to 100.

A second set of experiments was performed using only the uniform distribution on the interval (5,10) in order to see how problem size, as measured by number of jobs, affected the results. There were thirty experiments in this second set, corresponding to m = 2, 3, 4, 5, and 10 and number of jobs n = 25, 50, 75, 100, 125, and 150.

The comparison of theoretical values of mean and standard deviation with the sample means and sample standard deviations for the processing times is shown in the following table:

Uniform (a, b)								
U (0,1) u2d			U (19,20) u1d			U (5,10) u3d		
mean	stdev	CV	mean	stdev	CV	mean	stdev	CV
0.5	0.2887	0.5774	19.5	0.2887	0.0148	7.5	1.4434	0.1925
0.4985	0.2879	0.5775	19.4985	0.2879	0.0148	7.4927	1.4394	0.1921
Normal (mean, var)								
n1d			n2d			n3d		
mean	stdev	CV	mean	stdev	CV	mean	stdev	CV
0.5	0.2887	< 0.5774	19.5	0.2887	0.0148	7.5	1.4434	0.1925
0.5262	0.2895	0.5502	19.4983	0.2895	0.0148	7.4917	1.4475	0.1932
Exponential (λ, shift)								
E (λ=1.3643, shift=0.2113) e1d			E (λ=1.3643,shift=19.2111) e2d			E (λ=0.9124, shift=6.0566) e3d		
mean	stdev	CV	mean	stdev	CV	mean	stdev	CV
0.5	0.2887	0.5774	19.5	0.2887	0.0148	7.5	1.4434	0.1925
0.4988	0.2866	0.5746	19.4986	0.2866	0.0147	7.4942	1.4329	0.1912

Note: stdev : standard deviation.

The mean and stdev in the clear area are the theoretical ones calculated from the distribution parameters.

The values in the shadow area are the sample mean and sample stdev.

Table 3.1 Mean, Standard Deviation and CV of Experiments

3.2 Simulation Results

3.2.1 Behaviour of the Algorithms for Constant Number of Jobs

Let the sum of all of the processing times for a problem instance be denoted by b. A near optimal makespan algorithm will assign to each machine approximately (1/m)-th of the total time b. Therefore we would expect that the graph of makespan versus ($1/m$) will be approximately linear for a good algorithm. Graphs B.1 to B.3 (see Appendix B) reveal this linearity for LPTa and for LPTo, but not for P(m). This information in confirmed by the information in Table A.3 (see Appendix A), where it is clear that LPTa and LPTo generally produce near optimal solutions, while P(m) tends to produce results which are greater than optimal (M/OPT > 1).

To get some idea of what one might intuitively expect, we compute the makespan for problems with n=100 jobs when all processing times are equal to 1 (this could be done in automated fashion by use of the uniform distribution with lower and upper end points equal to 1). If we do this, we find that the estimated makespans under the various algorithms are as follows, for n=100 jobs:

# machines, m	2	3	4	5	10
LTPa	50	34	25	20	10
LPTo	50	34	25	20	10
P(m)	66	40	35	25	13
% by which P(m) exceeds LPTa	32% *	17.6%	40%	25%	30%

* i.e., 32% = (66-50)/50.

Table 3.2 Makespan with n=100 and $a_i = 1$

To see that this applies to our simulations, consider Table A.1, column U(19,20). Roughly speaking, we may assume that all processing times are approximately 19.5 (this is reasonable, given the low coefficient of variation). Thus, we may multiply the times computed above by 19.5 to "verify" the operations of the algorithms. The agreement of Table A.1 with the above estimates is excellent.

3.2.1.1 Average Performance Ratios : Makespan

Table A.1 reveals excellent performance of the LPTa and LPTo algorithms, with average ratio of heuristic value to lower bound (M/LB) below 1.01 in most cases. Ten of the nineteen cases which yielded ratios greater than 1.01 occurred under the shifted exponential distribution, confirming our prior expectations, which were based upon the idea that the shifted exponential would "upset the apple-cart" by producing one or two very large times. Recall that the exponential distribution has a long tail to the right, so with low probability will generate values far from the mean. In this regard, over half the times in which one of LPTa or LPTo exceeded 1.01, so did the other. Of course, the excellent ratios for LPTa and LPTo actually overestimate the worst case performance ratio, since the denominator is a lower bound which may be less than the optimal value in each case. These observations on Table A.1 are also reflected in the 95% confidence intervals shown in Tables A.3 and A.5 (see Appendix A).

There is a suspicion based upon the data in Table A.1 that problems with more machines may produce higher ratios than are seen up to m=10. In particular, note the apparent growth in the ratio for the LPTo algorithm under the distribution E(1.3643, 0.2113), which ratio is close to 1.09 for m=10. While no broad conclusions can be made, further study on this point would be worthwhile. And once again, the shifted exponential was the "guilty party", i.e., performance using a skewed distribution was worse than with a symmetrical one.

The worse and average case performances of the algorithms P(m) illustrate what is acknowledged by researchers to be typical: guarantees of good worst case performance often entail a severe sacrifice of average performance. The predicted performance based upon all times equal to 1 were closely approximated, as expected (see Table 3.2).

One idea clearly emerges from the above observations: LPTa and LPTo are almost indistinguishable with respect to worst case performance ratio for makespan for the data configurations which were examined.

It is also worth noting that the greatest deterioration in algorithm performance for the LPT based algorithms occurred under the shifted exponential with the highest coefficient of variation. It is conjectured that the asymmetry and the high coefficient of variation are the causes of the deterioration, but to confirm this would entail further investigation.

3.2.1.2 Average Performance Ratio: Mean Completion Time

We now consider average completion time. Recall that for average completion time, LPTo gives optimal results. Thus, in Tables A.2 and A.4(see Appendix A), we examine the ratios using the LPTo value as denominator, and these ratios are truly the worst case performance ratio. Table A.2 reveals unexpected results. First, for our data configurations, there is a remarkable agreement between LPTa and LPTo with respect to average completion time. In fact, the only differences at all occur for the case of the highest coefficient of variation (.5774) which was examined, and the differences in these cases were tiny. While we expected close agreement between results using the two algorithms, we did not anticipate the nearly identical performance. A little thought based upon the following property reveals the reason for these surprising results:

Average completion time is minimized for the m machine problem if and only if each successive group of m jobs is assigned to different machines ([CMM], P77).

For example, for a 7 job, three machine problem, jobs 1 to 3 must each be on a different machine, jobs 4 to 6 must each be on a different machine, and job 7 may go anywhere. Consider how this observation applies to the case of U(19,20). In the LPTa algorithm, after the assignment of 3k jobs the amount of time assigned to each machine will be 19k + (a relatively small amount). Moreover, the relatively small amounts should be about equal. Therefore, the next three jobs will be split between the three machines, and an inductive argument suggests that for "not badly behaved" data LPTa will optimize average completion time. Thus, for data drawn from a "smooth" distribution LPTa can be expected to produce near optimal values of average completion time.

P(m) performs much better with respect to average performance than with respect to worst case performance. Table A.2 shows that the mean value of C(LPTa)/C(LPTo) is almost always 1.00, while the average value of C(P(m))/C(LPTo) takes on the following values as a function of m (the values shown below are the mean of the nine means in the appropriate row of Table A.2; for example, for m=4, 1.084 = 1.08387 + 1.08385 + ... + 1.08330):

Number of Machines:	2	3	4	5	10
Mean of 9 Means:	1.101	1.075	1.084	1.076	1.077

Thus, P(m) is not competitive with either of the other algorithms for the mean completion time objective.

To summarize, the performance of LPTa and of LPTo is of almost the same quality for both the makespan and average completion time objectives, while P(m) is on average considerably worse for both objectives.

We have largely ignored the information on confidence intervals (which of course gives hypothesis test results as a by-product) in Tables A.3 to A.5 (see Appendix A). The reason is that the very small standard deviations shown in Tables A.1 and A.2 (see Appendix A) have the effect of making the mean observed values into virtual constants, and there is really no additional insight obtained by additional discussion of the confidence intervals. Perhaps experiments with higher CV's might show wider confidence intervals: however, as discussed in the conclusions, a CV of 0.5774 is certainly not small for empirical data. Tables A.6 and A.7 (see Appendix A) give the observed mean values and standard deviations for makespan and for average completion time from the various experiments, and are included for completeness. Finally, the data in Table A.9 simply confirms previous conclusions with respect to the mean completion time objective.

3.2.2 Behaviour of the Algorithms for Different Numbers of Jobs

The next experiments examined the performance of the various algorithms for different problem sizes, using only the distribution U(5,10).

3.2.2.1 Makespan as a Function of n:

For smooth distributions with a not too large coefficient of variation, the lower bound $\max\{ a_1, a_m+a_{m+1}, b/m \}$ will equal b/m for n of reasonable size, for example, $n>2m$. For small values of n, b/m is likely to be a significant underestimate of the true optimal value. For example, suppose $m=10$ and $n=21$. If all times were equal to 1, the lower bound would equal

2.1, while the optimal value would be 3. More generally, assuming all times are equal to 1, for $n>2m$ we have optimal value equal to $\lceil n/m \rceil$ while $b/m = n/m$, and hence (optimal value)/(lower bound) = $\lceil n/m \rceil/(n/m)$, which is relatively large for n near to 2m, but converges to 1 for n large.

Intuitively, the lower bound b/m "assigns" to each machine n/m jobs, and unless n/m is integer, an underestimate will tend to occur because some machines will have assigned to them at least $\lceil n/m \rceil$ jobs. Thus for n=25 and m=10, the ratios in Tables A.8 and A.10 (see Appendix A) may be assumed to be significant overestimates of the true worst case performance ratios. There is some effect for the other n=25 cases, but for $n \geq 50$ much smaller effects. For the case of m=5, the above effects will not occur at all since 5 divides all of the values of n which were used. It is recommended that the work reported here be modified in the future by use of the more general bounding procedures, such as those described in [VD].

Next we compare the performance of LPTa and LPTo. For m=2 to 5 and n>25, there are very few cases in which the ratios shown in Table A.8 exceed 1.01 (see Graphs B.4 and Graph B.5 in Appendix B as well). For n=25, it is likely that the ratios shown represent a significant overestimate of the worst case performance ratio, as suggested in the previous paragraph, with the overestimation increasing with the number of machines. Notice that for the case of m=5, the performance ratios are all less than 1.01, with the exception of the LPTo algorithm with n=25. These uniformly low values reflect the fact that was mentioned above, namely that m divides n for all of the n used.

Thus, to evaluate the effectiveness of the algorithms LPTa and LPTo under different problem sizes, it is useful to focus on the pairs (m, n) for which m divides n, as follows:

m= 2, n= 50, 100, 150

m= 3, n= 75, 150

m= 4, n= 100

m= 5, n= 25, 50, 75, 100, 125, 150

m=10 n= 50, 100, 150

For these cases, there are only two cases of ratios greater than 1.01, and these are for LPTo with m=5 and n=25, and LPTo with m=10 and n=50. Even in these cases the ratios were only about 1.015. The conclusion is that LPTa and LPTo give very good values of worst case performance ratio, approaching 1 as n increases. This confirms well documented information about LPTa, but previously unknown information about LPTo.

For the P(m) algorithms, the performance ratios approximate what would be expected by setting all times equal to 1. As n gets large, the values predicted by setting all times equal to 1 are:

m= 2: 1.33

m= 3: 1.20

m= 4: 1.43

m= 5: 1.25

m=10: 1.33

The results for P(m) are illustrated in Graph B.6.

Hence, we can conclude that P(m) will give makespans whose performance ratios approach their predicted values, which are in all cases much greater than 1, and much greater than the values achieved by the algorithms LPTa and LPTo.

3.2.2.2 Average Completion Time as a Function of n:

As previously discussed, LPTa for the smooth distributions under consideration gives the same average completion times as LPTo.

For all values of m the ratios observed for P(m) approximate closely those which would be predicted by setting all times equal to 1.

To conclude, LPTa and LPTo both perform optimally for the problems under consideration, although theoretically only LPTo is guaranteed to do so in all circumstances.

As previously, there is nothing significant to learn from the confidence interval data of Tables A.10 to A.12 (see Appendix A).

To summarize the empirical results reported in this chapter, the LPT algorithm using either actual or ordinal data gives extremely good worst case performance over data drawn from a variety of smooth distributions and for varying problem sizes, for both the makespan and the average completion objectives. Simply because heuristics often produce good results, it was expected in advance of this study that the ordinal LPT algorithm would give good results for makespan, and that the LPT algorithm using actual data would give good results for average completion time. However, it was never anticipated just how surprisingly good these results would be.

From a practical point of view, ordinal data is as good as actual data for minimizing makespan and average completion time with the LPT algorithm for smooth data distributions. On the other hand, unless very unusual data configurations are expected, the special algorithms P(m)

which guarantee good worst case performance for makespan should be bypassed for both objective functions.

Chapter 4: Conclusions and Further Research Directions

This research was motivated by the observation that many real-world optimization problems have to be solved with very incomplete knowledge of the data. Often one is forced to rely on estimates, "guestimates" or even worse. One possible model for such data for certain kinds of problems is the ordinal data model.

For the range of data examined, the P algorithms are, in a practical sense, useless. For the makespan problem, they guarantee theoretical worst case performance to be minimal ($m=2,3$), close to minimal ($m=4$), and reasonable ($m>4$), for smooth data with $CV<.58$ but they consistently give worst makespan considerably higher than LPTo. Similarly, they give significantly worse average completion time than LPTo.

While larger CV values should be explored, note that the CV's examined are representative of large classes of real data. For example, to match the CV of .577, an exam with an mean mark of 70% would need to have a standard deviation of 40%.

