ANALYSIS OF JOB SCHEDULING ALGORITHMS

FOR HETEROGENEOUS MULTIPROCESSOR COMPUTING SYSTEMS

BY

CONSTANTINOS D. SPYROPOULOS, B.Sc.,M.Sc.

A Doctoral Thesis

Submitted in partial fulfilment of the requirements

for the award of Doctor of Philosophy

of the Loughborough University of Technology

August, 1979.

Supervisor: Professor D.J. EVANS, Ph.D.,D.Sc.

Department of Computer Studies

DECLARATION

I declare that the following thesis is a record of research work carried out by me, and that the thesis is of my own composition. I also certify that neither this nor the original work contained therein has been submitted to this or any other institution for a degree.

C.D. SPYROPOULOS

ABSTRACT

The problem of scheduling independent jobs on heterogeneous multiprocessor models (i.e., those with non-identical or uniform processors) with independent memories has been studied. Actually, a number of demand scheduling non-preemptive algorithms have been evaluated, with respect to their mean flow and completion time performance criterion. In particular, the deterministic analysis has been used to predict the worst-case performance whereas simulation techniques have been applied to estimate the expected performance of the algorithms. As a result from the deterministic analysis, informative worst-case bounds have been proven, from which the behaviour of the extreme performance of the considered algorithms can be well predicted. However, relaxing some or a combination of the system parameters then, our model corresponds to versions which have already been studied (i.e. the classical homogeneous and heterogeneous models or the homogeneous one with independent memories). For such cases, the proven bounds in this thesis either agree or are better and more informative than the ones found for these simpler models. Finally, the analysis of the worst-case and expected performance results reveals that there is a high degree of correlation in the behaviour of the algorithms as predicted or estimated by these two performance measurements, respectively.

DEDICATED TO MY PARENTS

DEMETRIUS and IRENE

ACKNOWLEDGEMENTS

I wish to express my sincere thanks and gratitude to my supervisor, D.J. Evans, Head of Computer Studies Department, for his constructive ideas, interest and constant encouragement throughout my research as well as for his efforts in reading the manuscript of this thesis.

Also, I would like to thank Dr. I.A. Newman, for his useful discussions, Miss J.M. Briers who typed this thesis more than professionally and especially my friend, Dr. N. Missirlis, for numerous conversations and discussions on the subject of this thesis as well as on many other subjects.

My sincere thanks are extended to Miss Maria Tsintanou, whose devoted love, encouragement and trust gave me confidence to carry on and finish this research.

Last, but not least, I wish to thank my parents Demetrius and Irene Spyropoulos for their help, encouragement and financial support without which this work would not have been possible.

TABLE OF CONTENTS

PAGE

CHAPTER 1

INTRODUCTION

The brief history of computing has been characterised by a constant pressure for more and more computing power while at the same time remaining within budget constraints. Until recently, the high cost of processor hardware has imposed the computer manufacturers to design computer systems around a single CPU, with emphasis on optimising its use by the different components of the system. Moreover, since the need for computational power has grown even faster than the developments in electronics, some manufacturers have decided to include more processing elements in order to satisfy such demands. So, a number of computing systems based on tightly coupled processors have been built (i.e., CDC 7600, ILLIAC IV, CDC STAR, etc.). Although there is no doubt about the capability of such systems, they are very expensive and it seems that they could realise their full potential on only a small subset of problems, most of them being of a scientific nature (i.e. problems where matrix manipulation and solutions of linear or partial differential equations are required).

An alternative approach to build computing systems with a desired power and suitable to more general applications would be a local complex of independent processors (i.e. loosely coupled processors). Actually, it was the appearance of cheap micro- and mini-computers with operational and functional characteristics that compared favourably with conventional medium and large-scale mainframe systems, which pushed ahead this approach. More specifically, some of the intuitive advantages of such multimicro- or multimini-computer complexes over a single mainframe are:

- the economics of LSI technology
- reliability
- total system's power
- incremental expansion and
- more effective utilisation of existing equipment.

These advantages are well illustrated in [EN1,2],[F1,2],[Fv],[Ha],[MF75],

[Sh],[Wh],[Wu] or [WL]. In effect, a few such systems are already in operation (i.e., Cmmp., at Carnegie Mellon Univ., DSC, at California Univ., Irvine etc.) and others are under development (i.e. Loughborough Univ. of Tech., etc.).

However, the basic duties for computer researchers are to investigate the behaviour of a proposed system before it appears in the real world and there is no exception in multiprocessor systems based on tightly or loosely coupled processors. Especially, one of the topics of interest is: how the jobs can be organised and allocated to the various processing units in order to achieve certain performance objectives which is an important function in the design of computer operating systems. Generally, assigning external priorities to the jobs, based on a judgement about their importance, and using a particular job-scheduling function could answer this question. Moreover, in systems where the jobs requirements can be predicted or estimated in advance then, a pre-set ordering of the jobs according to their priorities, depending on some attribute or combination of attributes (i.e., memory requirement, estimated processing time, resources required, etc.), could be as important as the scheduling function itself in order attain desired performance goals. Such systems are in our interests. Nevertheless, the choice of a pre-set ordering and/or scheduling function is neither obvious nor an easy task. This can be realised from the substantial research which has been done in this area since the end of the 1960's. Actually, a number of abstract multiprocessor models have been examined and many job scheduling algorithms evaluated under various procedures to form the pre-set ordering.

Apart from the insight and the new scheduling ideas that such studies may reveal for real multiprocessor systems, many of the results obtained in job scheduling have immediate interpretations to several problem areas in operations research, industrial engineering, management science and business

administration. However, it is known that the first steps in performance evaluation of multiprocessor models were taken from the existing results in the above mentioned areas.

Two common approaches to evaluate a scheduling algorithm are its worst-case and expected behaviour. These measures can be predicted and estimated respectively using different analysis. In particular, we must use the deterministic scheduling theory in order to predict the worst-case behaviour. According to this theory everything about the system and jobs' requirements are known in advance. On the other hand, we must use stochastic queueing theory to estimate the expected behaviour of an algorithm. Queueing theory assumes that many things about the system are uncertain. This uncertainty is characterised by probability distributions for the job's requirements, the arrival of each job in the system and the selection of processors or other resources for the various job steps. In addition, the expected behaviour can also be estimated by using simulation techniques.

The main attraction of deterministic scheduling is its precision and its ability either to look for optimal algorithms or to calculate with no uncertainty the worst-case performance of heuristic algorithms. Instead, the stochastic queueing theory is mainly attractive for its compactness; relatively few parameters are required to calculate performance characteristics for complicated systems.

The deterministic approach is a suitable tool for systems where a guaranteed level of performance is required. In addition, it could also direct us to examine more elaborated procedures for pre-set ordering and/or scheduling algorithms. Further, the worst-case performance of job scheduling algorithms, under various pre-set orderings, relatively to the corresponding optimal scheduling process give us the ability to compare and rank them. However, an open question is, whether the deterministic approach could be useful for systems, under our interests, where the expected behaviour is

more meaningful. This could happen if the worst-case behaviour of any algorithm agrees with its expected behaviour. Such a verification would show that these two approaches can be used in a supplementary way where, the deterministic approach will be used to choose the correct pre-set ordering and/or job scheduling algorithm and the queueing network theory or simulation techniques approach to estimate its expected performance. However, the literature contains very little evidence about this and hence further research is needed.

As a matter of fact, this thesis will reveal some evidence towards the degree of correlation between these two approaches. A new abstract multiprocessor model will be examined, which may be interpreted as a multimicro- or multimini- complex design, using the deterministic scheduling theory and simple simulation techniques. In detail, this thesis is organised in the following way. In Chapter 2, a background for the deterministic scheduling theory related to multiprocessor computing systems is given while in Chapter 3, a survey of the previous work done is studied. In Chapter 4, the computation model is defined as well as the specific aims and objectives of this research. Further, in Chapter 5 and 6 we use deterministic analysis to evaluate the worst-case behaviour of a number of scheduling algorithms, under a variety of pre-set orderings, for different performance goals. On the other hand, in Chapter 7 we use simulation techniques to evaluate the expected behaviour of these algorithms. Finally, Chapter 8 is devoted to conclusions and suggestions for further work.

CHAPTER 2

BACKGROUND IN DETERMINISTIC COMPUTER SCHEDULING THEORY

2.1 INTRODUCTION

Generally, scheduling theory is a collection of algorithms, ordering procedures, models, techniques and logical conclusions that provide insight into the job scheduling function. Further, the job scheduling function can be defined as the allocation of available resources over a period of time to perform a set of tasks. (The terms task/job will be used interchangeably). Therefore, since the job scheduling function and the pre-set ordering of the jobs play an important role in the achievement of certain performance goals in computing systems, scheduling theory becomes a vital tool for the prediction of performance evaluation of new proposed systems. When the deterministic or the stochastic theory is used to evaluate scheduling algorithms for computing models, it is generally referred to as *deterministic* or *probabilistic computer scheduling theory* respectively. Similarly, the chosen analysis will characterise the computation models as deterministic or probabilistic ones.

As stated in Chapter 1, a part of this thesis is to analyse a new proposed computational model using deterministic analysis. Therefore, a background in deterministic computer scheduling theory is required. Actually, in this chapter we attempt to give all the necessary definitions and concepts for deterministic scheduling theory to clarify the investigations which will appear in Chapters 5 and 6. However, such concepts can also be found in [Co], [G],[GGJ] or [Li2].

2.2 A GENERAL MODEL

The computation model which is to be described in the following subsections will be so general as to include most of the models studied so far in the deterministic computer scheduling theory.

2.2.1 Resources

The resources in a deterministic computing model can be characterised through the following assumptions:

(a) A computing system consists of two classes of resources, *dedicated* resources and *shared* resources. In each class there are different types of resources.

(b) There is a certain number of units of dedicated resources of each type. A job cannot be executed on more than one unit of each type concurrently, and no other jobs can be executed on the same unit which has already been occupied by a job. Examples of dedicated resources are processors, input-output devices, etc.

(c) There is a unit of each type of shared resource (no loss of generality arises in normalising each type of shared resources to one unit). The execution of a job requires a fraction of a unit of each shared resource, including zero as a possibility. Concurrent execution of a number of tasks might share the same unit of shared resources, provided that the sum of fractions of the unit they share, does not exceed one. Examples of shared resources are core memories, magnetic disks and drums, etc.

(d) The units of a dedicated resource might not be identical. It might be the case that the execution times will be different when a job is executed on different units of a dedicated resource. It might also be the case that a job can only be executed on some of the units of a particular dedicated resource.

(e) The unit of each shared resource is considered to be uniform. A job will release the portions of shared resources it occupies when its execution on all the units of dedicated resources is completed.

2.2.2 Task Systems

A general task system for a given set of resources with $p \in Z^{+*}$ types of dedicated resources, $m \in Z^{+}$ units of each type (there is no loss in generality in assuming the same number of units in each dedicated resource), and $q \in Z^{+}$ types of shared resources can be defined as the system $(J, \ll, \{T_k\}, \{R_k\}, \{w_k\})$ as follows:

1. $J=\{J_1, J_2, \ldots, J_n\}$ is a set of jobs to be executed.
2. $\ll$ is a (irreflexive) partial order defined on J which specifies operational precedence constraints. That is, $J_i \ll J_j$ signifies that J_j cannot be started before J_i is completed. If $\ll$ is empty, then the jobs in the task system are said to be *independent*.
3. $T_k=[t_{ij}]$, $1 \leq k \leq n$, is a $(p \times m)$ matrix of execution times, where $0 \leq t_{ij} \leq \infty$ is the time required to execute a particular task J_k on the j^{th} unit of the i^{th} dedicated resource. Note that if $t_{ij}=\infty$ then the job J_k cannot be executed on the j^{th} unit of the i^{th} dedicated resource *and* that for each i there exists at least one j such that $t_{ij}<\infty$.
4. $R_k=[R_1(J_k), R_2(J_k), \ldots, R_q(J_k)]$, $1 \leq k \leq n$, specifies in the i^{th} component, the amount (fraction) of shared resource type R_i required throughout the execution of J_k. Always we have $0 \leq R_i(J_k) \leq 1$ for all i and k $(1 \leq i \leq q$ and $1 \leq k \leq n)$.
5. The weights w_k, $1 \leq k \leq n$ are interpreted as deferal costs (or more exactly cost rates), which in general may be arbitrary functions of the scheduling properties influencing J_k. However, in most of the cases w_k is taken as constant. Thus, the "cost" of a job J_k finishing at time t is simply $w_k t$.

If $\ll$ is empty or there are no shared resources in the system or $w_k=1$ for all $1 \leq k \leq n$ then, the parameters $\ll, \{R_k\}, \{w_k\}$ will not appear as system parameters.

---

* Z^{+} *represents the set of positive integers.*

The partial order $\lhd$ is represented as a directed acyclic graph (*dag*) with no (redundant) transitive arcs. In general however, the way in which a partial order is specified in a given problem may influence the complexity of its solution.

For a set of resources with one dedicated resource and m identical units, no shared resources and weights $w_k=1$, $1\leq k\leq n$, that can represent a set of identical processors $P=\{P_1,P_2,\ldots,P_m\}$, a *dag* representation is pictured in Fig.2.1. The notation J_k/t_k is introduced for labelling vertices. (Since there exists only one dedicated resource the execution time of job J_k is $t_{ij}=t_j$ and since all units are identical $t_k=t_j$ for all $1\leq j\leq m$, $1\leq k\leq n$).

It is necessary to define a number of terms relating to *dags*. In particular, a *path* of length k from J to J' in a given graph G is a sequence of vertices (tasks) $J_{i_1},J_{i_2},\ldots,J_{i_k}$ such that $J=J_{i_1}$, $J'=J_{i_k}$ $(k\geq 1)$ and $(J_{i_j},J_{i_{j+1}})$ is an arc in G for all $1\leq j\leq k-1$. Moreover, if such a path exists, J will be called a *predecessor* of J' and J' a *successor* of J. If k=2, the terms *immediate predecessor* and *immediate successor* will be used. *Initial* vertices are those with no predecessors, and *terminal* vertices are those with no successors. The graph forms a *forest* if either each vertex has at most one predecessor or each vertex has at most one successor. If a forest has in the first case exactly one vertex with no predecessor or in the second case exactly one vertex with no successor, then it is called a *tree*. The *level* of a vertex J is the sum of the execution times associated with the vertices in a path from J to a terminal vertex such that this sum is maximal. Such a path is called a *critical path* if the vertex J is at the highest level in the graph.