While it was anticipated that the LPTa and LPTo algorithms would yield similar results, the actual closeness of the performance of the two algorithms under varying problem sizes (measured by m and n) and different smooth distributions was quite a surprise. In retrospect, it is possible to get some intuitive idea of what seems to be going on by looking at problems with all times equal to 1, and this technique can be used in planning new experiments. The excellent results of the LPTo algorithm compared with the LPTa algorithm suggest further exploration of even weaker forms of data as the basis for parallel machine scheduling. For example, ordinal data implies prior knowledge of a total ordering of the jobs by processing time. What forms of partial orders would be sufficient to yield good results from "partial-ordinal algorithms"?

The algorithms which guarantee low worst case performance for makespan are really of no practical use for the kind of smooth data distributions which were studied, and it is likely that for most practical problems these algorithms are of little value.

There are several areas for further empirical research. First, the results reported for makespan could be sharpened by implementing more effective lower bounding procedures. Second, the development of algorithms using partial orders. Third, further experiments should be performed using distributions with much higher coefficients of variation. Third, it would be interesting to generalized this work from the parallel identical processor case to the parallel *uniform* processor case, in which the processing time for job i on machine j is given by $h_j a_i$, where h_j is the speed factor for machine j. For the parallel uniform processor case, the assumption would be that the h_j's are known, but the processing time information is only ordinal. Finally, a hybrid version of the parallel uniform processor problem, in which some the a_i's are known precisely, and in addition the ordinal sequence for all a_i's is known, would be worth investigating.

Bibliography

[A] Agnetis. Allessandro(1989). "*No-Wait Flow Shop Scheduling with Large Lot Size*". Rap. 16.89. Dipartimento di Informatica e Sistemistica. Universita Degli Studi di Roma "La Sapienza". Rome. Italy.

[C] Chen. B. (1993). "*A note on LPT scheduling*". Opns. Res. Letter 14. 139-142.

[CMM] Conway. W.. W.L. Maxwell and L.W. Miller(1967). *Theory of Scheduling*. Addison-Wesley. Reading. Mass.

[CS] Coffman. E.G. and Sethi. R. (1976). " *A generalized bound on LPT sequencing* ". Revue Française d'Automatique Informatique. Recherche Operationnelle Supplément au Vol. 10. 17-25(1976).

[G] Graham. R.L. (1966). "*Bounds for certain multiprocessing anomalies*." Bell Syst. Tech. J. 45. 1563-1581.

[GJ] Garey. M.R. and D.S. Johnson (1979). *Computers and Intractability - A Guide to NP-Completeness*. W.H. Freeman and Company. San Francisco.

[JD] John Dagpunar. "*Principles of Random Variable Generation*". Oxford Science Publications. 1988.

[L1] Lawler. E.L.(1976). *Combinatorial Optimization: Networks and Matroids*. Holt. Rinehart and Winston. Toronto.

[L2] Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys(1989). "*Sequencing and Scheduling: Algorithms and Complexity*", Centre for Mathematics and Computer Science Report BS-R8909, Stiching MathematischCentrum, Amsterdam.

[LS] Liu, W.P., and Sidney, J.B.(1993). "*Ordinal Algorithms for Parallel Machine Scheduling*", Faculty of Administration, University of Ottawa, Ottawa, Canada.

[VD] Vizvári, Béla and Demir Ramazan [1992]. "It is difficult to find a difficult problem for the scheduling of identical parallel machines.", Dept. of Industrial Engineering, Bilkent University, Ankara, Turkey.

[WH] Wichmann, Brian, and David Hill, *"Building a Random Number Generator"*, March 1987, BYTE Vol. 10, No. 3, 127-128.

[WH] Wichmann, B. A. & Hill , I. D. (1982) *Algorithm AS183, An efficient and portable pseudo-random number generator.* Applied Statistics, 31, 188-190.

[WH] Wichmann, B. A. & Hill , I. D. (1984) *Correction to algorithm AS183, An efficient and portable pseudo-random number generator.* Applied Statistics, 33, 123.

Appendices

Appendix A : Tabular Results

number of machine	algorithm x	Uniform (a, b)						Normal (mean, var)						Exponential (λ, shift)					
		U (0, 1)		U (19, 20)		U (5, 10)		N (0.5, 0.08333)		N (19.5, 0.08333)		N (7.5, 2.08333)		E (1.3643, 0.2113)		E (1.3643, 19.2111)		E(0.9124, 6.0566)	
		mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev
2	LPTa	1.00016	0.00016	1.00001	0.00000	1.00007	0.00006	1.00038	0.00039	1.00004	0.00004	1.00054	0.00052	1.00045	0.00113	1.00002	0.00005	1.00022	0.00067
	LPTo	1.00108	0.00088	1.00003	0.00002	1.00036	0.00029	1.00211	0.00180	1.00007	0.00006	1.00092	0.00077	1.00135	0.00446	1.00011	0.00012	1.00146	0.00152
	P(2)	1.32014	0.00118	1.32000	0.00003	1.32005	0.00040	1.31642	0.00302	1.31999	0.00011	1.31986	0.00138	1.30629	0.00691	1.31965	0.00018	1.31543	0.00237
3	LPTa	1.00041	0.00037	1.01949	0.00004	1.01331	0.00037	1.00100	0.00088	1.01921	0.00014	1.00966	0.00180	1.00783	0.00158	1.01968	0.00008	1.01583	0.00097
	LPTo	1.00210	0.00131	1.01948	0.00007	1.01329	0.00087	1.00494	0.00403	1.01912	0.00018	1.00858	0.00235	1.00947	0.00776	1.01952	0.00018	1.01381	0.00229
	P(3)	1.19942	0.00167	1.19999	0.00004	1.19981	0.00056	1.19481	0.00428	1.19991	0.00013	1.19876	0.00167	1.18823	0.00712	1.19970	0.00019	1.19608	0.00241
4	LPTa	1.00076	0.00050	1.00002	0.00001	1.00029	0.00017	1.00160	0.00118	1.00013	0.00009	1.00174	0.00113	1.00238	0.00433	1.00010	0.00024	1.00135	0.00304
	LPTo	1.00353	0.00151	1.00009	0.00004	1.00117	0.00050	1.00958	0.00676	1.00032	0.00017	1.00419	0.00218	1.01845	0.01421	1.00047	0.00038	1.00616	0.00182
	P(4)	1.38719	0.00327	1.39968	0.00007	1.39577	0.00098	1.38086	0.00565	1.39961	0.00018	1.39485	0.00234	1.36161	0.01042	1.39901	0.00029	1.38720	0.00364
5	LPTa	1.00131	0.00086	1.00004	0.00002	1.00048	0.00031	1.00239	0.00139	1.00019	0.00011	1.00251	0.00146	1.00130	0.00633	1.00019	0.00037	1.00244	0.00477
	LPTo	1.00486	0.00236	1.00012	0.00006	1.00162	0.00079	1.01148	0.00763	1.00029	0.00017	1.00382	0.00223	1.02851	0.01983	1.00073	0.00059	1.00951	0.00675
	P(m)	1.24795	0.00325	1.24995	0.00008	1.24932	0.00108	1.23953	0.00666	1.24971	0.00170	1.24621	0.00221	1.22500	0.01174	1.24936	0.00031	1.24168	0.00396
10	LPTa	1.00524	0.00283	1.00014	0.00008	1.00179	0.00097	1.00811	0.00339	1.00048	0.00021	1.00628	0.00278	1.02217	0.01365	1.00113	0.00116	1.01472	0.01492
	LPTo	1.01165	0.00564	1.00030	0.00014	1.00387	0.00184	1.03058	0.01988	1.00065	0.00039	1.00850	0.00507	1.08327	0.04568	1.00226	0.00121	1.02941	0.01551
	P(m)	1.31063	0.00677	1.30027	0.00018	1.30356	0.00229	1.28923	0.01167	1.29994	0.00031	1.29916	0.00407	1.24865	0.01908	1.29869	0.00050	1.28292	0.00642

Note : M(x) represents the makespan of the given algorithm x
LB represents the lower bound of makespan problem.
The shaded boxs are the cases where the ratio of M(x)/LB for LPTo exceed 1.01

Table A.1 : Mean and Standard Deviation of the Ratio of M(x) / LB

Appendix A : Tabular Results

number of machine	algorithm x	Uniform (a, b)						Normal (mean, var)						Exponential (λ, shift)					
		U (0, 1)		U (19, 20)		U (5, 10)		N (0.5, 0.08333)		N (19.5, 0.08333)		N (7.5, 2.08333)		E (1.3643, 0.2113)		E (1.3643, 19.211)		E(0.9124, 6.0566)	
		mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev
2	LPTa	1.00000	0.00001	1.00000	0.00000	1.00000	0.00000	1.00000	0.00001	1.00000	0.00000	1.00000	0.00000	1.00002	0.00007	1.00000	0.00000	1.00000	0.00000
	P(2)	1.10152	0.00061	1.10041	0.00001	1.10068	0.00016	1.10203	0.00078	1.10048	0.00004	1.10161	0.00058	1.09969	0.00031	1.10038	0.00001	1.10021	0.00008
3	LPTa	1.00000	0.00002	1.00000	0.00000	1.00000	0.00000	1.00000	0.00002	1.00000	0.00000	1.00000	0.00000	1.00003	0.00013	1.00000	0.00000	1.00000	0.00000
	P(3)	1.06984	0.00057	1.07733	0.00001	1.07554	0.00016	1.07084	0.00067	1.07728	0.00003	1.07486	0.00039	1.07307	0.00042	1.07738	0.00001	1.07630	0.00014
4	LPTa	1.00001	0.00002	1.00000	0.00000	1.00000	0.00000	1.00001	0.00003	1.00000	0.00000	1.00000	0.00000	1.00008	0.00024	1.00000	0.00000	1.00000	0.00000
	P(4)	1.08387	0.00097	1.08385	0.00002	1.08386	0.00025	1.08494	0.00094	1.08393	0.00004	1.08501	0.00062	1.08180	0.00061	1.08381	0.00001	1.08330	0.00016
5	LPTa	1.00001	0.00003	1.00000	0.00000	1.00000	0.00000	1.00001	0.00004	1.00000	0.00000	1.00000	0.00000	1.00016	0.00043	1.00000	0.00000	1.00000	0.00000
	P(m)	1.07424	0.00107	1.07803	0.00002	1.07712	0.00024	1.07526	0.00108	1.07803	0.00003	1.07718	0.00047	1.07436	0.00059	1.07803	0.00001	1.07710	0.00017
10	LPTa	1.00003	0.00008	1.00000	0.00000	1.00000	0.00000	1.00005	0.00018	1.00000	0.00000	1.00000	0.00000	1.00119	0.00167	1.00000	0.00000	1.00000	0.00000
	P(m)	1.06844	0.00197	1.08158	0.00003	1.07836	0.00047	1.07024	0.00203	1.08150	0.00006	1.07722	0.00089	1.07234	0.00113	1.08164	0.00003	1.07925	0.00035

Note : C(x) represents the average completion time of a given algorithm x.

Table A.2 : Mean and Standard Deviation of the Ratio for C(x)/C(LPTo)

Appendix A : Tabular Results

number of machine	algorithm x	Uniform (a b)						Normal (mean, var)						Exponential (λ shift)					
		U (0 1)		U (19 20)		U (5 10)		N (0.5 0.08333)		N (19.5 0.08333)		N (7.5 2.08333)		E (1.3643, 0.2113)		E (1.3643, 19.211)		E(0.9124, 6.0566)	
		95% CI		95% CI		95% CI		95% CI		95% CI		95% CI		95% CI		95% CI		95% CI	
2	LPTa	1.00013	1.00019	1.00000	1.00001	1.00005	1.00008	1.00030	1.00046	1.00003	1.00005	1.00044	1.00064	1.00023	1.00067	1.00001	1.00003	1.00008	1.00035
	LPTo	1.00090	1.00125	1.00002	1.00003	1.00030	1.00042	1.00166	1.00237	1.00006	1.00008	1.00077	1.00108	1.00347	1.00524	1.00009	1.00014	1.00115	1.00176
	P(2)	1.31991	1.32037	1.32000	1.32001	1.31998	1.32013	1.31582	1.31701	1.31997	1.32001	1.31959	1.32014	1.30492	1.30766	1.31961	1.31968	1.31496	1.31590
3	LPTa	1.00034	1.00048	1.01948	1.01949	1.01324	1.01339	1.00083	1.00117	1.01918	1.01923	1.00930	1.01002	1.00751	1.00814	1.01966	1.01969	1.01564	1.01603
	LPTo	1.00184	1.00236	1.01947	1.01950	1.01312	1.01346	1.00414	1.00574	1.01909	1.01916	1.00811	1.00904	1.00793	1.01101	1.01949	1.01956	1.01336	1.01427
	P(3)	1.19909	1.19975	1.19998	1.19999	1.19970	1.19992	1.19396	1.19566	1.19928	1.19993	1.19843	1.19909	1.18682	1.18964	1.19966	1.19973	1.19560	1.19655
4	LPTa	1.00066	1.00085	1.00002	1.00002	1.00025	1.00032	1.00137	1.00183	1.00012	1.00015	1.00152	1.00196	1.00153	1.00324	1.00006	1.00015	1.00074	1.00195
	LPTo	1.00323	1.00383	1.00008	1.00010	1.00107	1.00127	1.00823	1.01092	1.00029	1.00035	1.00376	1.00462	1.01563	1.02126	1.00040	1.00055	1.00520	1.00712
	P(4)	1.38654	1.38784	1.39966	1.39969	1.39558	1.39597	1.37974	1.38199	1.39957	1.39964	1.39438	1.39531	1.35954	1.36368	1.39896	1.39907	1.38648	1.38793
5	LPTa	1.00114	1.00148	1.00003	1.00004	1.00041	1.00054	1.00211	1.00266	1.00017	1.00022	1.00222	1.00280	1.00304	1.00556	1.00011	1.00026	1.00149	1.00338
	LPTo	1.00439	1.00533	1.00011	1.00014	1.00146	1.00177	1.00996	1.01299	1.00026	1.00033	1.00338	1.00427	1.02457	1.03244	1.00063	1.00084	1.00817	1.01085
	P(m)	1.24730	1.24859	1.24993	1.24996	1.24911	1.24954	1.23821	1.24085	1.24968	1.24974	1.24577	1.24665	1.22267	1.22733	1.24930	1.24942	1.24089	1.24247
10	LPTa	1.00468	1.00580	1.00012	1.00015	1.00159	1.00198	1.00744	1.00878	1.00044	1.00052	1.00573	1.00683	1.01947	1.02488	1.00090	1.00136	1.01176	1.01768
	LPTo	1.01054	1.01277	1.00027	1.00032	1.00350	1.00423	1.02663	1.03452	1.00058	1.00073	1.00750	1.00951	1.07920	1.09733	1.00202	1.00250	1.02633	1.03248
	P(m)	1.30928	1.31197	1.30024	1.30031	1.30310	1.30401	1.28692	1.29155	1.29987	1.30000	1.29835	1.29997	1.24487	1.25244	1.29859	1.29878	1.28164	1.28419

Note : M(x) represents the makespan of a given algorithm x
LB represents the lower bound of makespan problem.