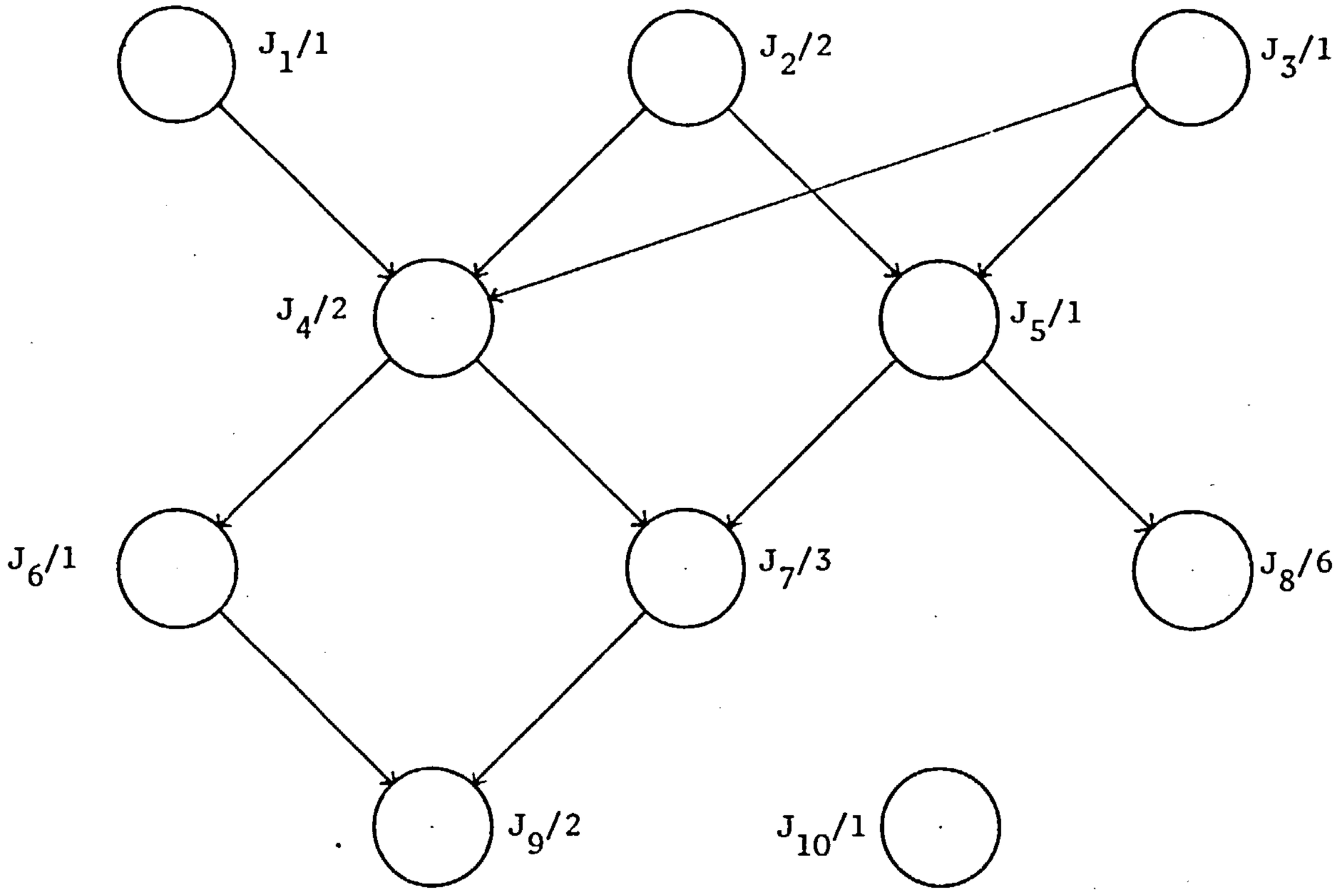


FIGURE 2.1: A *dag* representation of $(J,<,\{t_k\})$

Notations and properties:

1. Acyclic
2. No transitive edges: (J_1,J_6) would be such an edge.
3. J_1,J_2,J_3,J_{10} are initial vertices; J_8,J_9,J_{10} are terminal vertices.
4. For example, J_7 is a successor of J_1,J_2,J_3,J_4,J_5 but an immediate successor of only J_4,J_5; J_5 is a predecessor of J_7,J_8,J_9 but an immediate predecessor of only J_7,J_8.
5. Levels:

J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
8	9	8	7	7	3	5	6	2	1

6. Critical paths: J_2,J_5,J_8 and J_2,J_4,J_7,J_9.

2.3 BASIC DEFINITIONS

A number of useful terms concerning deterministic computer scheduling are defined in this section.

Although, the term scheduling has been previously defined, more precisely, by *scheduling* a set of jobs on a computing system it means to assign to each job, within certain time interval(s), resources that are needed for its execution with the constraint that all the resources needed for the execution of a job are assigned to the job simultaneously. Such an assignment of resources to jobs is called *schedule(S)*.

An explicit way to describe a schedule, when there are no shared resources (q=0), is a *timing diagram*, which is also known as the *Gantt chart*. As an example, the timing diagram for the execution of the task system shown in Fig.2.1 on three identical processors computing system is pictured in Fig.2.2. The specific processors are shown along the vertical axis and a time scale is shown along the horizontal axis. The shading shown in the figure represents periods in which the processors are *idle*.

In a schedule, a processor might be left idle either because there is no executable task at that time or because it is an intentional choice. (A task is said to be *executable* at a certain time instant if the execution of all its predecessors has been completed at that time). Clearly, it is neither necessary nor beneficial in a schedule to have all the processors idle at the same time. For a given schedule, an *idle period* of a processor is defined to be the time during which a processor is not executing any job (while at least one of the other processors is executing some job). The symbol $\emptyset$, appropriately subscribed when necessary, is used to denote such idle periods. Also, the symbol D denotes the timing diagrams. The symbols s_i and f_i denote, respectively, the start and the finishing times of job J_i. Where necessary for indicating the dependence on a particular schedule S, the notation $s_i(S)$ and $f_i(S)$ is used.

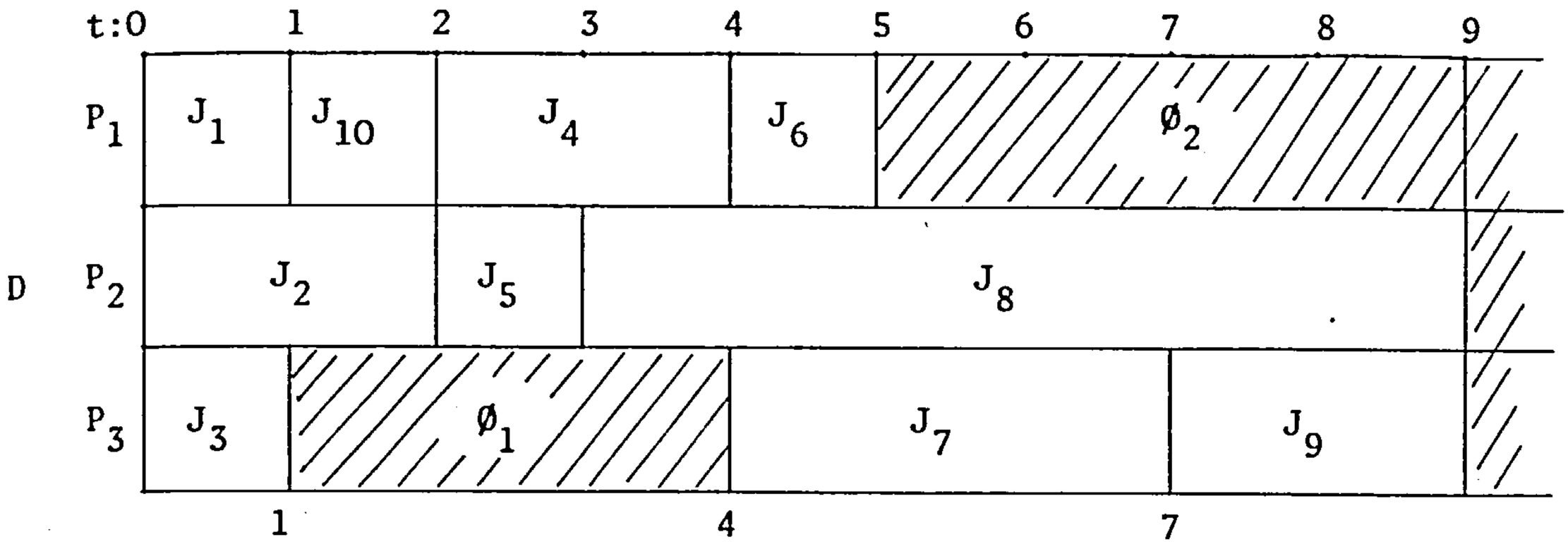


FIGURE 2.2: Example timing diagram for Fig.2.1 (m=3)

Furthermore, a *scheduling algorithm* is a procedure that produces a schedule for every given task system. A scheduling algorithm is said to be *non-preemptive* if it follows the rule that once the execution of a task has begun, it must continue without interruption until completion. On the other hand, a *preemptive* scheduling algorithm is the one that permits the execution of a job to be interrupted and removed from the processor, subject to the condition that the interrupted job is restarted later from the point at which it was last interrupted. In schedules that are produced by non-preemptive scheduling algorithms there is exactly one execution interval for each task, while in those produced by a preemptive one, there might exist more than one non-overlapping execution interval for each task.

2.4 PERFORMANCE CRITERIA

Different criteria can be used to measure the performance of a schedule produced by a specific scheduling algorithm. The most commonly used performance criteria are going to be considered in the following discussion.

A great amount of research has been done as far as the completion (or maximum finishing) time criterion is concerned. The *completion time* $\omega(S)$ (or simply ω) of a given schedule S is the total time it takes to complete the execution of all jobs of the task system according to schedule S. Assuming that the starting time of a schedule is zero, then the completion time can be represented symbolically as:

$$\omega(S) = \max_{1 \leq i \leq n} \{f_i(S)\},$$

where n is the number of jobs in the task system. It can be seen in Fig.2.2 that ω=9 units of time.

This criterion gives an insight about the utilisation of the resources and consequently about the throughput of the system. This is because shorter completion time means that the resources have been utilised better and therefore, the number of jobs processed per unit of time (throughput) will be greater.

Another performance criterion with less research involvement, but not of least importance for the computing systems, is the mean flow time. The *mean flow time* $\bar{\omega}(S)$ (or simply $\bar{\omega}$) of a given schedule is defined to be the sum of finishing times of all the jobs divided by the number of jobs (n) in the task system. Assuming again that the starting time of a schedule is zero then the mean flow time can be expressed as:

$$\bar{\omega}(S) = \frac{1}{n} \sum_{i=1}^{n} f_i(S).$$

Referring to the schedule in Fig.2.2, it can be found that $\bar{\omega}=43/9$ units of time. 43/10

This criterion provides a measure for the average time a task spends

in the computing system (turnaround time). Therefore, it is an indirect measure for the system's throughput. Because the shorter the time during which a task occupies certain resources (other than processors), the greater the amount of time that is available for other tasks to occupy those resources.

Another reason for the importance of this criterion lies in its connection with the *mean number $\bar{n}$ of incomplete tasks* over the schedule-length (completion time of the schedule), which can be a performance criterion by itself. It has been found (see [CD],[Co]) that

$$\frac{\bar{n}}{n} = \frac{\bar{\omega}}{\omega} .$$

Thus, the mean number of incomplete tasks is in the same ratio to the maximum number of tasks as the mean flow time to the completion time of a schedule. Also, the above equation indicates that, for a given task system and completion time of a schedule, the mean flow time is directly proportional to the mean number of incomplete tasks.

Once a performance criterion has been chosen, then a schedule can be characterised as an optimal or non-optimal schedule according to the chosen performance criterion. More exactly, a schedule S is said to be *optimal* with respect to a certain criterion of performance if it minimises the chosen performance index, and *non-optimal* if it does not minimise it. Thus, if the completion time is used as the performance criterion for a schedule, then an optimal schedule is one which has the shortest completion time. A non-optimal schedule is believed to be a good schedule if its completion time is close to the completion time of the optimal schedule.

2.5 SCHEDULING ALGORITHMS

As mentioned earlier a scheduling algorithm is a procedure that produces schedule for every given task system. Therefore, because of the generality they have, it becomes of great value to study the computational complexity of such algorithms, their optimality or non-optimality and also, as far as the non-optimal ones are concerned, to study their behaviour against the optimal one. These subjects will be analysed briefly in the following subsections.

2.5.1 Complexity of Scheduling Algorithms

In general, the *complexity* of an algorithm solving a given problem refers only to its execution time, expressed as a function of input-length; i.e. the number of elementary steps needed to describe an instance of the problem. Here, complexity is specified as a function of the basic problem parameters, primarily the number n of tasks. In some cases, this is a considerable simplification, but not an inappropriate one for this study. (In effect, it is assumed that numbers can be read and operated on in a constant time, which is reasonable in practice).

In order to represent the complexity of an algorithm, the order-of-magnitude notation O(.) will be used, which concentrates on the terms of a function that dominates its behaviour. Thus, if it is written that an algorithm has complexity $O(n^2)$, it simply means that there exists a constant c such that the function cn^2 bounds the execution time as a function of n. (For more details see [Kn], p.104-108 and [HoS2], p.24-30)

2.5.2 Optimal Scheduling Algorithms

A scheduling algorithm is optimal with respect to a performance criterion if it produces an optimal schedule for every given task system. Since for a set of n jobs there is only a finite number of schedules, one could derive an algorithm which could find the optimal schedule through an exhaustive examination of all the schedules. Clearly, such an algorithm requires

considerable computation time (most probably exponential), which offsets the advantages gained by the optimal schedules. In effect, an optimal algorithm will be considered efficient only if it requires a reasonable computation time to produce optimal schedules. Such algorithms are those whose complexity is bounded by a polynomial of small degree. Moreover, these optimal algorithms are usually referred to as *polynomial-time-bounded algorithms* or simply *polynomial algorithms*. So far, for the general scheduling problem it has not been found any optimal algorithm with polynomial complexity. Instead, it has been shown that it falls in the category of what is known as *non-deterministic-polynomial-time-hard* or simply NP-hard[†] problems.