Table A.3 : 95% CI of the Ratio of M(x) / LB

Appendix A : Tabular Results

number of machine	algorithm x	Uniform (a, b)						Normal (mean, var)						Exponential (λ, shift)					
		U (0, 1)		U (19, 20)		U (5, 10)		N (0.5, 0.08333)		N (19.5, 0.08333)		N (7.5, 2.08333)		E (1.3643, 0.2113)		E (1.3643, 19.211)		E(0.9124, 6.0566)	
		95% CI		95% CI		95% CI		95% CI		95% CI		95% CI		95% CI		95% CI		95% CI	
2	LPTa	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00003	1.00000	1.00000	1.00000	1.00000
	P(2)	1.10140	1.10164	1.10041	1.10041	1.10065	1.10071	1.10188	1.10219	1.10047	1.10048	1.10149	1.10172	1.09963	1.09975	1.10038	1.10038	1.10019	1.10022
3	LPTa	1.00000	1.00001	1.00000	1.00000	1.00000	1.00000	1.00000	1.00001	1.00000	1.00000	1.00000	1.00000	1.00000	1.00006	1.00000	1.00000	1.00000	1.00000
	P(3)	1.06971	1.06998	1.07733	1.07733	1.07551	1.07557	1.07070	1.07097	1.07728	1.07729	1.07479	[illegible]07494	1.07299	1.07315	1.07738	1.07738	1.07627	1.07632
4	LPTa	1.00000	1.00001	1.00000	1.00000	1.00000	1.00000	1.00000	1.00002	1.00000	1.00000	1.00[illegible]00	[illegible]0000	1.00003	1.00013	1.00000	1.00000	1.00000	1.00000
	P(4)	1.08368	1.08406	1.08384	1.08385	1.08381	1.08391	1.08476	1.08513	1.08392	1.08394	1.0[illegible]489	1.[illegible]8514	1.08168	1.08192	1.08381	1.08381	1.08327	1.08334
5	LPTa	1.00000	1.00002	1.00000	1.00000	1.00000	1.00000	1.00000	1.00002	1.00000	1.00000	1.00000	1.00000	1.00008	1.00025	1.00000	1.00000	1.00000	1.00000
	P(m)	1.07402	1.07445	1.07802	1.07803	1.07708	1.07717	1.07505	1.07547	1.07803	1.07804	1.07709	1.07728	1.07424	1.07448	1.07802	1.07803	1.07707	1.07713
10	LPTa	1.00002	1.00005	1.00000	1.00000	1.00000	1.00000	1.00001	1.00008	1.00000	1.00000	1.00000	1.00000	1.00086	1.00152	1.00000	1.00000	1.00000	1.00000
	P(m)	1.06805	1.06883	1.08157	1.08158	1.07827	1.07846	1.06984	1.07064	1.08148	1.08151	1.07704	1.07739	1.07212	1.07257	1.08163	1.08164	1.07918	1.07932

Note : C(x) represents the average completion time of a given algorithm x.

Table A.4 : 95% CI of the Ratio of C(x) / C(LPTo)

Appendix A : Tabular Results

number of machines	Uniform (a, b)						Normal (mean, var)						Exponential (λ, shift)					
	U (0, 1)		U (19, 20)		U (5, 10)		N (0.5, 0.08333)		N (19.5, 0.08333)		N (7.5, 2.08333)		E(1.3643,0.2113)		E(1.3643,19.2111)		E(0.9124, 6.0566)	
	95% CI		95% CI		95% CI		95% CI		95% CI		95% CI		95% CI		95% CI		95% CI	
2	0.99891	0.99927	0.99997	0.99998	0.99965	0.99977	0.99800	0.99874	0.99996	0.99998	0.99946	0.99978	0.99542	0.99684	0.99989	0.99992	0.99855	0.99897
3	0.99806	0.99858	0.99999	1.00001	0.99986	1.00019	0.99529	0.99689	1.00006	1.0001	1.00078	1.00138	0.99670	1.00018	1.00013	1.00018	1.00163	1.00237
4	0.99692	0.99756	0.99992	0.99994	0.99901	0.99922	0.99084	0.99344	0.99978	0.99984	0.99717	0.99796	0.98237	0.98636	0.99959	0.99967	0.99473	0.99572
5	0.99598	0.99695	0.99990	0.99993	0.99870	0.99903	0.98956	0.99258	0.99986	0.99994	0.99816	0.99923	0.97398	0.97945	0.99941	0.99950	0.99239	0.99362
10	0.99253	0.99484	0.99981	0.99987	0.99756	0.99831	0.97472	0.98241	0.99974	0.99992	0.99662	0.99902	0.93439	0.94661	0.99882	0.99893	0.98509	0.98641

Note : M(LPTa) represents the makespan of algorithm LPTa
M(LPTo) represents the makespan of algorithm LPTo

Table A.5 : 95% CI of the Ratio of M(LPTa)/M(LPTo)

# of machine	algorithm x	Uniform (a, b)						Normal (mean, var)						Exponential (λ, shift)					
		U (0, 1)		U (19, 20)		U (5, 10)		N (0.5, 0.08333)		N (19.5, 0.08333)		N (7.5, 2.08333)		E (1.3643, 0.2113)		E (1.3643, 19.211)		E(0.9124, 6.0566)	
		mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev
2	LPTa	24.9310	1.4880	974.932	1.488	374.661	7.440	26.3180	1.4760	974.957	1.715	374.786	8.575	24.9530	1.5190	974.949	1.529	374.791	7.629
	LPTo	24.9540	1.4880	974.954	1.488	374.769	7.442	26.3610	1.4780	974.986	1.712	374.929	8.561	25.0510	1.5440	975.012	1.544	375.257	7.720
	P(2)	32.9080	1.9790	1286.91	1.980	494.541	9.895	34.6340	1.9590	1286.88	2.270	494.400	11.33	32.5800	1.9590	1286.57	1.960	492.898	9.794
3	LPTa	16.6248	0.9915	662.616	0.990	253.080	4.950	17.5567	0.9870	662.427	1.150	252.135	5.751	16.7570	1.0090	662.745	1.008	253.759	5.031
	LPTo	16.6527	0.9909	662.615	0.994	253.075	4.968	17.6258	0.9930	662.373	1.167	251.865	5.835	16.7860	1.0380	662.643	1.010	253.254	5.058
	P(3)	19.9320	1.1880	779.932	1.188	299.660	5.942	20.9560	1.1780	779.872	1.386	299.359	6.929	19.7570	1.1930	779.749	1.193	298.785	5.967
4	LPTa	12.4730	0.7445	487.474	0.745	187.371	3.725	13.1754	0.7392	487.524	0.858	187.618	4.289	12.5009	0.7655	487.517	0.785	187.608	3.929
	LPTo	12.5073	0.7422	487.507	0.742	187.536	3.711	13.2805	0.7549	487.615	0.852	188.075	4.261	12.7024	0.8175	487.698	0.818	188.512	4.088
	P(4)	17.2910	1.0590	682.291	1.059	261.455	5.295	18.1650	1.0380	682.249	1.215	261.246	6.073	16.9790	1.0120	681.972	1.012	259.894	5.060
5	LPTa	9.98390	0.5963	389.985	0.597	149.926	2.984	10.5486	0.5931	390.042	0.685	150.209	3.423	10.0194	0.6096	390.046	0.642	150.251	3.211
	LPTo	10.0193	0.5975	390.019	0.597	150.096	2.987	10.6443	0.6042	390.081	0.681	150.405	3.404	10.2626	0.6791	390.259	0.679	151.313	3.395
	P(m)	12.4433	0.7469	487.443	0.747	187.216	3.735	13.0444	0.7408	487.354	0.866	186.725	4.328	12.2211	0.7467	487.216	0.747	186.106	3.733
10	LPTa	5.01180	0.3034	195.012	0.303	75.0610	1.516	5.30440	0.2983	195.077	0.344	75.3870	1.718	5.09820	0.3041	195.207	0.394	76.0470	1.970
	LPTo	5.04330	0.2993	195.043	0.299	75.2170	1.497	5.42220	0.3140	195.110	0.339	75.5520	1.696	5.42990	0.4198	195.428	0.420	77.1490	2.099
	P(m)	6.53450	0.4001	253.535	0.400	97.6730	2.000	6.78460	0.4031	253.466	0.449	97.3290	2.246	6.22860	0.3877	253.226	0.388	96.1430	1.938

Note : M(x) represents the makespan of a given algorithm x

Table A.6 : Mean and Standard Deviation of M(x)

Appendix A : Tabular Results

# of machine	algorithm x	Uniform (a, b)						Normal (mean, var)						Exponential (λ, shift)					
		U (0, 1)		U (19, 20)		U (5, 10)		N (0.5, 0.08333)		N (19.5, 0.08333)		N (7.5, 2.08333)		E (1.3643, 0.2113)		E (1.3643, 19.211)		E(0.9124, 6.0566)	
		mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev
2	LPTa	8.5993	0.7555	493.099	0.756	170.496	3.7780	9.72160	0.7267	493.168	0.902	170.842	4.5100	9.1595	0.44250	493.654	0.442	173.297	2.2120
	LPTo	8.5992	0.7555	493.099	0.756	170.496	3.7780	9.72160	0.7267	493.168	0.902	170.842	4.5100	9.1594	0.44240	493.654	0.442	173.297	2.2120
	P(m)	9.4724	0.8342	542.612	0.834	187.662	4.1710	10.7138	0.8045	542.720	0.996	188.201	4.9790	10.0725	0.48700	543.207	0.487	190.663	2.4350
3	LPTa	5.8173	0.5086	332.047	0.509	114.936	2.5430	6.57060	0.4891	332.093	0.607	115.166	3.0330	6.19240	0.29960	332.419	0.300	116.811	1.4980
	LPTo	5.8173	0.5085	332.047	0.509	114.936	2.5430	6.57060	0.4891	332.093	0.607	115.166	3.0330	6.19220	0.29960	332.419	0.300	116.811	1.4980
	P(m)	6.2237	0.5459	357.724	0.546	123.619	2.7300	7.03620	0.5258	357.758	0.655	123.788	3.2740	6.64460	0.32020	358.141	0.320	125.723	1.6010
4	LPTa	4.4268	0.3850	251.427	0.385	87.1340	1.9250	4.99560	0.3703	251.463	0.459	87.3130	2.2950	4.70860	0.22820	251.706	0.228	88.5410	1.1400
	LPTo	4.4267	0.3850	251.427	0.385	87.1340	1.9250	4.99550	0.3703	251.463	0.459	87.3130	2.2950	4.70820	0.22810	251.706	0.228	88.5410	1.1400
	P(m)	4.7982	0.4197	272.508	0.420	94.4410	2.0990	5.41990	0.4030	272.567	0.498	94.7360	2.4910	5.09340	0.24710	272.801	0.247	95.9170	1.2350
5	LPTa	3.5928	0.3109	203.093	0.311	70.4640	1.5540	4.05100	0.2991	203.122	0.370	70.6110	1.8520	3.81930	0.18540	203.317	0.185	71.5933	0.9261
	LPTo	3.5927	0.3109	203.093	0.311	70.4640	1.5540	4.05090	0.2991	203.122	0.370	70.6110	1.8520	3.81870	0.18520	203.317	0.185	71.5933	0.9261
	P(m)	3.8596	0.3361	218.94	0.336	75.8980	1.6800	4.35580	0.3223	218.972	0.399	76.0660	1.9940	4.10260	0.19880	219.18	0.199	77.1131	0.9940
10	LPTa	1.9273	0.1626	106.427	0.163	37.1360	0.8132	2.16450	0.1564	106.444	0.193	37.2222	0.9661	2.04504	0.09975	106.542	0.100	37.7131	0.4987
	LPTo	1.9272	0.1626	106.427	0.163	37.1360	0.8132	2.16440	0.1565	106.444	0.193	37.2222	0.9661	2.04261	0.09974	106.542	0.100	37.7131	0.4987
	P(m)	2.0592	0.1752	115.109	0.175	40.0461	0.8761	2.31660	0.1694	115.199	0.209	40.0960	1.0470	2.19030	0.10620	115.239	0.106	40.7017	0.5311

Note : C(x) represents the average completion time of a given algorithm x

Table A.7 : Mean and Standard Deviation of C(x)