Roughly speaking, an optimisation or a decision problem is said to be *NP-hard* if given a deterministic polynomial algorithm for the problem, it is possible to use that algorithm to obtain a deterministic polynomial algorithm for every problem in the class of NP-hard. Further, a decision problem is said to be *NP-complete*[†] if it is NP-hard and if there exists a non-deterministic algorithm to solve the problem in polynomial time. So, either there exists polynomial algorithms for all NP-hard problems, or none of them has a polynomial algorithm. Although a great deal of time and effort have been spent in finding a polynomial algorithm for any of the NP-hard problems, so far no such algorithm has been found. However, in spite of the overwhelming empirical evidence to the contrary, it is still an open question whether NP-hard problems can be solved in polynomial time.

It has been shown that the decision scheduling problem is a NP-complete one, even for the following computation models:

- all jobs in the given task system have equal execution time and the number of identical processors in the system is arbitrary;
- there are only two identical processors in the system and the execution time of each job in the task system is either one or two

---

[†] *A more exact definition is given in* [HoS2], *Chapter 11.*

units of time, when the completion time is the performance criterion (see [U1]); and

- there are only two identical processors in the system and the execution time and weight of each job in the task system are arbitrary, when the mean flow time is the performance criterion (see [BCS1]).

This means that the problem of designing optimal scheduling algorithms even for the above mentioned simple computation models is NP-hard and therefore intractable.

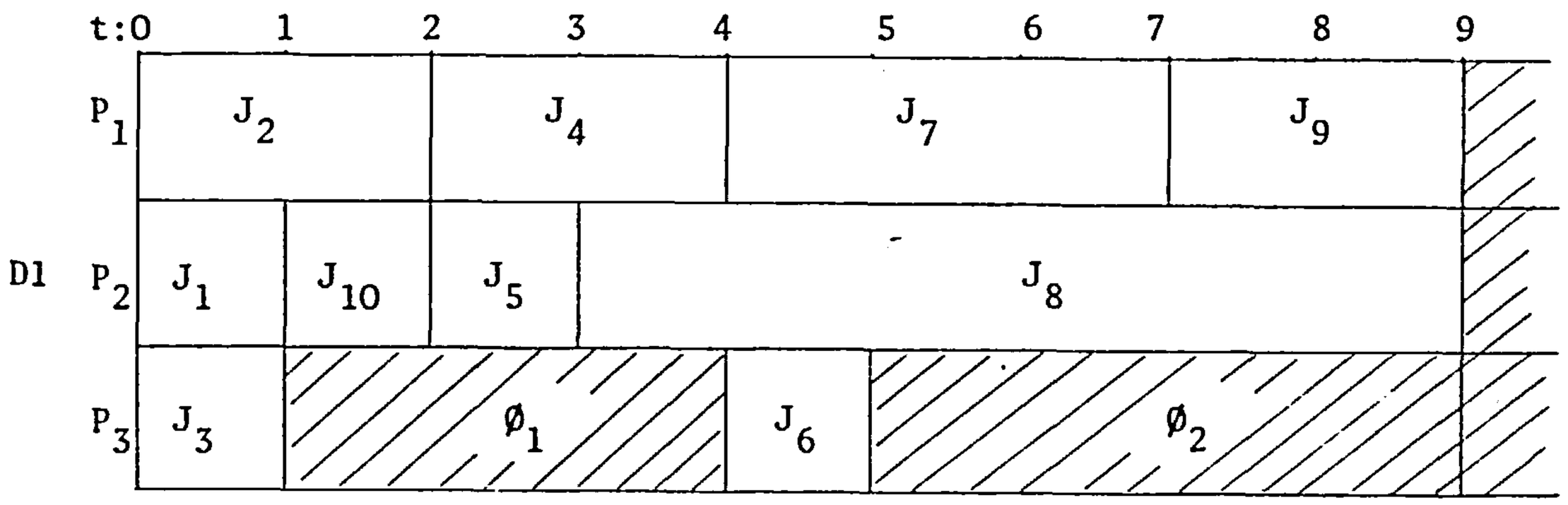
2.5.3 Heuristic Scheduling Algorithms

We saw in the previous section, that there is little hope in finding an efficient optimal scheduling algorithm even for simple submodels of the general model, which has been considered for this study. So, one comes to the decision to use approximate heuristic scheduling algorithms, which produce near optimal schedules, rather than search for optimal ones. Usually, *heuristic scheduling algorithms* are easy to understand, of low complexity and easy to implement. In other words, they hopefully produce good schedules in a reasonable amount of time, without giving a lot of trouble to the job-scheduler designer.

A class of heuristic scheduling algorithms is the so called *demand scheduling algorithms* class. Such algorithms have the restriction that a processor is never left idle intentionally. That is, a processor is left idle if and only if there is no executable job within that period. (However, it should be noticed that it is sometimes necessary to leave a processor idle intentionally in order to have better completion time for a given task system). A demand scheduling algorithm can be specified by merely giving the rules on how jobs have to be assigned on processors, or how jobs are to be chosen for execution at any instant when one or more processors are free. (Of course,

the choice is only amongst jobs that are executable at that time).

A subclass of demand scheduling algorithms is the *priority driven (P.D.) scheduling algorithms*. According to a P.D. scheduling algorithm, all jobs in the task system are assigned priorities and jobs with higher priorities are executed instead of jobs with lower priorities when they are competing for processors. Rules are also provided for breaking ties. In this type of algorithms, one just lists the jobs in J in descending order of their priorities from left to right. Such a list is called a *priority list* and will be noted by the L symbol. When a processor becomes free, the priority list is scanned from left to right until the first executable job is found; that is, the job can be executed on the given processor, if all predecessors of J have been completed and sufficient shared resources exist to satisfy $R_i(J)$ for each $1 \leq i \leq q$. Then, this job is assigned to the free processor. (If two or more processors are available at the same time, a rule is also specified as to which of the processors will be assigned which job.) Such a scheduling algorithm is also referred to as a *list scheduling*. As an example one can apply higher priorities to the jobs which are in higher level in the task system given in Fig.2.1. Then, the following priority list can be obtained $L=(J_2,J_1,J_3,J_4,J_5,J_8,J_7,J_6,J_9,J_{10})$. Therefore, for that priority list a priority-driven scheduling algorithm can give the schedule D1 shown in Fig.2.3.



$L=(J_2,J_1,J_3,J_4,J_5,J_8,J_7,J_6,J_9,J_{10})$

FIGURE 2.3: Example schedule for Priority-Driven algorithms

When two or more processors were available, in the above example, the simple rule of assigning a higher priority job to a processor with lower index was used. There are other subclasses of demand scheduling algorithms as well. One of them is the *earliest completion time (E.C.T.) algorithms*. According to these algorithms when a task is being considered for assignment to a processor, it is assigned to that processor on which its finishing time will be earliest. Priority lists can also be formed before such an algorithm is activated. The tasks in the priority list are considered one by one in the order they appear in the list. Ties are broken arbitrarily or by a specified rule. An E.C.T. algorithm, applied to the same priority list $L=(J_2,J_1,J_3,J_4,J_5,J_8,J_7,J_6,J_9,J_{10})$, as in the previous example, for the task system given in Fig.2.1, gives the schedule D2 as shown in Fig.2.4. In this example, ties are broken by assigning the task to the processor with the least index.

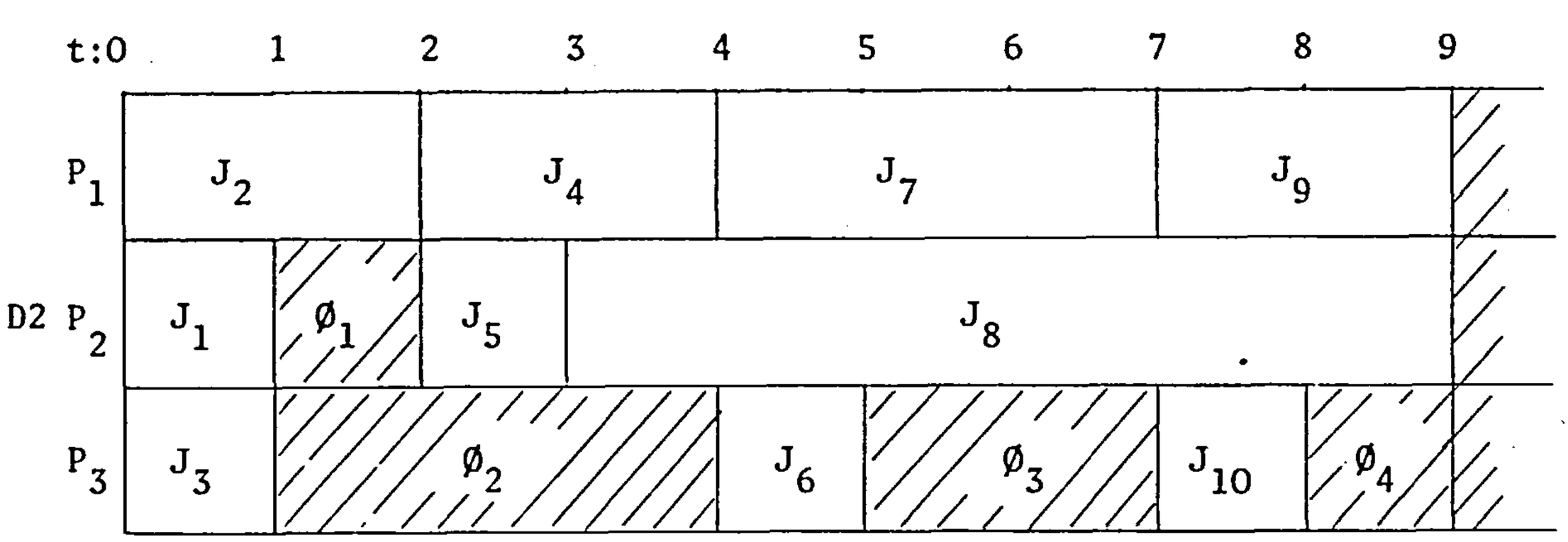


FIGURE 2.4: Example schedule for E.C.T. algorithms

Another subclass, similar to the one just described, is the so called class of *quick and dirty (Q.A.D.) scheduling algorithms*. According to the Q.A.D. algorithms a task is assigned to that processor on which its contribution to the mean flow time is the least possible. In contrast to the

previously mentioned algorithms these are right justified (i.e., a job J_i scheduled on the i^{th} processor before the J_{j+k} job will be executed on that processor after J_{j+k}).

As an example, consider the task system defined by the set of independent jobs $J=\{J_1,J_2,J_3,J_4,J_5\}$ to be run on a three non-identical processor system with no shared resources. Let the input matrix $[t_{ij}]$ be:

$$\begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \end{array} \begin{array}{c} \begin{array}{ccccc} J_1 & J_2 & J_3 & J_4 & J_5 \end{array} \\ \begin{bmatrix} 12 & 2 & 16 & 8 & 8 \\ 15 & 2.5 & 20 & 10 & 15 \\ 6 & 1 & 8 & 4 & 6 \end{bmatrix} \end{array}$$

Then, applying the Q.A.D. algorithm we get the schedule D3 as shown in Fig.2.5.

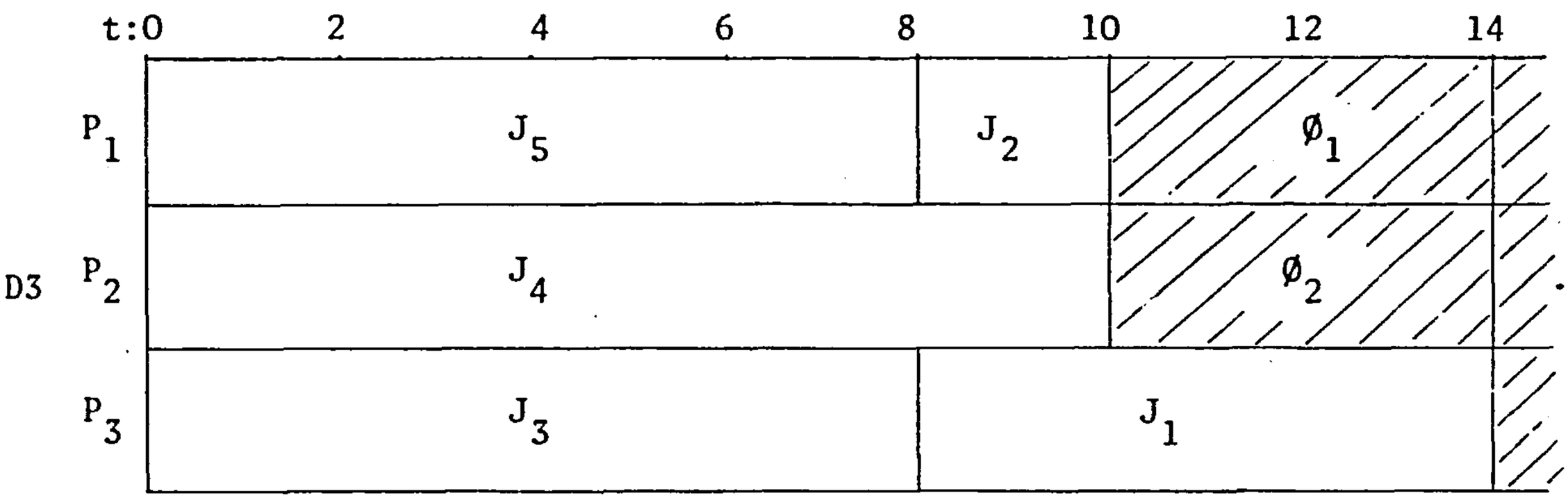


FIGURE 2.5: Example schedule for Q.A.D. algorithm

Furthermore, there are the *look ahead algorithms* (or two dimensional i.e. 2D). In such algorithms an optimistic guess for the schedule length is always made at the beginning. Then by working across the schedule from left to right and top to bottom, an attempt is made to form a schedule of that length. If a schedule is actually produced, then the process stops with a

minimal final completion time. Otherwise the optimistic guess is increased by a unit of time and the process is repeated. Such algorithms are most applicable to task systems with independent jobs. As an example, consider the task system defined by the set of independent jobs $J=\{J_1,J_2,J_3,J_4,J_5\}$ with execution time requirements $[t_{ij}]=[6,8,4,6,1]$ (t_j corresponds to J_j), to be scheduled on a multiprocessor model with three identical processors. Making the guess for the schedule length to be 8 units of time we cannot produce a final schedule (Fig.2.6,D4(a)). Then, increasing the guess length by one unit we can see (Fig.2.6,D4(b)) that a final schedule is produced.