# machines	# jobs x	25 mean	25 stdev	50 mean	50 stdev	75 mean	75 stdev	100 mean	100 stdev	125 mean	125 stdev	150 mean	150 stdev
2	LPTa	1.02681	0.00186	1.00026	0.00025	1.00889	0.00025	1.00007	0.00006	1.00535	0.00011	1.00003	0.00003
	LPTo	1.02830	0.00419	1.00113	0.00091	1.00893	0.00072	1.00036	0.00029	1.00539	0.00034	1.00019	0.00015
	P(2)	1.27972	0.00342	1.31542	0.00107	1.32749	0.00068	1.32005	0.0004	1.32616	0.00028	1.33036	0.00023
3	LPTa	1.05311	0.00376	1.01379	0.00062	1.00029	0.00021	1.01331	0.00037	1.00542	0.00013	1.00008	0.00006
	LPTo	1.05637	0.00620	1.01540	0.00186	1.00132	0.00079	1.01329	0.00087	1.00562	0.00037	1.00041	0.00027
	P(3)	1.19768	0.00531	1.19926	0.00156	1.19973	0.00099	1.19981	0.00056	1.19986	0.00038	1.19995	0.00030
4	LPTa	1.07897	0.00583	1.02724	0.00115	1.00935	0.00039	1.00029	0.00017	1.01596	0.00033	1.00895	0.00017
	LPTo	1.08406	0.01207	1.02958	0.00299	1.01059	0.00093	1.00117	0.00050	1.01595	0.00091	1.00916	0.00050
	P(4)	1.27506	0.00715	1.35521	0.00265	1.38179	0.00150	1.39577	0.00098	1.40358	0.00070	1.40893	0.00057
5	LPTa	1.00703	0.00414	1.00183	0.00105	1.00084	0.00048	1.00048	0.00031	1.00029	0.00017	1.00021	0.00012
	LPTo	1.01423	0.00734	1.00430	0.00189	1.00279	0.00133	1.00162	0.00079	1.00115	0.00060	1.00081	0.00040
	P(m)	1.20467	0.01084	1.21284	0.00371	1.25970	0.00175	1.24932	0.00108	1.24271	0.00076	1.26079	0.00060
10	LPTa	1.15015	0.01250	1.00714	0.00353	1.04622	0.00198	1.00179	0.00097	1.02738	0.00077	1.00077	0.00042
	LPTo	1.17683	0.02296	1.01493	0.00673	1.04642	0.00338	1.00387	0.00184	1.02916	0.00170	1.00226	0.00102
	P(m)	1.17064	0.01897	1.22139	0.00841	1.32266	0.00425	1.30356	0.00229	1.29164	0.00198	1.32908	0.00118

Note : M(x) represents the makespan of a given algorithm x
LB represens the lower bound of the makespan problem

Table A.8 : Mean and Standard Deviation of the Ratio of M(x)/LB, by # jobs

# machines	# jobs x	25		50		75		100		125		150	
		mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev	mean	stdev
2	LPTa	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
	P(2)	1.07226	0.00116	1.09689	0.00036	1.10565	0.00028	1.10068	0.00016	1.10518	0.00011	1.10839	0.00008
3	LPTa	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
	P(3)	1.06285	0.00091	1.07114	0.00037	1.07441	0.00020	1.07554	0.00016	1.07643	0.00009	1.07710	0.00008
4	LPTa	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
	P(4)	1.04332	0.00125	1.06594	0.00061	1.08526	0.00029	1.08386	0.00025	1.09214	0.00015	1.09250	0.00011
5	LPTa	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
	P(m)	1.04898	0.00129	1.05548	0.00067	1.06968	0.00035	1.07712	0.00024	1.08177	0.00018	1.08524	0.00012
10	LPTa	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000	1.00000	0.00000
	P(m)	1.00000	0.00000	1.03537	0.00101	1.08488	0.00071	1.07836	0.00047	1.07236	0.00048	1.09848	0.00029

Note : C(x) represents the average completion time of a given algorithm x

Table A.9 : Mean and Standard Deviation of the Ratio of C(x)/C(LPTo), by # jobs

#	# jobs	25		50		75		100		125		150	
machine	x	95% CI		95% CI		95% CI		95% CI		95% CI		95% CI	
	LPTa	1.02644	1.02717	1.00021	1.00031	1.00884	1.00894	1.00005	1.00008	1.00533	1.00537	1.00002	1.00003
2	LPTo	1.02746	1.02913	1.00095	1.00131	1.00879	1.00907	1.00030	1.00042	1.00533	1.00546	1.00016	1.00022
	P(2)	1.27904	1.28040	1.31521	1.31564	1.32735	1.32762	1.31998	1.32013	1.32611	1.32622	1.33031	1.33041
	LPTa	1.05236	1.05385	1.01367	1.01392	1.00025	1.00033	1.01324	1.01339	1.00540	1.00545	1.00006	1.00009
3	LPTo	1.05514	1.05760	1.01503	1.01577	1.00117	1.00148	1.01312	1.01346	1.00555	1.00569	1.00036	1.00047
	P(3)	1.19662	1.19873	1.19895	1.19957	1.19953	1.19993	1.19970	1.19992	1.19979	1.19994	1.19989	1.20001
	LPTa	1.07782	1.08013	1.02702	1.02747	1.00927	1.00943	1.00025	1.00032	1.01590	1.01603	1.00892	1.00898
4	LPTo	1.08166	1.08645	1.02898	1.03017	1.01041	1.01078	1.00107	1.00127	1.01577	1.01613	1.00906	1.00926
	P(4)	1.27364	1.27648	1.35468	1.35573	1.38149	1.38209	1.39558	1.39597	1.40344	1.40372	1.40882	1.40905
	LPTa	1.00621	1.00786	1.00163	1.00204	1.00075	1.00094	1.00041	1.00054	1.00026	1.00033	1.00019	1.00024
5	LPTo	1.01278	1.01569	1.00392	1.00467	1.00252	1.00305	1.00146	1.00177	1.00103	1.00127	1.00073	1.00089
	P(m)	1.20251	1.20682	1.21211	1.21358	1.25935	1.26004	1.24911	1.24954	1.24256	1.24286	1.26067	1.26091
	LPTa	1.14767	1.15263	1.00644	1.00784	1.04583	1.04661	1.00159	1.00198	1.02723	1.02753	1.00069	1.00085
10	LPTo	1.17227	1.18139	1.01360	1.01627	1.04575	1.04709	1.00350	1.00423	1.02882	1.02949	1.00206	1.00246
	P(m)	1.16688	1.17441	1.21972	1.22306	1.32181	1.32350	1.30310	1.30401	1.29125	1.29203	1.32885	1.32931

Note : M(x) represents the makespan of a given algorithm x
LB represents the lower bound of the makespan problem

Table A.10 : 95% CI of the Ratio of M(x)/LB, by # jobs

Appendix A : Tabular Results

# machine	# jobs x	25 95% CI		50 95% CI		75 95% CI		100 95% CI		125 95% CI		150 95% CI	
2	LPTa	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
	P(2)	1.07203	1.07249	1.09681	1.09696	1.10560	1.10571	1.10065	1.10071	1.10515	1.10520	1.10837	1.10840
3	LPTa	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
	P(3)	1.06267	1.06303	1.07106	1.07121	1.07437	1.07445	1.07551	1.07557	1.07642	1.07645	1.07708	1.07711
4	LPTa	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
	P(4)	1.04307	1.04356	1.06582	1.06606	1.08520	1.08531	1.08381	1.08391	1.09211	1.09217	1.09248	1.09252
5	LPTa	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
	P(m)	1.04872	1.04924	1.05534	1.05561	1.06961	1.06975	1.07708	1.07717	1.08173	1.08180	1.08522	1.08527
10	LPTa	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000	1.00000
	P(m)	1.00000	1.00000	1.03517	1.03557	1.08474	1.08502	1.07827	1.07846	1.07227	1.07246	1.09842	1.09835

Note : C(x) represents the average completion time of a given algorithm x

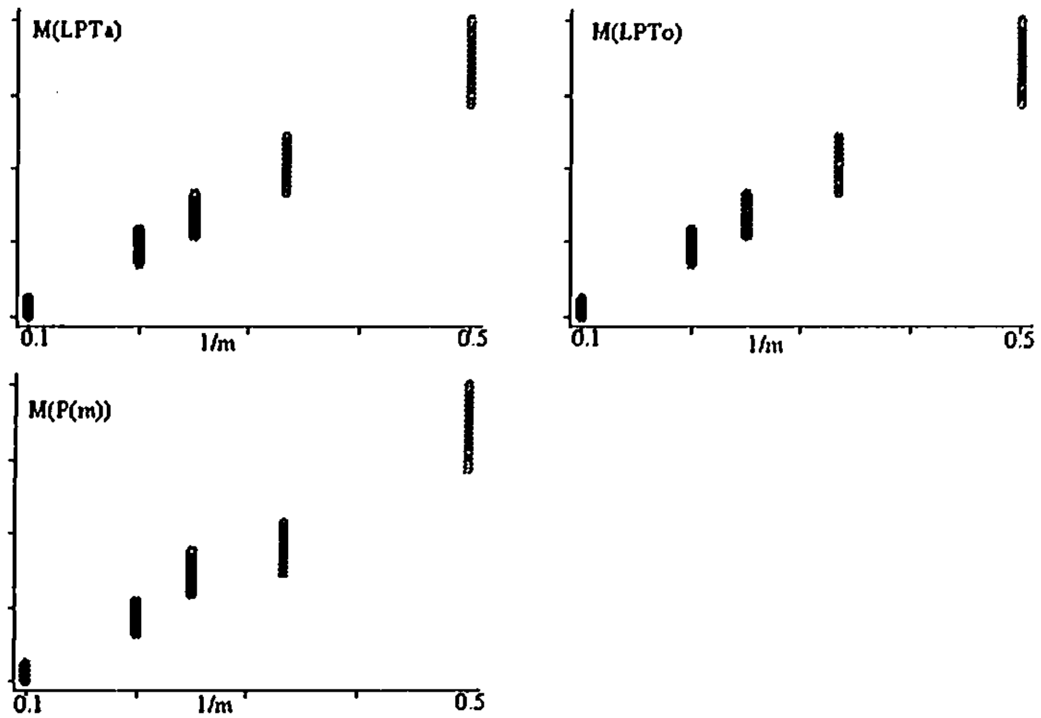
Table A.11 : 95% CI of the Ratio of C(x)/C(LPTo), by # jobs

#	25		50		75		100		125		150	
machines	95% CI		95% CI		95% CI		95% CI		95% CI		95% CI	
2	0.99783	0.99930	0.99896	0.99931	0.99983	1.00009	0.99965	0.99977	0.99989	1.00002	0.99981	0.99986
3	0.99584	0.99804	0.99805	0.99878	0.99882	0.99912	0.99986	1.00019	0.99974	0.99987	0.99961	0.99972
4	0.99349	0.99732	0.99719	0.99829	0.99860	0.99895	0.99901	0.99922	0.99984	1.00019	0.99970	0.99988
5	0.99153	0.99437	0.99714	0.99796	0.99781	0.99832	0.99870	0.99903	0.99901	0.99927	0.99932	0.99948
10	0.97417	0.98105	0.99119	0.99351	0.99919	1.00044	0.99756	0.99831	0.99794	0.99862	0.99830	0.99873

Note : M(LPTa) represents the makespan of algorithm LPTa
M(LPTo) represents the makespan of algorithm LPTo

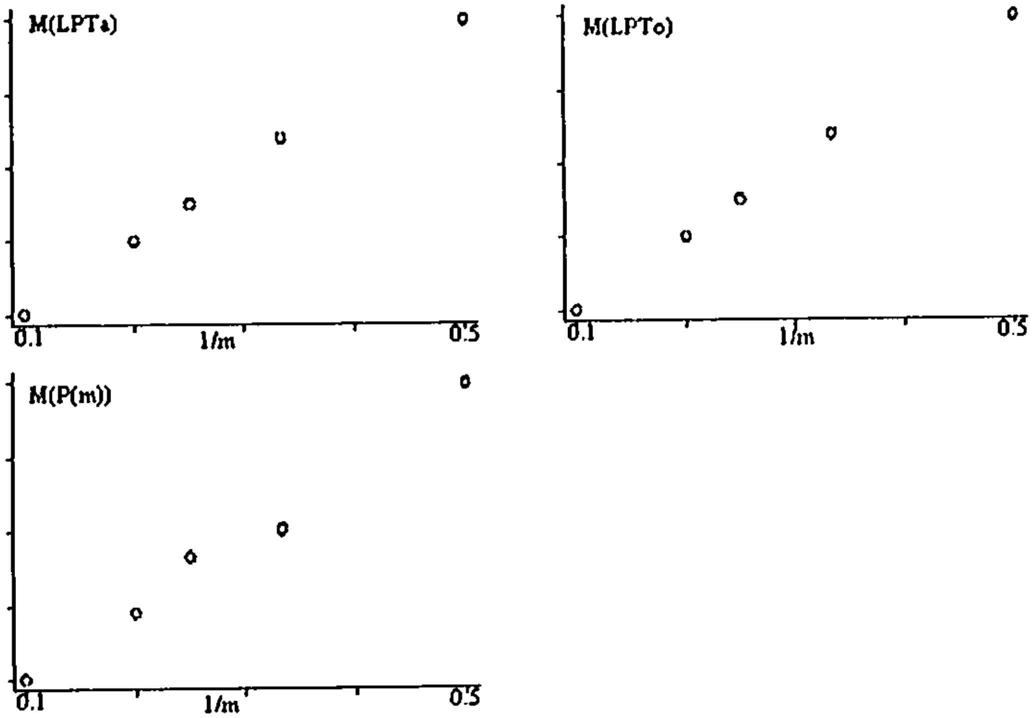
Table A.12 : 95% CI of the Ratio of M(LPTa)/M(LPTo), by # jobs

Appendix B : Graphical Results



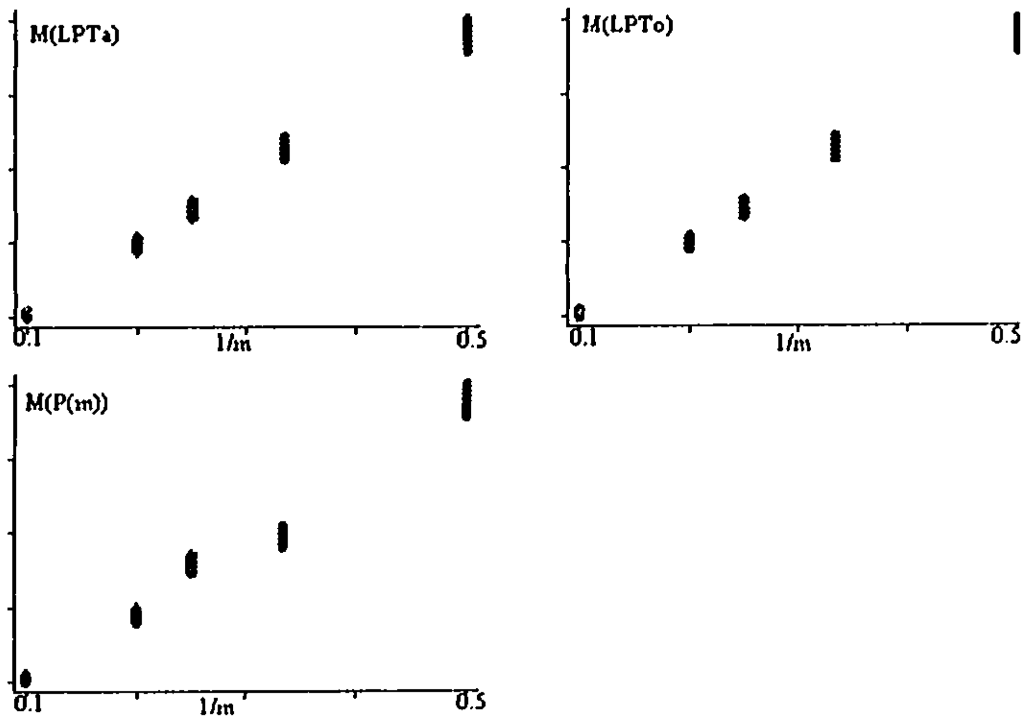
Note : Distribution U(0,1).
m : number of machines.
M(x) : the makespan of algorithm x.