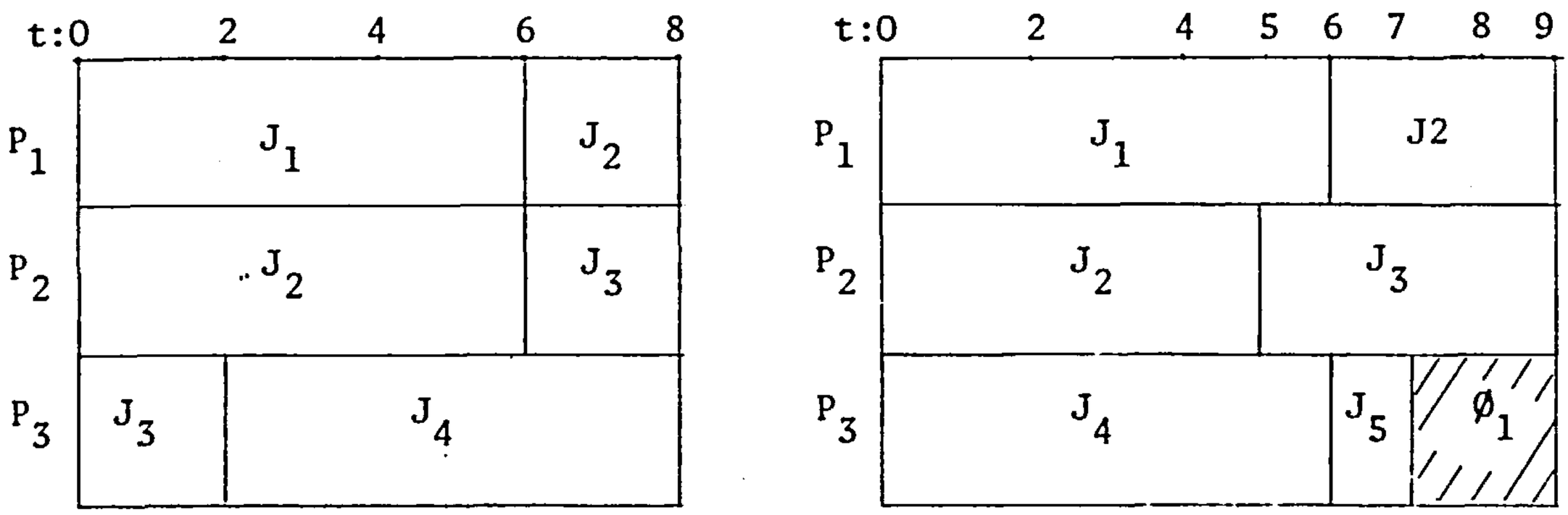


FIGURE 2.6: Schedules illustrating the look ahead algorithms

Finally, if jobs in a task system have been allocated to processors according to a scheduling algorithm and consequently have been sequenced in a STF order on each processor, then such a procedure characterises the algorithms known as *two-phase algorithms*. The objective of such algorithms is to improve the mean flow time of a task system without changing its completion time. So, a two-phase Q.A.D. algorithm (Q.A.D.*) for the same task system, used in schedule D3, will produce the schedule D5 as shown in Fig.2.7.

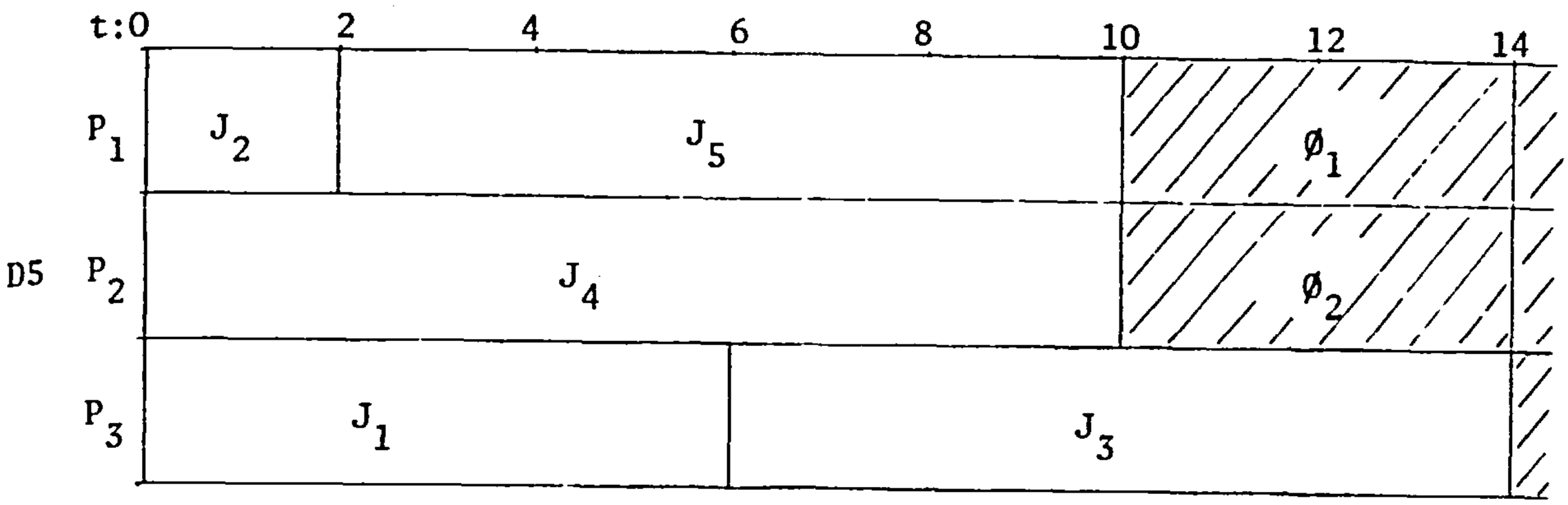


FIGURE 2.7: A schedule illustrating the two-phase algorithms

2.5.4 Evaluation of Scheduling Algorithms

In case a scheduling algorithm is not optimal, one would naturally like to know how effective it is. The deterministic scheduling theory measures the effectiveness of an algorithm by the worst schedules it produces. So, the worst-case performance of the algorithm is determined.

More exactly, the *worst-case performance* or *the extreme performance* of an algorithm is evaluated by comparing the worst relative to the theoretical optimum value of the schedules, which can be constructed, for a particular performance criterion. Most of the times, the worst value is upper bounded and the theoretical optimum value is lower bounded by expressions which contain relevant system parameters. Then, the ratio of the worst over the optimum value will result in upper bounds, which characterise the extreme performance of the algorithm. Actually, we refer to these bounds as *worst-case performance bounds* or *guaranteed performance levels*. Moreover, if examples can be constructed that cause the algorithm to deviate from optimal performance by the amount allowed by a proven worst-case bound (or asymptotically approach it), then that bound is clearly the best bound provable (i.e., best possible bound) and one may say that the worst-case performance has been determined.

CHAPTER 3

A SURVEY OF PREVIOUS WORK

3.1 INTRODUCTION

A great deal of research has been done during the past, and especially during the last decade as far as the deterministic scheduling of computing systems is concerned. So, before the model, which this thesis is concerned with, is described and the scheduling algorithms analysed, it would be worth while to answer some questions like:

- has the general model described in section 2.2 been fully examined?
- if not, which simplified versions have been studied?
- what scheduling algorithms have been analysed?
- what were their behaviour?
- in the case where scheduling algorithms are not optimal, what worst-case bounds have been found and what is the average performance behaviour?

As a matter of fact, this is the purpose of the present chapter.

The survey begins with a general description of the models examined so far, followed by the known optimal scheduling algorithms and NP-complete problems, the worst-case bounds of non-optimal algorithms and the average performance behaviour, where indicated.

3.2 MULTIPROCESSOR COMPUTING MODELS

Unfortunately, the general model described in section 2.2 has not yet been fully examined. All the computing models, which have been considered so far, consist of one type of dedicated resources (processors). For this reason such models are known as multiprocessor computing models. A multiprocessor model is a homogeneous one if all the processors in the model are identical. If there are different speed (uniform) processors or generally non-identical ones, then the multiprocessor system is called heterogeneous.

Apart from the above restriction to the dedicated resources of the model, constraints have also been made for the task system. An arbitrary task system has never been considered for scheduling. Always a simplified version characterised the task system.

We are not going to give any further description of the exact models examined at this stage. This knowledge will be accumulated as one follows the remaining sections of this chapter.

3.3 OPTIMAL SCHEDULING ALGORITHMS AND NP-COMPLETE PROBLEMS

It has been discussed in section 2.5 that algorithms which produce optimal schedules are of significant importance. Although there are such algorithms, these are applicable only for restricted versions of the general model.

This section provides a reference of the known optimal algorithms so far, as well as for the scheduling problems which have proved to be NP-complete. Moreover, a brief description of some of the algorithms is given. The optimal algorithms have been considered with respect to completion and mean flow time performance criteria respectively.

3.3.1 Optimal Scheduling Algorithms and NP-Complete Problems with Respect to Completion Time Performance Criterion

The homogeneous multiprocessor computing models are considered first. Actually, there are non-preemptive and preemptive polynomial time optimal algorithms, with respect to completion time performance criterion, for homogeneous multiprocessor models. Moreover, such non-preemptive algorithms are known only for the following three cases:

(1) a task system, that consists of unit execution time jobs with a forest being their partial order, scheduled on an arbitrary number of identical processors;

(2) a task system, that consists of unit execution time jobs with an arbitrary *dag* being their partial order, scheduled on two identical processors; and

(3) a task system, that consists of independent unit execution time jobs and arbitrary shared resource requirements, scheduled on two identical processors.

Hu [Hu] has found a list scheduling algorithm for case (1), when a tree charactetises the jobs' partial order. Higher priorities are assigned to

jobs of higher level. (Assignment of priorities to jobs of the same level is arbitrary.) The level of a job is decided by the following rules:-

(i) the level of a job that has no successors is defined to be one;

(ii) the level of a job that has one or more successors is equal to the maximum value of the levels of its successors increased by one.

Chen and Liu [CLi] have considered the same problem. Actually, they have proved that Hu's algorithm is optimal, when either a tree or a forest is considered as the jobs' partial order.

Fujii, Kasami and Ninomiya [FKN], Coffman and Graham [CG] and Sethi [Se2] have discovered different scheduling algorithms for case (2). In [FKN] an undirected graph G' is constructed from a task system with arbitrary *dag* G. An edge (J,J') in G' denotes the fact that J$\ll$J' and J'$\ll$J are both false. A schedule on two processors is then determined from a maximal matching of G'. Both the construction of G' and the maximal matching are of $O(n^3)$ time complexity. Later, Coffman and Graham gave a list scheduling algorithm to execute tasks, level by level as Hu's algorithm for case (1), but when there is more than one task at the highest level, it makes a judicious choice which task to execute first. More exactly, in Coffman-Graham's (C.G.) algorithm, the priorities are assigned to jobs as follows:-

(i) starting with 1, which is the lowest priority, distinct and constructive priorities are arbitrarily assigned to jobs that have no successors;

(ii) priorities are assigned to jobs with one or more successors recursively:

(a) Let S be the set of jobs with unassigned priorities and all their successors have assigned priorities. Then, label each job in S using the priorities of its successors $(i_1,i_2,\ldots)$, which should be in increasing order;

(b) Find the smallest labelled job, according to lexicographical order rule, and assign to this job the lowest unassigned priority.

While C.G.'s algorithm needs $O(n^2)$ steps to determine the priorities of the jobs and $O(n^2)$ steps to construct the schedule, Sethi [Se2] gave an algorithm with O(n+e) and O(na(n)+e) time complexity respectively, where e is the number of edges in a *dag* and a(n) is a functional inverse of the Ackermann's† function. $(a(n)=\min\{i:A(i,3)<\lfloor \log_2 n \rfloor\})$.

For the case (3) Garey and Johnson [GJ1] gave an algorithm where, an n-node graph G is constructed, having each node labelled by a distinct task, with an edge joining J_i to J_j if and only if $R_k(J_i)+R(J_j)\leq 1$ for all $1\leq k\leq q$. Thus, task J_i and J_j can be executed simultaneously if and only if there is an edge joining the corresponding nodes. Then, a maximal matching algorithm is applied, in order to construct the optimal schedule.

The inability to discover efficient non-preemptive algorithms to produce optimal schedules for more complex versions of the general model has been a rather frustrating experience for quite a number of years. However, such frustration is at least partially pacified by some recent results in Complexity Theory, as mentioned in section 2.5.2. But apart from the two problems Ulman [U1] found to be NP-complete, other ones have also been found, to be in the class of NP-complete problems, by other authors.

Table 3.1 summarises these problems together with the optimal scheduling algorithms, which have already been discussed, though some remarks will be made for a better interpretation of the table.