Graph B.1 Makespan M(LPTa), M(LPTo) and M(P(m)) vs. 1/m



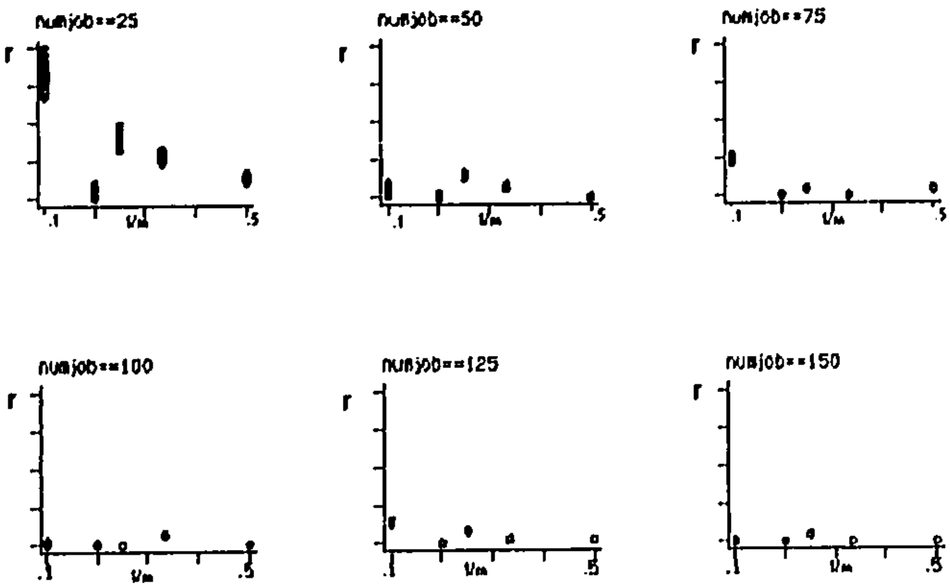
Note : Distribution U(19,20).
m : number of machines.
M(x) : the makespan of algorithm x.

Graph B.2 Makespan M(LPTa), M(LPTo) and M(P(m)) vs. 1/m



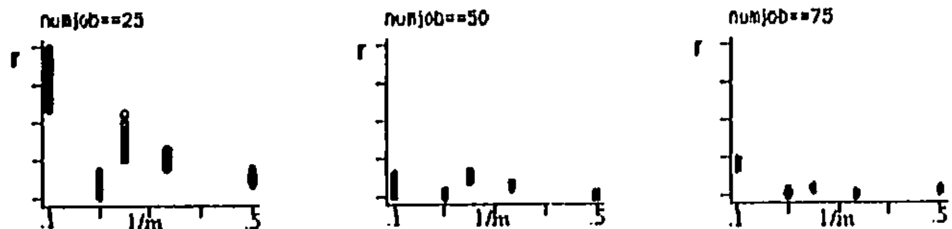
Note : Distribution U(5,10).
m : number of machines.
M(x) : the makespan of algorithm x.

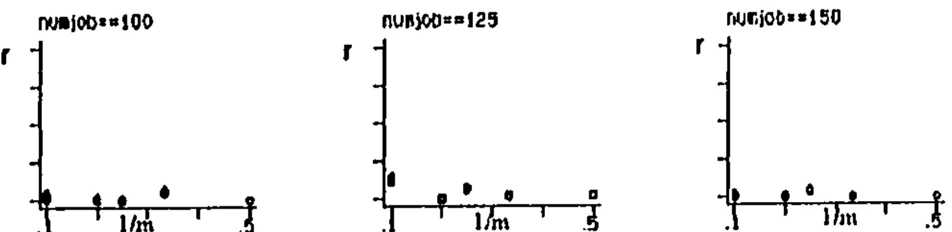
Graph B.3 Makespan M(LPTa), M(LPTo) and M(P(m)) vs. 1/m



Note: Distribution: Normal (5, 10).
m : number of machines.
M(LPTa): the makespan of algorithm LPTa.
LB: the lower bound of the makespan problem.
r = M(P(m))/LB.
numjob: number of jobs.

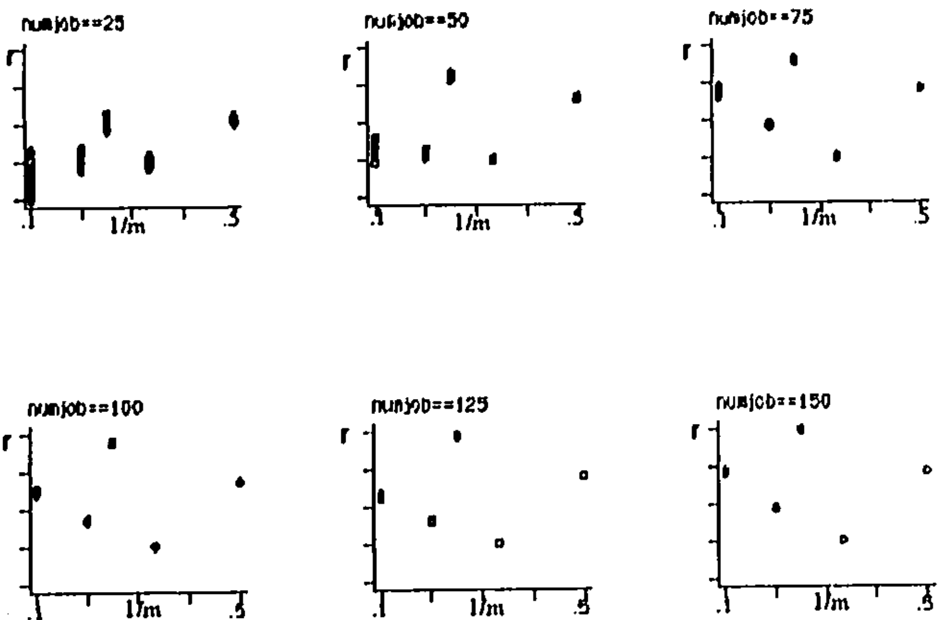
Graph B.4 The Ratio of M(LPTa)/LB vs 1/m, by # Jobs





Note: Distribution: Normal (5, 10).
m : number of machines.
M(LPTo): the makespan of algorithm LPTo.
LB: the lower bound of the makespan problem.
r = M(P(m))/LB.
numjob: number of jobs.

Graph B.5 The Ratio of M(LPTo)/LB vs. 1/m, by # Jobs



Note: Distribution: Normal (5, 10).
m : number of machines.
M(P(m)): the makespan of algorithm P(m).
LB: the lower bound of the makespan problem.
r = M(P(m))/LB.
numjob: number of jobs.