The columns correspond to the possible parameters of an algorithm that

---

†*Ackermann's function is defined by:*

$$A(0,j) = 2j \quad \textit{and}$$

for $i\geq 1,\ j\geq 2$

$$A(i,0) = 0, \quad A(i,1)=2$$

$$A(i,j) = A(i-1,A(i,j-1))$$

solves a problem defined by the assumptions given in a row of the table. For those entries in which a value is specified, the free parameter is eliminated. For example, in problem 3, one finds that m is not a parameter but it is fixed at the value m=2. Similarly, $\{t_j\}$ is not a parameter, because the specific common value $\{t_j\}$ does not influence the algorithm. But, the partial order is a free parameter. Also, the number of jobs is a free parameter.

The NP-complete problems indicated in Table 3.1 represent the simplest known cases for NP-completeness. Therefore, one should make appropriate inferences regarding more general problems. If one generalises any of the parameter restrictions in a given problem, obviously a problem will be produced which will be at least as hard as the original one. An important observation in this respect concerns a comparison of rows such as 8 and 6. Since problem 8 is NP-complete, it is easy to see that the problem with m as a free parameter is also NP-complete. However, the converse is not necessarily true. Problem 6 is a NP-complete one, but it is not known whether for any fixed value $m \geq 3$ the corresponding problem is NP-complete and so it is regarded as an *open* problem.

Finally, regarding polynomial-time algorithms, we do not in all cases claim that the complexity shown is minimal.

TABLE 3.1: Optimal non-preemptive scheduling algorithms - NP-complete problems - Completion time - Homogeneous multiprocessor models. $\{w_k\}=1$. Parameters *n and*

	No. of Processors m	No. of Shared Resources q	Partial Order $<$	Time Requirement $T_k=t_k$	Algorithm Complexity	References
1.	-	0	Forest	Equal	$O(n)$	[Hu], [CLi]
2.	2	0	-	Equal	$O(n^3)$	[FKN]
3.	2	0	-	Equal	$O(n^2)$	[CG]
4.	2	0	-	Equal	$O(na(n)+e)$	[Se2]
5.	Fixed ≥ 3	0	-	Equal	open	
6.	-	0	-	Equal	NP-complete	[U1]
7.	Fixed ≥ 3	0	-	$t_j \in \{1,2\}$	NP-complete	[U1]
8.	2	0	ϕ	-	NP-complete	[BCS1]
9.	2	-	ϕ	Equal	$O(n^3)$	[GJ1]
10.	Fixed ≥ 2	1	Forest	Equal	NP-complete	[GJ1]
11.	Fixed ≥ 3	1	ϕ	Equal	NP-complete	[GJ1]
12.	-	0	ϕ	$\frac{\max_i\{t_i\}}{\min_i\{t_i\}}=p$	NP-complete	[IK]

Notation:

1. - stands for a free parameter;
2. ϕ stands for empty partial order.

We move now from non-preemptive to preemptive scheduling algorithms. Although the general preemptive scheduling problems, without shared resources, is a NP-complete one ([U2], p.159), there are known optimal algorithms for the following four cases:-

(1') a task system, that consists of independent jobs with arbitrary execution time requirements, scheduled on an arbitrary number of identical processors;

(2') a task system, that consists of jobs with arbitrary execution time requirements and an arbitrary *dag* being their partial order, scheduled on two identical processors;

(3') a task system, that consists of jobs with arbitrary execution time requirements and a forest being their partial order, scheduled on an arbitrary number of identical processors;

(4') a task system that consists of independent jobs with arbitrary execution time requirements, scheduled on an arbitrary number of identical processors with independent memories. (The memories are independent in the sense that the information stored in the i^{th} memory can only be accessed by the P_i processor.)

First, McNaughton [McN] has produced an efficient optimal algorithm for case (1') with linear time complexity. Such an algorithm also can be found in Coffman's book ([Sel], pp.76-78). Later, Muntz and Coffman [MC1] have found an optimal algorithm for case (2'). Actually, their algorithm constructs a processor shared schedule which can easily be converted into a preemptive schedule of the same length. Tasks at the same level get the same level service. More exactly, the Muntz-Coffman's (M.C.) algorithm is as follows:

Let s be the time when assignment of processors is made. Initially s=0. Amongst the tasks that are ready to be executed assign one processor each to

the tasks at the highest level. If there is a tie among b tasks (because they are at the same level) for the last a (a<b) processors, then assign a/b of a processor to each of these b tasks. Continue such an assignment until time t, at which one of the following events occurs.

Event 1. A task is completed at t.

Event 2. A task's level has caught up to the level (lower) of another task at t.

In either case set s=t and reassign the processors to the unexecuted portion of the task system. Also, the authors [MC2] have realised that the above algorithm produces optimal schedules for case (3') as well. Finally, for case (4') Kafura and Shen [KS2] found an optimal polynomial time algorithm, based on the idea of McNaughton.

The above results are presented in Table 3.2. The remarks made for Table 3.1 are also valid for Table 3.2.

TABLE 3.2: Optimal preemptive scheduling algorithms - NP-complete problems - Completion time - Homogeneous multiprocessor computing models. q=0, $\{w_k\}$=1. Parameters *n and*

	Number of processors m	Partial Order $\prec$	Time Requirement $T_k=t_k$	Algorithm Complexity	References
1.	-	-	-	NP-complete	[U2]
2.	2	-	-	$O(n^2)$	[MC1]
3.	-	Forest	-	$O(n\log_2 n)$	[MC2]
4.	Fixed ≥ 3	-	-	open	
5.	-	ϕ	-	$O(n)$	[McN]
6.	-*	ϕ	-	$O(n\log_2 n)$	[KS2]

**indicates that each processor is connected with a private memory.*

At this point one can raise the following question. If for a particular problem, a preemptive and a non-preemptive optimal algorithm is known, then which of them is most preferable? That question was answered by Liu [Li1] for a multiprocessor system with identical processors and no shared resources. He shows that if ω is the completion time of an optimal non-preemptive algorithm and ω' the corresponding one of a preemptive algorithm then $\omega' \geq 3\omega/4$, for $m=2$ and $\omega' \geq (m+1)\omega/2m$, for $m \geq 3$. Moreover, he found examples which reach these bounds. In other words, these results suggest, in terms of completion time, that although the optimal preemptive algorithms are preferred, if we know an optimal non-preemptive algorithm it might not be worth a great deal of work and effort to try to design an optimal preemptive one, because the gain will not be significant. Such a comparison between preemptive and non-preemptive algorithms can be found in Coffman's book ([Se1]pp.86-87).

So far, the homogeneous multiprocessor computing models have been considered. Now, we turn to consider the behaviour of optimal algorithms for heterogeneous ones.

Non-preemptive polynomial time optimal algorithms are known only for two processor systems with different speeds and no shared resources in the following two cases:-

(1") a task system consists of equal execution time requirement jobs and a tree being their partial order. The ratio between the speeds of the two processors is 2 (i.e., $b_1/b_2=2$);

(2") a task system consists of independent jobs with equal execution time requirements. The ratio between the speeds of the processors is arbitrary.

Moreover, preemptive polynomial time optimal algorithms are known for models without shared resources in the following four cases:-

(3") a task system, that consists of arbitrary execution time requirement jobs and an arbitrary *dag* being their partial order, scheduled on two uniform processors;

(4") a task system, that consists of independent jobs with arbitrary execution time requirements, scheduled on an arbitrary number of uniform processors;

(5") a task system, that consists of independent jobs with arbitrary execution time requirements, scheduled on a model with one fast processor and the remainder identical;

(6") a task system, that consists of independent jobs with arbitrary execution time requirements, scheduled on two non-identical processors, but each job can be executed on both processors.

Optimal algorithms for cases (1") and (2") have been found by Baer [Ba]. In case (1"), he adapted Hu's algorithm to construct an optimal algorithm, while in case (2") he presented a simple optimal algorithm with linear time complexity. More or less at the same time, Liu and Yang [LY] presented an optimal preemptive algorithm for case (5") and, Schindler and Simonsmeir [SS] another one for case (6"). But very recently, Horvath, Lam and Sethi [HLS] have adapted the Coffman-Muntz algorithm for multiprocessor models with different speed processors. Actually, that algorithm proved to construct optimal schedules for case (3") and (4").

Table 3.3 summarises the above discussed algorithms. The remarks made for Table 3.1 are valid for Table 3.3 as well.

TABLE 3.3: Optimal scheduling algorithms - Completion time - Heterogeneous multiprocessor computing models. q=0, $\{w_k\}$=1. Parameters *n and*

	Type of Processors	No. of Processor m	Time Requirement $T_k=\{t_j\}$	Partial Order $<$	Rule	Algorithm Complexity	References
1.	Uniform($\frac{b_1}{b_2}=2$)	2	Equal	tree	Nonpr.	O(n)	[Ba]
2.	Uniform	2	Equal	ϕ	Nonpr.	O(n)	[Ba]
3.	One fast and the rest ident.	-	-	ϕ	Pr.	$O(n\log_2 n)$	[LY]
4.	Non-ident.	2	-	ϕ	Pr.	O(n)	[SS]
5.	Uniform	2	-	-	Pr.	$O(n^2)$	[HLS]
6.	Uniform	-	-	ϕ	Pr.	$O(mn^2)$	[HLS]

Notation:

1. Nonpr. - corresponds to non-preemptive algorithms;
2. Pr. - corresponds to preemptive algorithms.

3.3.2 Optimal Scheduling Algorithms and NP-Complete Problems with Respect to Mean Flow Time Performance Criterion

The problem of constructing optimal scheduling algorithms with respect to mean flow time criterion appears to be more difficult than the corresponding one where the completion time is considered as the performance index. As a matter of fact, optimal non-preemptive algorithms are known only if the task system consists of independent jobs, equally weighted and there are no shared resource requirements. Such algorithms exist for homogeneous and heterogeneous multiprocessor models.

Conway, Maxwell and Miller ([CMM],pp.74-79) have considered homogeneous models and they have found that a list scheduling algorithm, with higher priorities assigned to jobs with less execution time requirement, constructs schedules that minimise the mean flow time of the jobs in the task system. In order to show the optimality the authors base their proof on the observation that, the mean flow time of a task system with n jobs can be written as $\bar{\omega}=\sum_{i=1}^{n} k_i t_{ij}/n$, where k_i is one greater than the number of tasks following J_i on the j^{th} processor and the fact that, if $a_1 \geq a_2 \geq \ldots \geq a_n$ and $b_1, b_2, \ldots, b_n$ are two sequences of numbers, a permutation $\alpha=(\alpha_1,\alpha_2,\ldots,\alpha_n)$ minimises the quantity $\sum_{i=1}^{n} a_i b_{\alpha_i}$ if and only if $b_{\alpha_1} \leq b_{\alpha_2} \leq \ldots \leq b_{\alpha_n}$.

Also, they have shown, based on the same observation and fact, that an algorithm can be found to minimise the mean flow time when uniform processors characterise the heterogeneous models. A more precise implementation of their idea has been given by Horowitz and Sethi [HoS1].

Finally, Bruno, Coffman and Sethi [BSC2] have found an optimal algorithm when non-identical processors characterise the heterogeneous models. They reduce the scheduling problem to an equivalent minimal-cost flow problem and then they propose an efficient algorithm that produces optimal schedules. A detailed description of the algorithm is given by Bruno [Br1].

No optimal algorithms are known when a partial order exists between the jobs of a task system. Very recently, Sethi proved that the scheduling problem of a task system with arbitrary execution time requirement jobs, equally weighted and a tree being their partial order is a NP-complete problem. On the other hand, Bruno, Coffman and Sethi [BCS1] have shown that the same problem but for task systems with independent jobs of different weights is also a NP-complete one. (See also Sahni [Sa].)

These results are contained in Table 3.4. The comments describing Table 3.1 also apply to Table 3.4.

TABLE 3.4: Optimal non-preemptive scheduling algorithms - NP-complete problems - Mean flow time - Multiprocessor computing models. q=0. Parameters $n, T_k=\{t_j\}$ and

Type of Processors	No. of Processors m	Weights $\{w_k\}$	Partial Order $<$	Algorithm Complexity	References
1. Identical	-	Equal	ϕ	$O(n\log_2 n)$	[CMM]
2. Uniform	-	Equal	ϕ	$O(n\log_2 n)$	[CMM],[HoS1]
3. Non-ident.	-	Equal	ϕ	$O(\max\{mn^2, n^3\})$	[BCS2],[Br1]
4. Identical	Fixed $m\geq 2$	Equal	tree	NP-complete	[Se3]
5. Identical	Fixed $m\geq 2$	-	ϕ	NP-complete	[BCS1],[Sa]

3.4 WORST-CASE BOUNDS FOR HEURISTIC SCHEDULING ALGORITHMS

In view of the difficulty in designing optimal algorithms for arbitrary task systems and the discovery of the classes of NP-hard and NP-complete problems, it is rather clear that heuristics and efficient enumerative procedures are very likely to play the main role in the progress of computer scheduling theory.

Initially, performance bounds are derived for arbitrary (unstructured or random) sequencing rules. The known worst-case bounds for homogeneous and heterogeneous models, under the completion time performance criterion appear in Table 3.5. All the bounds in Table 3.5, as well as in the following tables, are best possible unless otherwise stated. The objectives for deriving such bounds is to demonstrate the importance of developing efficient heuristics and to gain insight into the combinatorial structure of the problem.

TABLE 3.5: Worst-case bounds of arbitrary list scheduling algorithms - Completion time. $w_k=1$. Parameters *n and*

Type of Processors	No. of Processors m	No. of Shared Resources q	Partial Order $\prec$	Time Requirement $T_k=\{t_j\}$	Bounds	References
1. Identical	-	0	-	-	$\frac{\omega}{\omega_0} \leq 2-\frac{1}{m}$	[Gr1]
2. Identical	-	0	ϕ	-	$\frac{\omega}{\omega_0} \leq 1+(m-1)\frac{\max_i\{t_i\}}{\sum_{i=1}^{n} t_i}$	[Gr3]
3. Uniform	-	0	ϕ	-	$\frac{\omega}{\omega_0} \leq \frac{b_1}{b_\ell}+1-\frac{b_1}{\sum_{i=1}^{\ell} m_i b_i}$	[LiL1]
4. Identical with indepen. memories	-	0	ϕ	-	$\frac{\omega}{\omega_0} \leq 1+\log_2(m)$	[KS2],[Kaf]
5. "	-	0	-	-	$\frac{\omega}{\omega_0} \leq m$	[KS2]
6. Identical	-	1	-	-	$\frac{\omega}{\omega_0} \leq m$	[GG1],[GG2]
7. "	$m \geq n$	-	ϕ	-	$\frac{\omega}{\omega_0} \leq q+1$	[GG1],[GG2]

TABLE 3.5: (Continued)

	Type of Processor	No. of Processors m	No. of Shared Resources q	Partial Order $\prec$	Time Requirement $T_k=\{t_j\}$	Bounds	References
8.	Identical	$m\geq 2$	-	ϕ	-	$\frac{\omega}{\omega_0}\leq \min\left\{\frac{m+1}{2}, q+2-\frac{2q+1}{m}\right\}$	[GG2]
9.	"	$m\geq n$	-	ϕ	Equal	$\omega\leq (q+\frac{7}{10})\omega_0+\frac{5}{2}$	[GGJY]
10.	"	$m\geq n$	-	-	Equal	$\frac{\omega}{\omega_0}\leq \frac{q}{2}\omega_0+\frac{q}{2}+1$	[GGJY]
11.	"	-	1	ϕ	Equal	$\frac{\omega-2}{\omega_0}<\frac{27}{10}-\frac{24}{10m}$	[KSS2],[Kr]

Notation:

1. ω is the length schedule of the arbitrary list
2. ω_0 is the length of the optimal list schedule
3. $m=m_1+m_2+\ldots+m_\ell$, where ℓ is the number of groups with different processor speeds and m_i the number of processors of speed b_i, $1\leq i\leq \ell$.

A number of unexpected anomalies have been reported by Graham ([Gr1], [Gr2],[Gr3]) for the problem of minimising the schedule length in a multiprocessor model with no shared resources, identical processors and m, t_k, and $\prec$ arbitrary, when list scheduling algorithms are used. The anomalies show that individual decreases in jobs' execution time requirements, increases in the number of processors and removal of precedence constraints can in fact lengthen the schedules. An analysis is provided by the same author to demonstrate that the new schedule length resulting from the changes can be no more than $1+(m-1)/m'$ times larger than the original, where $m' \geq m$ is the new number of processors. Note that problem 1 in Table 3.5 follows from the bound just mentioned when $m=m'$.

From the bounds stated in Table 3.5 some observations can be made. For homogeneous multiprocessor models with no shared resources and m, t_k and $\prec$ arbitrary, a schedule produced by a list scheduling algorithm, with priorities assigned arbitrarily to the jobs, is never worst than an optimal schedule by 100%. For the same problem, but with independent jobs in the task system, the bound in the second row indicates that if none of the execution times is large compared to the sum of all execution times then the schedule length of an arbitrary list scheduling algorithm is not far from the optimal schedule length. In problem 3, where uniform processors characterise the heterogeneous models and for the same task system parameters as in problem 2, the bound implies that the ratio of the maximum and minimum speed of the processors mainly characterise the performance of such models. This is apparent if $\sum_{i=1}^{\ell} m_i b_i$ is very large. When homogeneous multiprocessor models with dedicated private memories are considered, the bounds in the 4^{th} and 5^{th} row conclude that if the independent jobs constraint relaxes to an arbitrary *dag* then the bound converts from a logarithmic to a linear function of m. Similar effects are indicated from problems 6,7,8,9 and 10 where, the consideration of shared resources causes an increase in the worst-case bounds. Finally, the

bound of problem 11 shows that the presence of the processor constraint, due to a shared memory, contributes about 1 to the worst-case bound.

But, can the worst-case bounds, presented in Table 3.5, be improved if one uses a heuristic approach to sequence the tasks before the actual scheduling takes place? An answer to that question is given in Tables 3.6, 3.7 and 3.8.

Firstly, Table 3.6 shows the worst-case bounds of some cases, where efficient algorithms that produce optimal schedules under a certain set of conditions are considered as heuristics and applied to situations that do not satisfy all or part of these conditions.

One can see that, as far as the homogeneous multiprocessor models are concerned we could not achieve much better worst-case bounds if one of the algorithms, which produces optimal schedules under a certain set of conditions, is used as a heuristic to form a priority list for more general models. For heterogeneous multiprocessor models with uniform processors a worst-case bound has been found, which is independent from the speeds of the processors.

On the other hand, Table 3.7 contains the worst-case bounds for heterogeneous and homogeneous models when simple different ordering strategies are used to construct the priority list needed for a list scheduling algorithm.

So, in homogeneous multiprocessor models with no shared resources and when the partial order $<$ is empty (independent tasks) if a little care is taken to prepare the priority list L then, we can see, the worst-case bounds can be improved considerably. Especially, if the tasks are ordered in non-increasing sequence of their execution time requirements then, as Graham [Gr2] shows, a list scheduling algorithm can never produce a schedule of length greater than (4/3-1/3 m) times the optimal one. Recently, Coffman and Sethi [CS2] have found a general bound (the bound in row 2)

TABLE 3.6: Worst-case bounds - Homogeneous and heterogeneous models - List scheduling algorithms - Priority list constructed from special sequencing algorithms. $\{w_k\}=1$. Parameters n, *and*

	Type of Processors	No. of Processors m	No. of Shared Resources q	Partial Order $\prec$	Time Requirement $T_k=\{t_j\}$	L(alg)	Bounds	References
1.	Identical	$m\geq 3$	0	tree	-	Hu	$\frac{\omega}{\omega_0}\leq 1+(m-1)\frac{\max\{t_i\}}{\sum_{i=1}^{n} t_i}$	[Kau]
2.	Identical	$m\geq 3$	0	tree	-	Hu	$\frac{\omega}{\omega_0}\leq 2-\frac{2}{m+1}$	[Gr4]
3.	Identical	m=2	0	-	Equal	Hu	$\frac{\omega}{\omega_0}\leq \frac{4}{3}$	[Ch]
4.	Identical	$m\geq 3$	0	-	Equal	Hu	$\frac{\omega}{\omega_0}\leq 2-\frac{1}{m-1}$	[Ch]
5.	Identical	$m\geq 3$	0	-	Equal	C.G.	$\frac{\omega}{\omega_0}\leq 2-\frac{2}{m}$	[LS]
6.	Identical	$m\geq 3$	0	tree	-	M.C.	$\frac{\omega}{\omega_0}\leq 2-\frac{2}{m}$	[LS]
7.	Uniform	$m\geq 3$	0	-	-	H.L.S.	$\frac{\omega}{\omega_0}\leq \sqrt{1.5m}$	[HLS]

Notation:

1. ω corresponds to list L formed by a sequencing algorithm; 2. ω_0 corresponds to optimal list L ;
3. L(alg) is the priority list produced by the indicated sequencing algorithm.

TABLE 3.7: Bounds for some heuristic list scheduling algorithms - Completion time. $\{w_k\}=1$. Parameters: n,m *and*

Type of Processors	No. of Shared Resources q	Partial Order $<$	Time Requirement $T_k=\{t_j\}$	L(strat.)	Bounds	References
1. Identical	0	ϕ	-	LTF	$\frac{\omega}{\omega_0}<\frac{4}{3}-\frac{1}{3m}$	[Gr2],[Gr3]
2. Identical	0	ϕ	-	LTF	$\frac{\omega}{\omega_0}<\frac{k+1}{k}-\frac{1}{km}$	[CS2]
3. Identical	0	ϕ	-	First r elements produce an optimal schedule for r largest tasks	$\frac{\omega}{\omega_0}\leq 1+\frac{1-\frac{1}{m}}{1+\left\lfloor\frac{r}{m}\right\rfloor}$ †	[Gr2],[Gr3]
4. Uniform	0	ϕ	-	"	$\frac{\omega}{\omega_0}\leq 1+\frac{1}{Q}-\frac{1}{Q\left(\sum_{i=1}^{\ell} m_i b_i\right)}$	[LiL1]
5. One fast and the rest identical	0	ϕ	-	LTF	$\frac{\omega}{\omega_0}\leq \frac{2(m-1+b)}{b+2}$, $b\leq 2$ $\frac{\omega}{\omega_0}\leq \frac{m-1+b}{2}$, $b\geq 2$	[LiL1]

†*the bound is best possible if* $r\equiv 0 \pmod m$

TABLE 3.7: (Continued)

	Type of Processors	No. of Shared Resources q	Partial Order $\prec$	Time Requirement $T_k=\{t_j\}$	L(strat.)	Bounds	References
6.	Identical	1	ϕ	Equal	LMF	$\frac{\omega-1}{\omega_0}\leq 2-\frac{2}{m}$, $m\geq 2$	[KSS2],[Kr]
7.	Identical	-	-	Equal	CP	$\frac{\omega}{\omega_0}\leq\frac{17}{10}q+1$, $m\geq n$	[GGJY]
8.	Identical with indepen. memories	0	ϕ	-	STF,SMF,SPF	$\frac{\omega}{\omega_0}\leq 1+\log_2(m)$	[KS2], [Kaf]
9.	"	0	ϕ	-	LTF,LPF	$\max\left\{\frac{\omega}{\omega_0}\right\}\geq \ell n(m)$	
10.	"	0	ϕ	-	LMF	$\frac{\omega}{\omega_0}\leq 2-\frac{1}{m}$	
11.	"	0	ϕ	-	LMLT	$\frac{\omega}{\omega_0}\leq\begin{cases}5/4, & m=2\\ 2-\frac{1}{m-1}, & m\geq 3\end{cases}$	

Notation:

1. ω_0 corresponds to optimal list L ;
2. ω corresponds to the list produced by the ordering strategy;
3. k: the maximum number of jobs executed on the processor which finishes at ω ;
4. r: the number of the first r largest tasks;
5. $$Q=\max\left[\min_{1\le j\le \ell}\left(\left\lceil\frac{r+1}{\sum_{i=1}^{\ell} m_i\lceil b_i\rceil}\right\rceil,\ \frac{\lceil b_j\rceil}{b_j}-\frac{\lceil b_j\rceil-1}{b_j}\right),\ \frac{r+1}{\sum_{i=1}^{\ell} m_i b_i}\right] ;$$
6. $m=m_1+m_2+\ldots+m_\ell$, where ℓ is the number of groups with different processor speeds and m_i the number of processors with speed b_i;
7. b: the speed of a processor;
8. LTF largest time first; STF shortest time first;
9. CP critical path;
10. SMF shortest memory first; LMF largest memory first;
11. SPF shortest product first; LPF largest product first;
12. LMLT largest memory first and then the jobs with the same memory requirement largest time first;
13. $\lfloor x\rfloor$: the greatest integer less than or equal to x;
14. $\lceil x\rceil$: the least integer greater than or equal to x.

which includes the bound derived by Graham. That bound is more informative than Graham's. The greater the number of jobs in the task system the closer the schedule lengths, produced by a list scheduling algorithm under LTF ordering strategy, are to the optimal ones. (Although, the authors present an analysis to prove the bound, a much simpler one is known by the author of this thesis.) Another approach (problem 3) to prepare the priority list L for the same problem was also given by Graham. According to this approach, having scheduled the r largest task optimally then the list L is formed by joining the remaining tasks arbitrarily. The larger the number of optimal scheduled tasks the closer to optimal the worst-case bound becomes. However, finding an optimal list for the largest r tasks may itself be a hard problem. The same approach was used by C. Liu and J. Liu [LiL1] for heterogeneous multiprocessor models characterised by uniform processors. As before, the worst-case bound depends on the number of largest tasks scheduled optimally. When there is one processor fast and the rest are identical C. Liu and J. Liu [LiL1] using the LTF ordering strategy have found worst-case bounds relatively better than the corresponding ones, when an arbitrary priority list was used. Problem 6 indicates that if the tasks are ordered according to their memory requirements in non-increasing order (LMF) then the bound of problem 11 in Table 3.5 can be improved by one. Also the bound in row 7 of Table 3.7 shows the improvement which can be achieved if a critical path is used for preparing the priority list of problem 10 in Table 3.5. Further, problems 8 till 11 show the worst-case bounds of a number of different ordering strategies when models of identical processors with private memories are considered. Although some of the ordering strategies can not produced better bounds then an arbitrary priority list, the LMF or LMLT ordering have produced bounds upper bounded by 2.

Finally, Table 3.8 presents the known worst-case bounds for some E.C.T. algorithms as well as the bound for a 2D algorithm.

TABLE 3.8: Worst-case bounds - E.C.T. and 2D scheduling algorithms - Heterogeneous and homogeneous multiprocessor models - Completion time. q=0, $\{w_k\}$=1, $\prec$=empty. Parameters: n,m, $T_k=\{t_j\}$ *and*

	Type of Processors	$L(\omega)$	Bounds	References
1.	Uniform	LTF	$\frac{\omega}{\omega_0} \leq 2 - \frac{2}{m+1}$ †	[GIS]
2.	One fast and the rest identical	LTF	$\frac{\omega}{\omega_0} \leq \frac{3}{2} - \frac{1}{2m}$ ††	[GIS]
3.	Non-identical	-	$\frac{\omega}{\omega_0} \leq m$	[IK]
4.	Non-identical	LTF_{MIN}	$\frac{\omega}{\omega_0} \leq m$	[IK]
5.	Non-identical	LTF_{MAX}	$\frac{\omega}{\omega_0} \leq m$	[IK]
6.	Identical with independent memories	LMF(2D)	$\frac{\omega}{\omega_0} \leq 2 - \frac{2}{m+1}$	[KS2],[Kaf]

†*the bound is best only for m=2*

††*the bound is best only for m≤3*

Notation:

LTF_{MIN}: the smallest time requirement of each job is chosen first and then jobs ordered in LTF according to the chosen values;

LTF_{MAX}: the largest time requirement of each job is chosen first and then jobs ordered in LTF.