Graph B.6 The Ratio of M(P(m))/LB vs. 1/m, by # Jobs

Appendix C : Simulation Program

C.1 Program : Problem_generator

```
{$M 65000,5000,5000}
{$N+}
Program problem_generator (input, output);

{ Name : Makeprob.pas}
{ 1. Generating suitable distributed random numbers based on the pseudo-random
     numbers on U(0,1)
  2. Examing different distributions : U--Uniform, N--truncated Normal,
     E--shifted Exponential
  3. Giving the number of machines : num_machines = 2,3,4,5,10;
  4. Calling simulation run.

 Varibles are:
     number of jobs                     : num_jobs;
     number of machines                 : num_machines;
     type of distributions              : disttype;
     # of repetition                    : rep;
     LPT of 'LPTo'                      : ord_lpt;
     LPT of 'LPTa'                      : act_lpt;
     LPT of 'Ordinal Algorithms'        : m_lpt;
     lower bound of makespan problem    : lb;
     Distribution parameters            : dparm1, dparm2;
     pseudo r.n. generator seed         : seed, rnsx, rnsy, rnsz;
 All these variables are GLOBAL.

 Currently procedure call:
    ordalg42.pas  ( simulation run )
    curdate.pas   ( time update )
    whrng1.pas    ( function to generate a pseudo r.n. )
}

uses DOS, CRT;

const
 maxnum_job = 150;          { maximum number of jobs }
 maxnum_machine = 15;       { maximum number of machines }
 maxrep = 250;              { maximum number of replicatins}

type
  jary = array [1..maxnum_job] of real;
  mary = array [1..maxnum_machine] of real;
  mjary= array [1..maxnum_machine, 1..maxnum_job] of real;
  maryi= array [1..maxnum_machine] of integer;


var
 num_jobs, num_machines, rep: integer;
 disttype: char;
```

```
        {U = uniform, N = Normal, E = Exponential }
seed: real;
dparm1, dparm2  : real;
        { Uniform (a,b)         : a=dparm1, b=dparm2 }
        { Normal  (mu, sigma)   : mean=dparm1, stdev=dparm2 }
        { Exponential (d, shift) : d=dparm1, shift=dparm2 }
ord_lpt, act_lpt, m_lpt, lb: real;
num_rep: integer;    { current # of replications }
disx, disy, disz: real;
rnsx, rnsy, rnsz: integer; { global seeds }
status: boolean;
fname, disfname, algfname: string;
    { file name with NO extension}
    { file name of distributed random numbers = fname + '.dat' }
    { file name of simulation results = fname + '.sol' }
con, disf, algf, indfile : text;
dr, u1, u2, b, s: real;
i, j: integer;

{ Current program calls following procedures/functions }
{$I curdate.pas}      { time updated precedure }
{$I whrng.pas}        { pseudo r.n. generating function }
{$I ordalg.pas}       { simulation run }

var
 mach_set: array [1..10] of integer;
 pos: boolean;
 means, vars : array[1..maxrep] of double;
 meanm, meanv, varm, varv : double;
 mean, vvar : double;  { sample mean and sample standard deviation }

{ Main program }

begin
  assign(con,'CON');
  rewrite(con);
  assign(input,''); reset(input);
  assign(output,''); rewrite(output); {needed for I/O redirection}

  { initialize }
  for i:=1 to maxrep do
  begin
    means[i] := 0.0;
    vars[i]  := 0.0;
  end; { for }
  meanm := 0.0;
  meanv := 0.0;
  varm  := 0.0;
  varv  := 0.0;
  pos := true;
  num_rep :=1;   { initialize current replication }

  { write information on screen }
  writeln(' -- prepare machine job assigment problems');
```

```
write('Number of Jobs    : '); readln(num_jobs);
write('Distribution type : '); readln(disttype);
write('# of Replications : '); readln(rep);
write('Pseudo Random Number Generator Seed : '); readln(seed);
disttype := upcase(disttype);
rnsx := round(seed*2);        { initialize global seeds }
rnsy := round(seed+10000);
rnsz := round(seed+5000);

case disttype of      { pass parameters to each distribution }
 'U' : begin {uniform}
      writeln('Parameters of Uniform distribution:');
      while pos do
      begin
       write('a: '); read(dparm1);
       write('b: '); readln(dparm2);
       if dparm1 <= dparm2 then pos := false
       else
         writeln(' !!! Error input: parameter a >= b');
      end; { while }
      pos := true;
      writeln('Mean and variance:',0.5*(dparm1+dparm2),' ',
            (dparm2-dparm1)*(dparm2-dparm1)/12.0);
     end; { uniform }
 'N' : begin {truncated Normal}
       writeln('Parameters of Normal distribution: ');
       while pos do
       begin
        write(' mu = '); readln(dparm1);
        write(' sigma  (not SQUARED) '); readln(dparm2);
        if (dparm1>=0) and (dparm2>=0) then pos := false
        else
          writeln('!!! Error Input: negative input');
       end; { while }
       pos := true;
       writeln('Mean and variance:',dparm1,' ',
             dparm2*dparm2);
     end; { Normal }
 'E' : begin { Exponential }
       write('Parameters of Exponential distribution: ');
       while pos do
       begin
        write(' b = '); readln(dparm1);
        write(' shift = '); readln(dparm2);
        if dparm1 >= 0 then pos := false
        else
          writeln('!!! Error input: Negative input.');
       end; { while }
       pos := true;
       writeln('Mean and variance:',
             (dparm1*dparm1*dparm2+1.0)/(dparm1*dparm1),' ',
             1.0/(dparm1*dparm1*dparm1*dparm1));
     end; { Exponential }
 else begin
```

```
write('Not ready.'); readln;
        halt;
      end; { else }
  end; { case }

 write('Filename (NO extension) : '); readln(fname);
 disfname := fname + '.dat';
 algfname := fname + '.sol';

 assign(disf, disfname);
 rewrite(disf);
 assign(algf, algfname);
 rewrite(algf);
{  assign(indfile, 'test.ind');
 rewrite(indfile);  }
 { write information to distributed r.n. file }
 write(disf, '# ');
 curdate(disf);
 writeln(disf, '# Data file name    : ', disfname);
 writeln(disf, '# Number of jobs    : ', num_jobs);
 writeln(disf, '# Total number of replications: ', rep);

 writeln(disf, '#');
 case disttype of
 'U': writeln(disf,'# Uniform distribution');
 'G': writeln(disf,'# Truncated Normal distribution');
 'E': writeln(disf,'# Shifted Exponential distribution');
 else begin
      writeln(' Failure -- what distribution ?');
      readln; halt;
    end;
 end; {case}

 case disttype of   { Generate suitable distributed random numbers }
 'U': begin        { Uniform }
      writeln(disf, '# a = ',dparm1:10:5,'  b = ',dparm2:10:5);
      writeln(disf, '#');
      for num_rep :=1 to rep do
      begin
        writeln(disf, '# Replicate');
        writeln(disf, num_rep:7);
        mean := 0.0;
        vvar := 0.0;
        for i:=1 to num_jobs do
        begin
          { transform  a pseudo-random number to a Uniform r.n. on (a,b) }
          dr := dparm1+(dparm2-dparm1) * whrnd;
          write(disf, dr:8:6, ' ');
          if 8*int(i div 8) = i then writeln(disf);
          mean:=mean+dr; vvar:=vvar+dr*dr;
        end;  {  for }
        if 8*int(i div 8) <> i then writeln(disf);
        writeln(disf, '# end of replicate');
            { calculate sample mean for each replication }
```

```
      mean:=mean/num_jobs;
           { calculate sample standard deviation for each replication }
      vvar:=(vvar-num_jobs*mean*mean)/(num_jobs-1);
      means[num_rep]:=mean; vars[num_rep]:=vvar;
    end; { for }
  end;  { case U }
'N': begin  { Normal }
    writeln(disf, '# mu = ',dparm1:10:5,'  sigma = ',dparm2:10:5);
    writeln(disf, '#');
    for num_rep :=1 to rep do
    begin
     writeln(disf, '# Replicate');
     writeln(disf, num_rep:7);
     j := 0;
     s := 1.0;
     mean := 0.0;
     vvar := 0.0;
     for i:=1 to num_jobs do
     begin
      repeat
       if j = 1 then
       begin
        dr := b*u2;  { generate a standard Normal r.n. }
        j := 0;
       end
       else
       begin { j=0 }
        repeat
         u1 := 2.0*whrnd-1.0;
         u2 := 2.0*whrnd-1.0;
         s  := u1*u1 + u2*u2;
        until s<1.0;
        b := sqrt(-2.0*ln(s)/s);
        dr := b*u1;            { generate a standard normal r.n. }
        j := 1;
       end; { else }
           {transform a standard normal r.n to a general normal r.n.}
       dr := dparm1 + dr*dparm2;
       if dr<0 then j:=0;
      until dr>=0;       { truncated if r.n. <0 }
      write(disf, dr:8:6, ' ');
      if 8*int(i div 8) = i then writeln(disf);
      mean:=mean+dr; vvar:=vvar+dr*dr;
     end; { for }
     if 8*int(i div 8) <> i then writeln(disf);
     writeln(disf, '# end of replicate');
          { calculate sample mean for each replication }
     mean:=mean/num_jobs;
          { calculate sample standard deviation for each replication }
     vvar:=(vvar-num_jobs*mean*mean)/(num_jobs-1);
     means[num_rep]:=mean; vars[num_rep]:=vvar;
    end; { for }
  end;  { case N }
'E': begin  { Exponential }
```

```
    writeln(disf, '# d = ',dparm1:10:5, '  shift = ', dparm2:10:5);
    writeln(disf, '#');
    for num_rep :=1 to rep do
    begin
     writeln(disf, '# Replicate');
     writeln(disf, num_rep:7);
     mean := 0.0;
     vvar := 0.0;
     for i:=1 to num_jobs do
     begin
        { transform a pseudo r.n. to a shifted exponential r.n. }
      dr := dparm2+ln(1.0/(1.0-whrnd))/(dparm1*dparm1);
      write(disf, dr:8:6, ' ');
      if 8*int(i div 8) = i then writeln(disf);
      mean:=mean+dr; vvar:=vvar+dr*dr;
     end; { for }
     if 8*int(i div 8) <> i then writeln(disf);
     writeln(disf, '# end of replicate');
        { calculate sample mean for each replication }
     mean:=mean/num_jobs;
        { calculate sample standard deviation for each replication }
     vvar:=(vvar-num_jobs*mean*mean)/(num_jobs-1);
     means[num_rep]:=mean; vars[num_rep]:=vvar;
    end; { for }
   end; { case E }
end; {case}

writeln(disf,'## display and compute means and variances');
for i:=1 to rep do
begin
 writeln(disf,'## ',i:4,' ',means[i]:10:6,' ',vars[i]:10:6);
 meanm:= meanm+means[i];
 varm := varm+means[i]*means[i];
 meanv:= meanv+vars[i];
 varv := varv+vars[i]*vars[i];
end;
meanm:=meanm/rep;
meanv:=meanv/rep;
varm :=(varm-rep*meanm*meanm)/(rep-1);
varv :=(varv-rep*meanv*meanv)/(rep-1);
writeln(disf,'## MEANS - mean=',meanm:10:6,'  variance=',varv:10:6);
writeln(disf,'## VARIANCES - mean=',meanv:10:6,'  variance=',varm:10:6);

{ set up num_machines = 2, 3, 4, 5, 10 }
mach_set[1] := 2;
mach_set[2] := 3;
mach_set[3] := 4;
mach_set[4] := 5;
mach_set[5] := 10;
mach_set[6] := -1;
for i:=1 to 6 do
begin
 num_machines := mach_set[i];
 if num_machines <> -1 then
```

```
    ordalg(i, disf, algf, algfname);  { call simulation run }
  end; { for }


  { Minitab command to input the }
  { sample means and sample stdevs for replications }
  writeln(algf, 'read  c36 c37');
  for i:=1 to rep do
    writeln(algf, means[i]:10:6,' ',vars[i]:10:6,' ');
  writeln(algf, 'end');
  writeln(algf, 'name c36 ',chr(39),fname,'means',chr(39),
          ' c37 ',chr(39),fname,'vars',chr(39));

  writeln(algf,'SAVE ',chr(39),fname,chr(39));

  flush(disf);
  close(disf);
  flush(algf);
  close(algf);
{   flush(indfile);
  close(indfile);
}
end. {problem_generator}
```