Comparing the bounds of problems 1 and 2, in the above table, with the corresponding problems, 3 in Table 3.5 and 4,5 in Table 3.7, one can conjecture the advantages of these algorithms. A disadvantage is that the bounds are not best possible ones, for all $m \geq 2$. When the heterogeneous models with non-identical processors are considered, one could not notice any advantages of the worst-case bounds if some effort is spent to prepare a priority list, according to the results given by Ibarra and Kim [IK] for the problems 3 till 5 in Table 3.8. What one can notice is the large worst-case bounds which has been found in the case of non-identical processors without any shared resources and the partial order being empty. In the last problem, where identical processors with independent memories are considered, a 2D algorithm cannot produce much better worst-case bounds compared with the known bounds of a list scheduling algorithm for the same problem. (See Table 3.7, problems 10,11).

So far, attention has been given to form heuristic scheduling algorithms with respect to completion time performance cirterion. When a task system of independent jobs is scheduled on an arbitrary number of non-identical processors and the mean flow time is considered as the performance criterion, only Clark [Cℓ] has tried to form heuristic scheduling algorithms and find their worst-case bounds. His results are presented in Table 3.9.

Although Clark's bounds are very rough, one could not ignore them since they open a way to form heuristics when the mean flow time criterion is of interest. Also, he has proved that there is a permutation of jobs in the task system for which Q.A.D. algorithms perform optimally. Their time complexity varies from $O(nm)$ to $O(\max\{nm, n\log n\})$ when an arbitrary or permuted list is considered respectively. Thus, Q.A.D. algorithms spend less computation time to schedule a number of independent jobs rather than the optimal known algorithm (see Table 3.4, problem 3).

TABLE 3.9: Worst-case bounds - Q.A.D. scheduling algorithms - Heterogeneous multiprocessor models - Mean flow time. q=0, $\{w_k\}$=1, ≤=empty. Parameters, n, m, T_k $\{t_j\}$ *and*

Type of Processors	L(strategy)	Bounds	References
1. Non-identical	-	$\frac{\bar{\omega}}{\bar{\omega}_0} < n$	[Cℓ]
2. "	LTF_{AVE}		
3. "	LTF_{MAX}		
4. "	LTF_{MIN}	$\frac{\bar{\omega}}{\bar{\omega}_0} < \frac{n+1}{2}$	

Notation:

$\bar{\omega}$ corresponds to the mean flow time of the schedule derived from L(strategy);

$\bar{\omega}_0$ corresponds to an optimal mean flow time schedule;

AVE: the average execution time.

Further, we should mention the effort, which has been made to find good schedules with respect to completion time for priority lists, when higher priorities are assigned to shorter execution time requirement jobs (STF). We recall that if a task system of independent jobs is scheduled on identical processors then a priority-driven scheduling algorithm with priority list formed as described, performs optimally, when the mean flow time is considered as the performance criterion. It has been shown [BCS1] that there are $O(m!^{n/m})$ schedules which might have different completion times but the same optimal mean flow time. Therefore, as it was expected some heuristics were studied. Their worst-case bounds together with the

worst-case bound of a two-phase heuristic algorithm, with respect to mean flow time, appear in Table 3.10.

Generally, schedules produced by using a STF list are bounded as those produced by an arbitrary list (problem 1).

TABLE 3.10: Worst-case bounds - Simple and two-phase list scheduling algorithms - Homogeneous multiprocessor models - Mean flow and completion time. q=0, ◁=empty, $\{w_k\}$=1, Parameters: $n, m, T_k=t_k$

	$L(\omega)$	$L(\omega')$	Bounds	References
1.	OPT	STF	$1 \leq \frac{\omega'}{\omega} \leq 2-\frac{1}{m}$	[BCS1]
2.	LTF	STF	$\frac{\omega'}{\omega} \leq 2-\frac{1}{m}$	
3.	STF	LTF	$\frac{\omega'}{\omega} \leq \frac{5m-2}{4m-1}$	[BCS2]
4.	STF	$\overline{\text{STF}}$	$\frac{\omega'}{\omega} \leq \frac{3}{2}$	[CS1]
5.	STF	STF^1	$1 \leq \frac{\omega'}{\omega} \leq \frac{4m-3}{3m-2}$ †	
6.	STF	STF^2	$1 \leq \frac{\omega'}{\omega} \leq \frac{5m-4}{4m-3}$	
7.	OPT	LTF*	$\frac{\bar{\omega}'}{\bar{\omega}} \leq m$	[BCS1]

†*that bound is best possible only for* $m \leq 3$

Notation:

$\omega, \bar{\omega}$ correspond to the schedule produced by list $L(\omega)$;

$\omega', \bar{\omega}'$ corresponds to the schedule produced by list $L(\omega')$;

$\overline{\text{STF}}$ corresponds to the longest possible schedule a STF list can produce (the largest task of each rank executes on the same processor);

$\underline{\text{STF}}$ corresponds to a minimal length STF schedule;

STF^1 corresponds to STF schedules whose last rank is assigned the largest job first as processors become available;

Notation:

STF^2 corresponds to STF schedules whose tasks of each rank are assigned the largest job first as processors become available;

LTF* corresponds to two-phase schedules produced by LTF list.

Schedules produced by a STF list may be up to 20% shorter than the one produced by a LTF list for a task system. A schedule produced by a $\overline{STF}$ list cannot be more than 50% worse than the one produced by a STF list for a task system. If a STF^1 list is considered, then the constructed schedule is no more than 4/3 times the length of a STF schedule. Finally, if a STF^2 list is considered then STF^2 schedules are at most 25% longer than STF schedules. The inverse problem is of interest as well. We recall that a LTF ordering strategy produces lists which construct schedules very near to optimal. Unfortunately, their mean flow time is very far from optimal. So, without changing the completion time of the LTF schedules we can apply a two-phase algorithm, which is expected to have a near optimal mean flow time. Although the proven bounds is increasing linearly with the number of processors in the system, we cannot reject it since there are no other bounds available to compare with.

3.5 AVERAGE PERFORMANCE OF NON-OPTIMAL SCHEDULING ALGORITHMS

As stated in Chapter 1 there is a question if the deterministic scheduling analysis can also be a useful tool for systems where the average (expected) performance is more meaningful than the worst-case bounds. In addition, it was mentioned there, how this could be verified. So far, towards the verification of this question, which is believed to be positive according to [Gr3], [Ch] and [Kr], very little work has been done. The average performance of an algorithm, with respect to a performance criterion, has been approximated by simulating selected configurations of the computation model and evaluating the algorithm for different task system using statistical analysis.

Adam, Chandy and Dickson [ACD] have chosen the average performance as a measurement to evaluate scheduling algorithms. They have examined the problem of an arbitrary task system, with equally weighted and no shared resource requirement jobs, scheduled on an arbitrary number of identical processors, when the completion time is considered as the performance criterion. The results they have provided show that the scheduling algorithm, which assigns higher priorities to higher level jobs, has significantly better performance (near-optimal one) than the one which assigns priorities randomly. However, apart from the fact that the average performance results are much better than the worst-case ones the ranking of the average performance agrees with the ranking of the worst-case bounds for these two algorithms. (See problem 1, Table 3.5 and problems 2-6, Table 3.6.) Also, they have considered a number of other heuristic list scheduling algorithms, whose worst-case bounds are not known. The near-optimality of longest-path list scheduling algorithm has been confirmed by Kohler [Koh] as well. Moreover, he demonstrated that the average performance of this algorithm deviates from the optimal one as the number of processors increases. One can guess such behaviour of the algorithm from the worst-case bounds which are presented in Table 3.6 (problems 2-6).

On the other hand, Clark [Cℓ], through his simulation study, has shown that the ranking of the average performance of the Q.A.D. algorithms he

considered, with respect to mean flow time is: LTF_{MIN},LTF_{AVE},LTF_{MAX} and RAND. That ranking does not violate the ranking of the worst-case performance bounds presented in Table 3.9. We note that he considered the problem of a task system of independent, equally weighted jobs to be scheduled on an arbitrary number of non-identical processors with no shared memory.

Finally, a promising answer for the question mentioned at the beginning of the present section has been given by Kafura [Kaf]. Kafura examined the problem of a task system of independent, equally weighted jobs to be scheduled on an arbitrary number of identical processors, each one associated with a private memory, and no shared resources, when the completion time is considered as the performance criterion. His simulation results show that there is a high correlation between the ranking of the average performance and the worst-case performance bounds for the list scheduling algorithms he studied.

CHAPTER 4

A HETEROGENEOUS MULTIPROCESSOR COMPUTING MODEL

4.1 INTRODUCTION

From the previous chapter one can see that the multiprocessor computing systems which have been studied by deterministic analysis are:-

(a) those with only processors as resources and where the processors have been considered identical ([Hu],[FKN],[CG],[Se2,3],[U1,2],[BCS1,2],[GJ],[IK],[MC1,2],[McN],[CMM],[Sa],[Gr,1,2,3,4],[Kau],[Ch],[LS],[Li1],[CS1,2]), uniform ([Ba],[LY],[HLS],[CMM],[HoS1],[LiL1],[GIS]), and non-identical ([C],[IK],[BCS1,2],[Br1],[SS2]);

(b) homogeneous models with shared memory ([Kr],[KSS1,2]) or arbitrary many types of shared resources ([GG1,2],[Ya],[GGJY]); and

(c) the models with identical and independent processors, each processing a fixed, though possibly different sized memory ([Kaf],[KS1,2]).

It is obvious therefore, that most of the investigation, which has been done, characterise models with only processors as resources. Such a motivation was due to the tractability of those models and also due to the lack of results as far as the worst-case performance was concerned. Although little attention has been given to models with independent processors, each processing a fixed though possibly different sized memory, they appear very attractive since such models may be interpreted as local networks of mini- or micro-computers (uniprocessors). The intuitive advantages of such complexes over a single mainframe have already been indicated in Chapter 1.

So far, only Kafura and Shen ([Kaf],[KS1,2]) have examined such models with identical processors as mentioned in (c). So, the idea of considering similar models with non-identical or uniform processors sounds very interesting. Those models are of particular interest because they may be interpreted to systems (networks), more close to reality. This could be realised from the necessity to replace a processor, due to its failure to

reach some requirements, with a smaller cheaper and more powerful one, or to add a new processor (or more) to the system, due to increases in work load. Furthermore, the technology of large-scale integration assures us for the production of cheaper and more powerful mini- and micro-computers in the near future. The inherent significance and the lack of results for such models would be enough justification for someone to decide to consider and study them. As a matter of fact those models will be examined in this thesis and we shall refer to them as *heterogeneous multiprocessor systems with independent memories*.

4.2 A FORMAL DESCRIPTION OF THE MODEL

The model of computation being analysed consists of $m \geq 2$ non-identical (or uniform) independent abstract processors $P=\{P_1,P_2,\ldots,P_m\}$. A fixed size private memory, denoted by $|P_i|$, $1 \leq i \leq m$, is associated with each processor. Each memory is private in the sense that information contained in the i^{th} memory is accessible only by the P_i processor. The memory sizes are fixed since $|P_1|,|P_2|,\ldots,|P_m|$ remain constant throughout the execution of a task system. For convenience, the processors are indexed so that $|P_i| \geq |P_{i+1}|$, $1 \leq i \leq m-1$.

The task system J, to be scheduled on the above mentioned set of processors, will consist of n independent jobs ($\prec$ is empty), $J=\{J_1,J_2,\ldots,J_n\}$. Each job is specified by its time requirements $T_j=\{t_i\}$, $1 \leq i \leq m$ and its memory requirement m_j, $1 \leq j \leq n$. So, the task system can be represented by a three-tuple $(J,\{m_j\},\{T_j\})$. An important scheduling parameter is the number of processors which can execute a given task under its memory constraint. For the j^{th} task the number of such processors will be denoted by ℓ_j.

Because of the one dedicated resource the parameter $\{T_j\}$ in the task system can be replaced by a $(m \times n)$ matrix $[t_{ij}]$, that gives complete information about the time requirements of each job on different processors. So, the task system can appear as $(J,m_j,[t_{ij}])$ or just $(J,[t_{ij}])$. When the memory requirement of the j^{th} job cannot be satisfied by the i^{th} processor's memory, then this job cannot be executed on that processor and therefore $t_{ij}=\infty$. In the case when uniform processors are considered, the task time requirements for the processors on which it can be executed, differ by the ratio of their speeds. In particular, if $b_1,b_2,\ldots,b_{\ell_j}$, $1 \leq \ell_j \leq m$, are the speeds of the first ℓ_j processors, where the j^{th} job can run, then the time requirements will be $t_{1j}=t/b_1$, $t_{2j}=t/b_2,\ldots,t_{\ell_j j}=t/b_{\ell_j}$, where t is the time needed by job j to run on a "standard" processor.

4.3 AIMS AND OBJECTIVES

In this thesis, we shall analyse only non-preemptive scheduling algorithms. The nature of the multiprocessor model, which is being considered, does not allow us to examine preemptive algorithms due to the high cost of transmission rates. However, note that the problem of finding optimal non-preemptive scheduling algorithms, with respect to completion time performance criterion, for task systems of independent jobs scheduled on any model configuration considered in this thesis is a NP-hard problem. This is implied from the fact that the problem of determining optimal schedules for task systems of independent jobs on two identical processors is a NP-hard problem (see Table 3.1, problem 8). On the other hand, Bruno's algorithm [Br1] can be adapted to produce optimal schedules for our problem when the mean flow time is taken as the performance criterion. Moreover, its time complexity is $O(\max\{n^3, n^2 m\})$. Therefore, these facts make us devote our attention to heuristic scheduling algorithms.

From sections 3.4 and 3.5 one can realise that the performance of the heuristic scheduling algorithms has either been measured according to the mean flow or the final completion time performance criterion. The only exception was for the classical homogeneous multiprocessor model where the P.D. algorithm under the LTF ordering rule was measured for both of these performance criteria. In the following chapters the performance of all the scheduling algorithms will be measured with respect to the mean flow as well as to the completion time criterion. This approach will give a better understanding of the behaviour of each particular algorithm. Furthermore, we will be able to find out if there is any algorithm which performs well under both performance criteria. The existence of such an algorithm is of great importance because both computer users and system manager will be satisfied.