C.2 Procedure : Ordalg

-- Simulation Run

```
Procedure ordalg ( num_run : integer;
          var indata, df : text; dfname: string);

{ Programe Name: ordalg.pas }
{ Parameters passed in
   num_run    : current # of simulation run;
   indata     : data file with distributed ramdom numbers
   df, dfname : file for simulation resutls

 Notation of algorithms :
   LPTa : LPT algorithm using actual data for all m;
   LPTo : LPT algorithm using ordinal data for all m;
   P(2)  : ordinal algorithm when m=2;
   P(3)  : ordinal algorithm when m=3;
   P(4)  : ordinal algorithm when m=4;
   P(m)  : ordinal algorithm when m>=5;

 1. Calculate the makespan of LPTa, LPTo and P(m)
 2. Calculate the Average Completion Time of jobs of LPTa, LPTo and P(m)
 3. Calculate the lower bound of makespan problem.

 Functions and Procedures :

 Function   unbusy ( iary : jary    ) : integer;
   ---> Find the smallest value in an array and
        RETURN the corresponding array index of that element
 Function   busy  ( iarr : jary    ) : integer;
   ---> Find the largest value in an array and
        RETURN the corresponding array index of that element
 Function   largest  ( iary : jary  ) : real;
   ---> Find the largest value in an array and RETURN it.
 Function   lowerb  ( nm, nj: integer; iary : jary  ) : real;
   ---> Calculate the lower bound of makespan problem.

 Procedure  sort   ---> Sort input data from large to small;
 Procedure  lpta   ---> LPT for actual data on any number of machines,
                   perform algorithm LPTa;
 Procedure  lpto   ---> LPT for ordinal data on any number of machines,
                   perform algorithm lpto;
 Procedure  ord_two_proc   ---> Ordinal algorithm with 2 machines,
                       perform algorithm P(2);
 Procedure  ord_three_proc ---> Ordinal algorithm with 3 machines,
                       perform algorithm P(3);
 Procedure  ord_four_proc  ---> Ordinal algorithm with 4 machines,
                       perform algorithm  P(4);
 Procedure  ord_any_proc   ---> Ordinal algorithm with m (m>4) machines,
                      machines, perform algorithm P(m);
 Procedure  DataEvl ---> Given the input information, eg. number of
                   machines and data file, display the job assignment
                   on machines to a file.
```

```
}

{ call procedure for time update }
{$I curdate.pas}

{-----------------------------------------------------------------
   Function  istr : convert a value to a string
------------------------------------------------------------------}
function istr(myint: integer): string;
var
 mys: string;
begin
 str(myint, mys);
 istr:=mys;
end; { istr }

{-------------------------------------------------------------------
   Procedure : sort
   Sort input data from large to small
   Sort an array using the 'Quicksort' algorithm (recursive form)

   The source code is from: Program SortDemo.
              Author: Richard R. Rebouche
-------------------------------------------------------------------}

 Procedure Sort (var A : jary; L, R : Integer);

 Var
   I, J   : Integer;
   X, temp : real;

 Begin
  I := L;  J := R;
  X := A[trunc((L+R) Div 2)];
  Repeat
    While A[I] > X do
      I := I + 1;

    While X > A[J] do
     J := J - 1;
    If I <= J then
    Begin
     temp := A[I];   { swich A[i] and A[j] }
     A[I] := A[J];
     A[J] := temp;
     I := I+1;  J := J-1;
    End; { if }
   Until I > J;
   If L < J Then
    Sort (A, L, J);
   If I < R Then
    Sort (A, I, R);

 End;  { sort }
```

```
procedure quicksort (Var A : jary);

Begin
 Sort (A, 1, num_jobs);
End; { quicksort }
{--------------------------------------------------------------------
     Function : unbusy
     Find the smallest value in an array and RETURN the
     corresponding array index if the value of two array tie,
     then return the smaller index
--------------------------------------------------------------------}
Function   unbusy ( iary : mary ) : integer;

var
  temp : real;
  i : integer;
begin
 unbusy := 1;
 temp := iary[1];
 for i:=2 to num_machines do
    if temp > iary[i] then
    begin
     temp := iary[i];
     unbusy := i;
    end; { if }
end;  { function unbusy }


{--------------------------------------------------------------------
     Function : busy
     Find the largest value in an array and RETURN the corresponding
     array index if the value of two array tie,
     then return the smaller index
--------------------------------------------------------------------}
Function   busy  ( iarr : mary ) : integer;

var
  i,j : integer;
begin
 j := 1;
 for i:=2 to num_machines do
    if iarr[j] < iarr[i] then j := i;
 busy := j;
end;  { function busy }

{--------------------------------------------------------------------}
{     Function : largest
{          Find the largest value in an array and RETURN it.          }
{--------------------------------------------------------------------}
Function  largest  ( iary : mary ) : real;

var
  temp : real;
```

```
  i : integer;
begin
  temp := iary[1];
  for i:=2 to num_machines do
     if temp < iary[i] then temp := iary[i];
  largest := temp;
end;  { function largest }

{-------------------------------------------------------------------
     Function : lowerb
     Find the Lower Bound of makespan problem for m parallel machines
-------------------------------------------------------------------}
Function   lowerb  ( nj, nm: integer; iary : jary ) : real;
  {  nj: number of jobs;
    nm: number of machines;
    iary: input array;
  }


var
  lowb: array [1..3] of real;
  temp: real;
  i: integer;
begin
  if nj<=nm then
    for i:=nj+1 to nm+1 do
       iary[i] := 0;
  lowb[1] := iary[1];
  lowb[2] := iary[nm] + iary[nm+1];
  lowb[3] := 0;
  for i:=1 to nj do
   lowb[3] := lowb[3] + iary[i];
  lowb[3] := lowb[3]/nm;
    (* lower bound = MAX { a(1), a(m)+a(m+1), sum(a(i))/num_jobs } *)
  if lowb[1] >= lowb[2] then temp := lowb[1]
  else temp := lowb[2];
  if lowb[3] > temp then temp := lowb[3];
  lowerb:=temp;
end; { lowerb }

(*
{-------------------------------------------------------------------
   Procedure : job_assign
      write down the job assignment on each machines
 -------------------------------------------------------------------}
Procedure  job_assign ( im : mjary;
                   nj_mch : maryi;
                   tm : mary;
                   ac : real );


 { parameters  im[i,j]  : j-th job on machine i
           nj_mch[i]: # of jobs on machine i
           tm[i]    : total time on machine i
```

```
        ac      : average completion time  }

var
 i, j : integer;
begin
 writeln(indfile);
 writeln(indfile, '**************************************************');
 curdate(indfile);
 writeln(indfile, 'File Name : ', 'test.ind');
 writeln(indfile, 'Number of machines : ', num_machines);
 writeln(indfile, 'Number of jobs : ', num_jobs);
 writeln(indfile, 'Data distribution : ', disttype);
 write(indfile, 'Number of replications : ', rep, '     ');

 writeln(indfile, 'Current number of rep. : ', num_rep);
 writeln(indfile, '----------------------------------------------');
 for i:=1 to num_machines do
 begin
  writeln(indfile, 'Jobs on machine: ', i);
  for j:=1 to nj_mch[i]  do
  begin
   write(indfile, im[i,j]:9:4,' ');
   if 8*int(j div 8) = j then writeln(indfile);
  end; { for j }
  if 8*int(j div 8) <> j then writeln(indfile);
  writeln(indfile, 'Total Machine Time : ', tm[i]:10:4);
  writeln(indfile);
 end;  { for i }
 writeln(indfile, 'Average Completion Time : ', ac:10:4);
 writeln(indfile, '----------------------------------------------');
 writeln(indfile);
end;  { procedure }
*)

{-----------------------------------------------------------------
 Procedure : LPTa
    LPT algorithm for actual data to assign n jobs to m machines,
    return the longest processing time on machines and
    the logest processing time.
 -----------------------------------------------------------------}
Procedure  lpta   ( nm, nj  : integer;
                  in_data : jary;
                  var lt, act: real);

{
 variable nm : number_of_machines
       nj : number_of_jobs
       in_data : sorted processing time of jobs
       lt : logest processing time
       act: average completion time
}
```

```
var
  i, j, q, t : integer;
  mt         : mjary;
  numj_mt    : maryi;
  total_mt   : mary;


begin
 for i:=1 to nm do  { initialize mt, numj_mt and total_mt }
 begin
  for j:=1 to nj do
     mt[i,j]:= 0.0;
  numj_mt[i] := 0;
  total_mt[i]:= 0.0;
 end;  { ini }
 act := 0.0;

 if nm >= nj then   { number of machines >= nunber of jobs }
 begin              { 1 job goes to each machine }
  for i:=1 to nj do
  begin
   numj_mt[i] := 1;
   mt[i,1] := in_data[i];
   total_mt[i] := total_mt[i] + in_data[i];
  end;  { for }
 end;  { if }
 if nm < nj then    { number of machines < nunber of jobs }
 begin
  for i:=1 to nm do       { first nm out of nj jobs assigned to each
                        machines, the longest one gose first }
  begin
   numj_mt[i] := 1;
   mt[i,1] := in_data[i];
   total_mt[i] := in_data[i];
  end;  { for }
  for j:=nm+1 to nj do      { assign the rest of nj jobs }
  begin
   q := unbusy( total_mt );
   numj_mt[q] := numj_mt[q] +1;
               { keep tracking of time used on each machines }
   mt[q,numj_mt[q]] :=  mt[q,numj_mt[q]] + in_data[j];
   total_mt[q] := total_mt[q] + in_data[j];
  end; { for j }
 end;  { if }

 for i:=1 to nm do         { find the average completion time }
   for j:=1 to numj_mt[i] do
     act := act + j*mt[i,j];
 act := act/nj;
 lt := largest (total_mt);  { find the longest processing time }

(*
 job_assign( mt, numj_mt, total_mt, act);
```

```
 writeln(indfile, '@@@  End of LPT Actual Algorithm  @@@':50);
 writeln(indfile);
*)
end;  { procedure a1_act }

{-----------------------------------------------------------------------
 A1 procedure : LPT algorithm for ordinal data to assign n jobs to
            m machines, return the total processing time on each
            machines and the logest processing time.
    machines    jobs
      1        1  8  9  16
      2 (i)    2  7  10 15
      3        3  6  11 14
      4        4  5  12 13
              (j) (q)
-----------------------------------------------------------------------}
Procedure  a1_ord  ( nm, nj  : integer;
                in_data : jary;
               var lt,act: real);

{
 variable nm : number_of_machines
        nj : number_of_jobs
        in_data : sorted processing time of each job
        lt : longest processing time
        act: average completion time
}

var
  i, j, q, k : integer;
  mt : mjary;
  numj_mt : maryi;
  total_mt : mary;



begin
 for i:=1 to nm do  { initialize mt, numj_mt and total_mt }
 begin
  for j:=1 to nj do
     mt[i,j]:= 0.0;
  numj_mt[i] := 0;
  total_mt[i]:= 0.0;
 end;  { ini }
 act := 0.0;
 if nm > = nj then   { number of machines >= nunber of jobs }
 begin              { 1 job goes to each machine }
  for i:=1 to nj do
  begin
   numj_mt[i] := 1;
   mt[i,1] := in_data[i];
  end;  { for }
 end;  { if }
```

```
if nm < nj then   { number of machines < nunber of jobs }
begin
 for i:=1 to nm do
 begin
  k:=0;           { cycle of 2*nm }
  j:=i+2*nm*k;       { first half of cycle }
  q:=2*nm+1-i+2*nm*k; { second half of cycle }

  while j<=nj do   { assign first half of cycle }
  begin
   numj_mt[i] := numj_mt[i] + 1;
   mt[i, 2*k+1] := in_data[j]; { track the time on each machines }
   k := k+1;
   j := i+2*nm*k;
  end; { while }

  k := 0;
  while q<=nj do { assign second half of cycle }
  begin
   numj_mt[i] := numj_mt[i] +1;
   mt[i, 2*k+2] := in_data[q]; { track the time on each machines }
   k := k+1;
   q := 2*nm+1-i+2*nm*k;
  end; { while }
 end; { for }
end; { if }


for i:=1 to nm do
begin
 for j:=1 to nj do
 begin
  total_mt[i] := total_mt[i] + mt[i,j];
  act := act + j*mt[i,j];
 end; { for j }
end; { for i }
act := act/nj;
lt := largest( total_mt );
(*
 job_assign( mt, numj_mt, total_mt, act);
 writeln(indfile, '@@@  End of LPT Ordinal Algorithm  ':50);
 writeln(indfile);
*)
end; { procedure a1_ord }

{-----------------------------------------------------------------
A2 procedure : Algorithm for ordinal data to assign n jobs to
           2 machines, return the total processing time on each
           machines and the logest processing time.

   machines   jobs
     1       1  4  7
     2       2 3 5 6 8 9
-----------------------------------------------------------------}
```

```
Procedure ord_two_proc ( nj : integer;
                  in_data : jary ;
                 var lt,act: real);

{
 variable nj : number_of_jobs
        in_data : sorted processing time of each job
        lt : logest processing time
        act: average completion time
}

var
  d1, d2, d3, i, j : integer;
  mt     : mjary;
  numj_mt  : maryi;
  total_mt: mary;
  route   : boolean;


begin
 for i:=1 to 2 do  { initialize mt, numj_mt and total_mt }
 begin
  for j:=1 to nj do
     mt[i,j]:= 0.0;
  numj_mt[i] := 0;
  total_mt[i]:= 0.0;
 end;  { ini }
 act := 0.0;
 i  := 0; {cycle}    { initialize }
 route := true;
  while route do
  begin
   d1 := 3*i+1;
   d2 := 3*i+2;
   d3 := 3*i+3;
   if d1<=nj then
   begin
     numj_mt[1] := numj_mt[1] +1;
     mt[1, numj_mt[1]] := in_data[d1];
   end  { if }
   else route := false;
   if d2<=nj then
   begin
     numj_mt[2] := numj_mt[2] +1;
     mt[2, numj_mt[2]] := in_data[d2];
   end  { if }
   else route := false;
   if d3<=nj then
   begin
     numj_mt[2] := numj_mt[2] +1;
     mt[2, numj_mt[2]] := in_data[d3];
   end  { if }
   else route := false;
   i:=i+1;
```

```
 end; { while }

 for i:=1 to 2 do
 begin
  for j:=1 to numj_mt[i] do
  begin
   total_mt[i] := total_mt[i] + mt[i,j];
   act := act + j*mt[i,j];
  end;  {  for j }
 end;  { for i }
 act := act/nj;
 if total_mt[1] > total_mt[2] then lt:=total_mt[1]
 else lt:=total_mt[2];

(*
 job_assign(mt, numj_mt, total_mt, act);
 writeln(indfile, '@@@  End of Ordinal Algorithm for 2 machine ':50);
 writeln(indfile);
*)
end; { A2 }
{--------------------------------------------------------------
 A3 procedure : Algorithm for ordinal data to assign n jobs to
              3 machines, return the total processing time on each
              machines and the logest processing time.
    machines    jobs
      1         1   6     11
      2         2 5   8 10    13 15
      3         3 4   7 9     12 14
--------------------------------------------------------------}
Procedure  ord_three_proc  ( nj : integer;
                        in_data : jary;
                      var lt, act : real);


 {
  variable nj : number_of_jobs
         in_data : sorted processing time of each job
         lt : longest processing time
         act: average completion time
 }
 var
  d1, d2, d3, d4, d5, i, j : integer;
  route   : boolean;
  mt      : mjary;
  numj_mt  : maryi;
  total_mt: mary;

 begin
  for i:=1 to 3 do   { initialize mt, numj_mt and total_mt, nm=3 }
  begin
   for j:=1 to nj do
      mt[i,j]:= 0.0;
   numj_mt[i] := 0;
   total_mt[i]:= 0.0;
```

```
end; { ini }
act := 0.0;

if num_machines >= nj then { number of machines >= number of jobs }
begin { 1 job gose to each machine }
 for i:=1 to nj do
 begin
   numj_mt[i] := 1;
   mt[i,1] := in_data[i];
 end; { for }
end; { if }
if num_machines < nj then { number of machines < number of jobs }
begin
 i := 1;
 route := true;

 while route do
 begin
  while ( i<=nj ) and ( i<=5 ) do
  begin { assign first 5 jobs or less }
   case i of
    1   : begin
           numj_mt[1] := numj_mt[1] + 1;
           mt[1,numj_mt[1]] := in_data[i];
          end; { case 1 }
    2, 5 : begin
           numj_mt[2] := numj_mt[2] + 1;
           mt[2,numj_mt[2]] := in_data[i];
          end; { case 2,5 }
    3, 4 : begin
           numj_mt[3] := numj_mt[3] + 1;
           mt[3,numj_mt[3]] := in_data[i];
          end; { case 3,4 }
   end; { case }
   i := i+1;
            { stop if less then 5 jobs or the next job is 6th }
   if (i>nj) or (i>5) then route:=false;
  end; { while }
 end; { while 'route' }

 if i<=nj then { assign jobs start from 6th }
  begin
   route := true;
   i:=1;


   while route do
   begin
    d1 := 5*i+1;
    d2 := 5*i+2;
    d3 := 5*i+3;
    d4 := 5*i+4;
    d5 := 5*i+5;
    if d1<=nj then
```

```
      begin
        numj_mt[1] := numj_mt[1] +1;
        mt[1, numj_mt[1]] := in_data[d1];
      end  { if }
    else route := false;
    if d2<=nj then
      begin
        numj_mt[3] := numj_mt[3] +1;
        mt[3, numj_mt[3]] := in_data[d2];
      end  { if }
    else route := false;
    if d3<=nj then
      begin
        numj_mt[2] := numj_mt[2] +1;
        mt[2, numj_mt[2]] := in_data[d3];
      end  { if }
    else route := false;
    if d4<=nj then
      begin
        numj_mt[3] := numj_mt[3] +1;
        mt[3, numj_mt[3]] := in_data[d4];
      end  { if }
    else route := false;
    if d5<=nj then
      begin
        numj_mt[2] := numj_mt[2] +1;
        mt[2, numj_mt[2]] := in_data[d5];
      end  { if }
    else route := false;
    i:=i+1;
   end; { while }
  end; { if }
end; { if }

for i:=1 to 3 do
begin
 for j:=1 to numj_mt[i] do
 begin
  total_mt[i] := total_mt[i] + mt[i,j];
  act := act + j*mt[i,j];
 end; { for j }
end;  { for i }
act := act/nj;
lt := largest ( total_mt );

(*
 job_assign(mt, numj_mt, total_mt, act);
 writeln(indfile, '@@@  End of Ordinal Algorithm for 3 machine ':50);
 writeln(indfile);
*)
end; { A3 }


{----------------------------------------------------------------
```

```
A4 procedure : Algorithm for ordinal data to assign n jobs to
          4 machines, return the total processing time on each
          machines and the logest processing time.

   machines    jobs
      1        1    8           15          22           29
      2        2 7      11      16      21       25      30      35
      3        3 6   9  12  14     18 20  23  26  28        32  34
      4        4 5  10  13         17  19  24  27           31  33
                  (m=1)       (m=0)      (m=1)        (m=0)
-------------------------------------------------------------------}
Procedure  ord_four_proc  ( nj : integer;
                     in_data : jary;
                    var lt,act: real);

{
 variable nj : number_of_jobs
        in_data : sorted processing time of each job
        lt : longest processing time
        act: average completion time
}
var
  d1, d2, d3, d4, d5, d6, d7, i, j, m, s, t : integer;
  route   : boolean;
  mt      : mjary;
  numj_mt  : maryi;
  total_mt: mary;

begin
 for i:=1 to 4 do   { initialize mt, numj_mt and total_mt, nm=4 }
 begin
  for j:=1 to nj do
     mt[i,j]:= 0.0;
  numj_mt[i] := 0;
  total_mt[i]:= 0.0;
 end;  { ini }
 act := 0.0;

 if num_machines >= nj then
 begin  { 1 job gose to each machine }
  for i:=1 to nj do
  begin
   numj_mt[i] := 1;
   mt[i,1] := in_data[i];
  end; {  for }
 end { if }
 else  { number of machines < number of jobs }
 begin
  i := 1;
  route := true;

  while route do
  begin
   while ( i<=nj ) and ( i<=7 ) do
```

```
begin  { assign first 7 jobs or less }
 case i of
   1    : begin
          numj_mt[1] := numj_mt[1] + 1;
          mt[1,numj_mt[1]] := in_data[i];
        end;  { case 1 }
   2, 7 : begin
          numj_mt[2] := numj_mt[2] + 1;
          mt[2,numj_mt[2]] := in_data[i];
        end;  { case 2,7 }
   3, 6 : begin
          numj_mt[3] := numj_mt[3] + 1;
          mt[3,numj_mt[3]] := in_data[i];
        end;  { case 3,6 }

   4, 5 : begin
          numj_mt[4] := numj_mt[4] + 1;
          mt[4,numj_mt[4]] := in_data[i];
        end;  { case 4,5 }
 end; { case }
 i := i+1;
 if (i>nj) or (i>7) then route:=false;
                          { stop if less than 7 jobs or next job is 8th}
end; { while }
end; { while 'route' }

if i<=nj then { assign the rest of jobs start from 8th }
 begin
  route := true;
  i:=1;
  s:=0;
  t:=0;
  while route do
  begin
    d1 := 7*i+1;
    d2 := 7*i+2;
    d3 := 7*i+3;
    d4 := 7*i+4;
    d5 := 7*i+5;
    d6 := 7*i+6;
    d7 := 7*i+7;
    m  := i mod 2;
    case m of
     0 : begin
         t := t+1;
         if d1<=nj then   { assign a job to machine 1 }
         begin
           numj_mt[1] := numj_mt[1] +1;
           mt[1, i+1]:= in_data[d1];
         end  { if }
         else route := false;
         if d2<=nj then   { assign a job to machine 2 }
         begin
           numj_mt[2] := numj_mt[2] +1;
```

```
    mt[2, i+t+1]:= in_data[d2];
   end { if }
   else route := false;

   if d3<=nj then    { assign a job to machine 4 }
   begin
    numj_mt[4] := numj_mt[4] +1;
    mt[4, 2*i+1]:= in_data[d3];
   end { if }
   else route := false;
   if d4<=nj then    { assign a job to machine 3 }
   begin
    numj_mt[3] := numj_mt[3] +1;
    mt[3, 3*t+i+1]:= in_data[d4];
   end { if }
   else route := false;
   if d5<=nj then    { assign a job to machine 4 }
   begin
    numj_mt[4] := numj_mt[4] +1;
    mt[4, 2*i+2]:= in_data[d5];
   end { if }
   else route := false;
   if d6<=nj then    { assign a job to machine 3 }
   begin
    numj_mt[3] := numj_mt[3] +1;
    mt[3, 3*t+i+2]:= in_data[d6];
   end { if }
   else route := false;
   if d7<=nj then    { assign a job to machine 2 }
   begin
    numj_mt[2] := numj_mt[2] +1;
    mt[2, t+i+2]:= in_data[d7];
   end { if }
   else route := false;
   i:=i+1;
  end; { case '0' }
1 : begin
   s := s+1;
   if d1<=nj then    { assign a job to machine 1 }
   begin
    numj_mt[1] := numj_mt[1] +1;
    mt[1, i+1]:= in_data[d1];
   end { if }
   else route := false;
   if d2<=nj then    { assign a job to machine 3 }
   begin
    numj_mt[3] := numj_mt[3] +1;
    mt[3, 3*s+i-1]:= in_data[d2];
   end { if }
   else route := false;
   if d3<=nj then    { assign a job to machine 4 }
   begin
    numj_mt[4] := numj_mt[4] +1;
    mt[4, 2*i+1]:= in_data[d3];
```

```
            end  { if }
            else route := false;
            if d4<=nj then    { assign a job to machine 2 }
            begin
              numj_mt[2] := numj_mt[2] +1;
              mt[2, s+i+1]:= in_data[d4];
            end  { if }
            else route := false;
            if d5<=nj then    { assign a job to machine 3 }
            begin
              numj_mt[3] := numj_mt[3] +1;
              mt[3, 3*s+i]:= in_data[d5];
            end  { if }
            else route := false;
            if d6<=nj then    { assign a job to machine 4 }
            begin
              numj_mt[4] := numj_mt[4] +1;
              mt[4, 2*i+2]:= in_data[d6];
            end  { if }
            else route := false;
            if d7<=nj then    { assign a job to machine 3 }
            begin
              numj_mt[3] := numj_mt[3] +1;
              mt[3, 3*s+i+1]:= in_data[d7];
            end  { if }
              else route := false;
             i:=i+1;
           end; { case '1' }
        end; { case }
      end; { while }
    end; { if }
  end; { else }

  for i:=1 to 4 do
  begin
    for j:=1 to numj_mt[i] do
    begin
      total_mt[i] := total_mt[i] + mt[i,j];
      act := act + j*mt[i,j];
    end;  { for j }
  end;  { for i }
  act := act/nj;
  lt := largest ( total_mt );

(*
  job_assign(mt, numj_mt, total_mt, act);
  writeln(indfile, '@@@  End of Ordinal Algorithm for 4 machine ':50);
  writeln(indfile);
*)
end; { A4 }

{------------------------------------------------------------------
 A5 procedure : Algorithm for ordinal data to assign n jobs to
              m (m>4) machines, return the total processing time
```

```
        on each machine and the logest processing time.
   machines    jobs
     1          1  14     25     36
     2 (t=3)    2  13     24     35
     3          3  12     23     34
     ----------------------------------------
     4          4  11  18  22  29  33  40
     5 (r=4)    5  10  17  21  28  32  39
     6          6   9  16  20  27  31  38
     7          7   8  15  19  26  30  37
------------------------------------------------------------------}
Procedure ord_any_proc ( nm, nj : integer;
                 in_data : jary;
               var lt,act : real);

{
 variable nm : number_of_machines
        nj : number_of_jobs
        in_data : sorted processing time of each job
        lt : logest processing time
        act: average completion time
}
var
  k, r, t, q, i, j : integer;
  mt : mjary;
  numj_mt : maryi;
  total_mt : mary;

begin
  k := 0;
  if (nm mod 2) = 0 then
     r := round(nm/2)
  else
     r := round(nm/2)+1;
  t := trunc(nm/2);
  for i:=1 to nm do   { initialize mt, numj_mt and total_mt }
  begin
   for j:=1 to nj do
      mt[i,j]:= 0.0;
   numj_mt[i] := 0;
   total_mt[i]:= 0.0;
  end;  { ini }
  act := 0.0;

  if nm>=nj then   { number of machines >= number of jobs }
  begin          { 1 job gose to each machine }
   for i:=1 to nj do
   begin
     numj_mt[i] := 1;
     mt[i,1] := in_data[i];
   end;  { for }
  end; { if }
  if nm < nj then  { number of machines < number of jobs }
  begin
```

```
for i:=1 to nm do  { assign first nm jobs to each machine }
begin
  numj_mt[i] := numj_mt[i] +1;
   mt[i,numj_mt[i]] := in_data[i];
end; { for }
for i:=1 to t do { assign rest of jobs (nj-nm) }
begin  { assign jobs to first round(nm/2) machines }
  q := (2*nm+1-i) + k*(nm+r);
  while q<= nj do
  begin
    numj_mt[i] := numj_mt[i] +1;
    mt[i,k+2] := in_data[q];
     k := k+1;
    q := (2*nm+1-i) + k*(nm+r);
  end; { while }
k:=0;
end; { for }


 k := 0;
 for i:=t+1 to nm do  { assign jobs to the reat of machines }
 begin
   q := (2*nm+1-i) + k*(nm+r);
   j := (3*nm+1-i) + k*(nm+r);
   while q<=nj do
   begin
     numj_mt[i] := numj_mt[i] +1;
     mt[i,2*k+2] := in_data[q];
      k := k+1;
     q := (2*nm+1-i) + k*(nm+r);
   end; { while }

   k := 0;
   while j<=nj do
   begin
     numj_mt[i]  := numj_mt[i] +1;
     mt[i,2*k+3]:= in_data[j];
      k := k+1;
     j := (3*nm+1-i) + k*(nm+r);
   end; { while }
  k:=0;
  end; { for }
end; { else }

for i:=1 to nm do
begin
 for j:=1 to numj_mt[i] do
 begin
   total_mt[i] := total_mt[i] + mt[i,j];
   act := act + j*mt[i,j];
 end;  { for j }
end; { for i }
act := act/nj;
lt := largest( total_mt );
```

```
(*
  job_assign(mt, numj_mt, total_mt, act);
  writeln(indfile, '@ @ @  End of Ordinal Algorithm for m machine ':50);
  writeln(indfile);
*)
end; { A5 procedure }

{=============================================================}
{  Main Part }

var
  i, j, n, m   : integer;
  al : real;     { alfa }
  olp, alp, mlp, lob: real;
  o_act, a_act, m_act: real;
  in_data  : jary;
  iscomd   : char;
  firstd   : string;
  code     : integer;
  minitstr, ts : string;
  nmach : char;


begin

  reset(indata);
  num_rep:=0;
  al := 0;
  write(df, '# ');
  curdate(df);
  writeln(df, '#  File Name: ', dfname);
  writeln(df, '# Number of jobs: ', num_jobs);
  writeln(df, '# Number of machines: ', num_machines);
  writeln(df, '# Seed: ', seed:10:4);
  writeln(df, '# Distribution: ', disttype);
  writeln(df, '# Parameters: ', dparm1:8:2, dparm2:8:2, dparm3:8:2 );
  writeln(df);

  case disttype of
  'U','G','E' : begin
             writeln(df, '#','LPTa':10,'LPTo':10,'LPTm':10,
                    'ACTa':10, 'ACTo':10, 'ACTm':10, 'Lower B.':10);
           end;
  end;  { case }

  if num_machines = 10 then nmach:='x'
  else begin
    ts:=istr(num_machines);
    nmach:=ts[1];
  end;

  writeln(df, 'READ C',1+(num_run-1)*7, '-C', num_run*7);
```

```
while not EOF( indata ) do
begin
 iscomd := ' ';
 while iscomd=' ' do
  read( indata, iscomd );
  if iscomd = '#' then readln( indata )   { The line starting with '#'}
                                          { is a command, skip it   }
 else   { iscomd <> # }                   { Read data set by set }
 begin
  firstd := '';
  while (iscomd <> ' ') and (iscomd <> chr(13))  do
                                { read first data as a string }
  begin
   firstd := concat(firstd, iscomd);  {merge first data char by char}
   read( indata, iscomd );
  end; { while }
  val( firstd, num_rep, code );    { convert it to value }
  if code=0 then
  begin
    for i:=1 to num_jobs do        { read all the data of processing times}
       read( indata, in_data[i]);
   readln(indata);

   quicksort(in_data);
   lob := lowerb(num_jobs, num_machines, in_data);
   a1_act ( num_machines, num_jobs, in_data, alp, a_act);
   a1_ord ( num_machines, num_jobs, in_data, olp, o_act);
   case num_machines of
    2 : ord_two_proc   (num_jobs, in_data, mlp, m_act);
    3 : ord_three_proc (num_jobs, in_data, mlp, m_act);
    4 : ord_four_proc  (num_jobs, in_data, mlp, m_act);
    else begin
         ord_any_proc (num_machines,num_jobs,in_data,
              mlp,m_act);
       end; { case 'else' }
   end;  { case }

   case disttype of
   'U','G','E' : begin
                 writeln(df,alp:11:4,' ',olp:9:4,' ', mlp:9:4,' ',a_act:9:4,' ',o_act:9:4,' ',
                        m_act:9:4,' ',lob:9:4);
               end;  { case U, G, E }
   end;  { case }

  end { if }
  else begin
      gotoxy(1, 23);
      writeln(' code: ', code);
      writeln( ' Error: Convertion Fail ' );
      exit;
     end;
 end; { else }

end;  { EOF }
```

```
writeln(df);
writeln(df,'END');
writeln(df,'NAME C',num_run*7-6,' ',
        chr(39),fname,'la',nmach,chr(39));
writeln(df,'NAME C',num_run*7-5,' ',
        chr(39),fname,'lo',nmach,chr(39));
writeln(df,'NAME C',num_run*7-4,' ',
        chr(39),fname,'lp',nmach,chr(39));
writeln(df,'NAME C',num_run*7-3,' ',
        chr(39),fname,'aa',nmach,chr(39));
writeln(df,'NAME C',num_run*7-2,' ',
        chr(39),fname,'ao',nmach,chr(39));
writeln(df,'NAME C',num_run*7-1,' ',
        chr(39),fname,'ap',nmach,chr(39));
writeln(df,'NAME C',num_run*7, ' ',
        chr(39),fname,'bl',nmach,chr(39));
writeln(df);

end; { ordalg }
```

C.3 Function : Pseudo-random number generator

```
function whrnd : real;
{whrng.pas -- Wichmann & Hill uniform pseudo-random number generator}
{ Byte Magazine }
var
{  rnsx, rnsy, rnsz: integer;}
 temp: real;
begin
 { first generator }
 rnsx := 171 * (rnsx mod 177) - 2 * (rnsx div 177);
 if rnsx < 0 then rnsx := rnsx + 30269;
 { second generator }
 rnsy := 172 * (rnsy mod 176) - 35* (rnsy div 176);
 if rnsy < 0 then rnsy := rnsy + 30307;
 { third generator }
 rnsz := 170 * (rnsz mod 178) -63* (rnsz div 178);
 if rnsz < 0 then rnsz := rnsz + 30323;
 { combine to give function value }
 temp := rnsx/30269.0 + rnsy/30307.0 + rnsz/30323.0;
 whrnd := temp - trunc(temp)
end; {whrng}
```

C.4 Procedure : Time update

```
procedure curdate(var outfile:text);
   {Source code from "pdate.pas", J C Nash}
   {Display current date in a file}

const
 yearwrit = 1990;

type
 str2 = string[2];

var
 year: word;
 month: word;
 day: word;
 dayofweek: word;
 hour: word;
 minute: word;
 second: word;
 sec100: word;

begin
 getdate(year, month, day, dayofweek);
 if year<yearwrit then writeln('Computer clock/calendar needs setting');
 gettime(hour, minute, second, sec100);
 writeln(outfile, '* Current date & time: ',year,'/',month,'/',day,
       '  ',hour,':',minute,':',second,'.',sec100);
end; {curdate.pas}
```