Among the various heuristic scheduling algorithms we shall consider only the demand scheduling algorithms. Within this class Priority-Driven, Q.A.D. and Two-Phase algorithms will be analysed. When two or more processors are available simultaneously, or a task has the same flow time contribution on two or more processors then, the task is assigned to that processor which executes it faster and if there is another "tie break" at this stage, the task is assigned to the processor with the largest index (i.e., smaller memory).

We shall use the deterministic scheduling theory as well as simulation techniques to analyse the behaviour of the algorithms. So, using the deterministic analysis we shall establish worst-case performance bounds for each algorithm under various priority lists. Actually, in Chapters 5 and 6 we evaluate the algorithms with respect to the mean flow and completion time criterion, respectively[†]. So far (except for one case only [CS2]), the worst-case bounds were shown to be best possible using single pathological examples. As a result, such bounds are not as informative as might be desired. Therefore, more informative bounds are demanded and for this reason we consider that as an additional objective for this research. Finally, as indicated in Chapter 1, this thesis has the aim to reveal evidence towards the degree of correlation between the extreme and expected behaviour of the algorithms. So, in order to make that possible, in Chapter 7 the algorithms will be evaluated by their average performance using simulation techniques.

[†]*Moreover, in Apendices I-IV the worst-case bounds of the algorithms for the corresponding homogeneous multiprocessor model are also given.*

4.4 A FORMAL DESCRIPTION OF THE ALGORITHMS

Although a brief description of the algorithms which we are concerned with has already been given in section 2.5, it would be better to specify them completely for the computation model under investigation.

In every algorithm, which appears in a structured programming form, the input is as follows:

Input: m, the number of processors; n, the number of jobs; $J=\{J_1,J_2,\ldots,J_n\}$, the set of jobs to be scheduled; $[t_{ij}]$, a (m×n) matrix defining the time requirements of the jobs on the various processors; and ℓ_j, the number of processors with sufficient memory to run the J_j job.

Algorithm 1.

procedure Priority-Driven algorithms (m,n,J,$[t_{ij}]$,L_i)

//input: as given above;

output: L_i, $1\leq i\leq m$, the set of jobs to be run on processor P_i in a FIFO order//

A←1; //P.D. algorithm//

A←2; //P.D.* algorithm//

//step 1// *for* i=1(1)m *do*

F_i←0; //F_i, the current completion time of processor P_i//

L_i←∅;

end;

//step 2// *for* j=1(1)n *do*

find the set $\{P_k\}$:$F_k=\min_{1\leq i\leq \ell_j}\{F_i\}$; //find the processor(s) with the earliest completion time//

let s be the largest index of the processors in $\{P_k\}$ where $t_{rj}=\min_{i\in\{k\}}\{t_{ij}\}$;

L_s←L_sUJ_j;//assign the job J_j on the P_s processor//

F_s←F_s+t_{sj};//update the completion time of processor P_s//

end;

//step 3// *if* A=2 *then for* i=1(1)m *do*

put the jobs in L_i, $1 \leq i \leq m$, in non-decreasing order according to their time requirements;

end;

return.

Algorithm 2.

procedure Quick-And-Dirty algorithms (m,n,J,$[t_{ij}]$,Q_i)

//input: as given at the beginning;

output: Q_i, $1 \leq i \leq m$, the set of jobs to be run on processor P_i in a LIFO order//

B←1;//Q.A.D. algorithm//

B←2;//Q.A.D.* algorithm//

//step 1// *for* i=1(1)m *do*

$Q_i \leftarrow \emptyset$;

$h_i \leftarrow 1$;//h_i, the number of jobs already allocated on the P_i processor increased by 1//

end;

//step 2// *for* j=1(1)n *do for* i=1(1)m *do*

$R_{ij} \leftarrow h_i t_{ij}$;//$R_{ij}$, the mean flow time contribution of the job J_j on P_i processor//

end;

find the set $\{P_k\}: R_{kj} = \min_{1 \leq i \leq \ell_j} \{R_{ij}\}$;//find the processor(s) where the job J_j has the minimum mean flow time contribution//

let s be the largest indexed processor in $\{P_k\}$ where

$t_{rj} = \min_{i \in \{k\}} \{t_{ij}\}$;

$Q_i \leftarrow Q_i \cup J_j$;

$h_i \leftarrow h_i + 1$;

end;

```
//step 3//  if B=2 then for i=1(1)m do
                                put the jobs in Q_i, 1≤i≤m in non-increasing
                                order according to their time requirements;
                            end;
        return.
```

Therefore, the time complexity of the P.D. and P.D.* algorithms are $O(n\log_2 m)$ and $O(n\log_2 n)$ respectively, since we can implement steps 1,2 and 3 of the Algorithm 1 in times proportional to m, $n\log_2 m$ and $n\log_2 n$, respectively. On the other hand, the Q.A.D. and Q.A.D.* algorithms have time complexity expressed by $O(nm)$ and $O(\max\{nm, n\log_2 n\})$, since the steps 1,2 and 3 of Algorithm 2 can be implemented in times proportional to m, nm and $n\log_2 n$ respectively.

CHAPTER 5

DETERMINISTIC ANALYSIS OF HEURISTIC SCHEDULING ALGORITHMS - MEAN FLOW TIME PERFORMANCE CRITERION

5.1 INTRODUCTION

As was indicated in section 4.2, this thesis is concerned with the problem of scheduling a task system $(\mathcal{J},[t_{ij}])$ of independent jobs on heterogeneous multiprocessor systems with independent memories and a fixed number of processors. This means that a number of scheduling algorithms should be examined. In fact, the aim is to analyse non-preemptive scheduling algorithms with respect to mean flow time as well as to completion time performance criterion. The investigation begins in this chapter with the analysis of the worst-case performance of several scheduling algorithms when the mean flow time is chosen as the performance criterion.

The mean flow time, as described in section 2.4, is the sum of finishing times of all the jobs divided by the number of the jobs in the task system, i.e., $\bar{\omega}=\frac{1}{n}\sum_{j=1}^{n} f_j$ or equivalently $\bar{\omega}=\frac{1}{n}\sum_{j=1}^{n} k_j t_{ij}$, where k_j is one greater than the number of tasks following J_j on the i^{th} processor. Instead of comparing such quantities to decide the performance of a particular heuristic algorithm relative to the optimal one, it is equivalent to compare just the summation of the finishing times. So, we shall follow this approach in the remainder of this thesis, i.e., we shall consider

$$\bar{\omega} = \sum_{j=1}^{n} f_j \qquad \text{or} \qquad \bar{\omega} = \sum_{j=1}^{n} k_j t_{ij} \ . \tag{5.1.1}$$

A variety of scheduling algorithms i.e., priority driven, two-phase priority driven, quick and dirty as well as two-phase quick and dirty, under a number of ordering rules will be analysed in sequence. At first, a worst-case bound is derived for each class of algorithms, when an arbitrary (random) ordering rule is used to form the priority list. Such a bound is meaningful because an arbitrary ordering rule represents a first-come, first-served (FCFS) scheduling function. In addition, this is the largest possible bound since, the priority list of an arbitrary ordering rule can be chosen to be the same as the priority list formed by any other ordering rule.

Having established the worst-case bound for an arbitrary ordering rule the analysis then turns to the examination of several heuristic ordering rules. However, since we are dealing with heterogeneous multiprocessor systems we cannot order the jobs according to execution time requirements until a single value is chosen for each job. Such single values can be obtained by considering the minimum or the maximum execution time of each job. Therefore, using one of these simple functions, when it is required, the following ordering rules are examined:

- *largest memory first* (LMF) : job J_j preceeds job J_k if $\ell_j \leq \ell_k$;
- *shortest time first* (STF_A)†: job J_j preceeds job J_k if $t_j^A \leq t_k^A$ ††;
- *largest time first* (LTF_A) : job J_j preceeds job J_k if $t_j^A \geq t_k^A$;
- *largest memory shortest time* ($LMST_A$) : job J_j preceeds job J_k if:

  (a) $\ell_j < \ell_k$ or (b) $\ell_j = \ell_k$ and $t_j^A \leq t_k^A$;
- *largest memory largest time* ($LMLT_A$) : job J_j preceeds job J_k if:

  (a) $\ell_j < \ell_k$ or (b) $\ell_j = \ell_k$ and $t_j^A \geq t_k^A$.

However, using any of these ordering rules the complexity of the P.D. or P.D.* and Q.A.D. or Q.A.D.* algorithms will be $O(n\log_2 n)$ and $O(\max\{mn, n\log_2 n\})$ respectively.

---

† *A is the function to single value the jobs.*

†† t_p^A *is the time requirement of job* J_p, $1 \leq p \leq n$, *according to the function A.*

5.2 PRELIMINARIES

Before we start analysing the scheduling algorithms it is necessary to define a number of variables, to declare some set of jobs as well as to prove some lemmas, which are used in the following sections.

Let us consider a task system $(J,\{m_j\},[t_{ij}])$, as being defined in section 4.2. We define τ as the minimum execution time requirement of a job, i.e. $\tau_j = \min_{1\leq i\leq \ell_j} \{t_{ij}\}$, where $1\leq \ell_j \leq m$ and $1\leq j\leq n$, and σ as the maximum execution time requirement of a job, i.e., $\sigma_j = \max_{1\leq i\leq \ell_j} \{t_{ij}\}$, where $1\leq \ell_j \leq m$ and $1\leq j\leq n$.

Also, we define:

$$\tau_j^{F_r} = \min_{\substack{1\leq i\leq r \\ J_j\in F_r}} \{t_{ij}\}, \qquad \sigma_j^{F_r} = \max_{\substack{1\leq i\leq r \\ J_j\in F_r}} \{t_{ij}\} \text{ , where}$$

F_r is the set of jobs with $\ell_j=r$, $1\leq r\leq m$;

moreover, let n_r be the number of jobs belonging to F_r;

$$\tau_j^{G_r} = \min_{\substack{1\leq i\leq \ell_j \\ J_j\in G_r}} \{t_{ij}\}, \qquad \sigma_j^{G_r} = \max_{\substack{1\leq i\leq \ell_j \\ J_j\in G_r}} \{t_{ij}\}, \text{ where } 1\leq \ell_j \leq m$$

and G_r is the set of jobs scheduled on the P_r processor, where $1\leq r\leq m$; moreover let n_r' be the number of jobs belonging to G_r;

Finally, D_r is the set of jobs $\{J_{(r-1)m+1},\ldots,J_{rm}\}$ in the priority list, where $1\leq r\leq \left\lceil \frac{n}{m} \right\rceil$. If n is not a multiple of m then, $D_{\left\lceil \frac{n}{m} \right\rceil}=\{J_{\left\lfloor \frac{n}{m} \right\rfloor m+1},\ldots,J_n\}$. Similarly, F_i^r is the set of jobs $\{J_{(r-1)i+1},\ldots,J_{ir}\}$ inside F_i, $1\leq r\leq \left\lceil \frac{n_i}{i} \right\rceil$, and G_i^r is the set of jobs $\{J_{(r-1)\nu_i+1},\ldots,J_{r\nu_i}\}$ inside G_i, $1\leq r\leq \left\lceil \frac{n_i'}{\nu_i} \right\rceil$ and $\nu_i = \max_{J_j\in G_i} \{\ell_j\}$. However it is obvious that $\bigcup_r \{D_r\}=J$, $\bigcup_r \{F_i^r\}=F_i$ and $\bigcup_r \{G_i^r\}=G_i$.

Now, we continue this section of preliminaries with the description and proof of several useful lemmas.

Lemma 5.2.1: If $a_{s+1} \geq a_{s+2} \geq \ldots \geq a_{s+d}$, where $a_{s+i} \in R^{+*}$ for i=1(1)d and $s=d(\ell-1)$, where $\ell, d \geq 1$ and $\ell, d \in Z^+$, then

$$\frac{(k+s+1)a_{s+1}+(k+s+2)a_{s+2}+\ldots+(k+s+d)a_{s+d}}{\ell(a_{s+1}+a_{s+2}+\ldots+a_{s+d})} \leq \frac{k}{\ell} + d - \frac{d-1}{2\ell}, \qquad (5.2.1)$$

where $k \geq 0$ and $k \in Z^+$.

Proof: The above inequality (5.2.1) is equivalent to

$$\frac{2k+2d\ell-d+1}{2\ell} - \frac{(k+s+1)a_{s+1}+(k+s+2)a_{s+2}+\ldots+(k+s+d)a_{s+d}}{\ell(a_{s+1}+a_{s+2}+\ldots+a_{s+d})} \geq 0$$

which, since all a_i, i=s+1(1)s+d and ℓ are positive, is equivalent to

$$(2k+2d\ell-d+1)(a_{s+1}+a_{s+2}+\ldots+a_{s+d})-2(k+s+1)a_{s+1}$$
$$-2(k+s+2)a_{s+2}-\ldots-2(k+s+d-1)a_{s+d-1}-2(k+s+d)a_{s+d} \geq 0 \qquad (5.2.2)$$

Because $s=d\ell-d$ the inequality (5.2.2) is equivalent to

$$(d-1)a_{s+1}+(d-3)a_{s+2}+\ldots+(3-d)a_{s+d-1}+(1-d)a_{s+d} \geq 0. \qquad (5.2.3)$$

Matching a_{s+1} with a_{s+d}, a_{s+2} with a_{s+d-1} and so on, we obtain

$$(d-1)(a_{s+1}-a_{s+d})+(d-3)(a_{s+2}-a_{s+d-1})+\ldots+(d-d)a_{s+\lceil \frac{d}{2} \rceil} \geq 0, \qquad (5.2.4)$$

if d is an odd integer; and

$$(d-1)(a_{s+1}-a_{s+d})+(d-3)(a_{s+2}-a_{s+d-1})+\ldots+(a_{s+\frac{d}{2}}-a_{s+\frac{d}{2}+1}) \geq 0, \qquad (5.2.4')$$

if d is an even integer.

Due to the constraint that $a_i \geq a_{i+1} > 0$ for i=s+1(1)s+d-1, each term in (5.2.4) and (5.2.4') is non-negative. This establishes (5.2.3) and therefore (5.2.1). Equality holds in (5.2.4) and (5.2.4') if all the terms are 0, or equivalently $a_i = a_{i+1}$ for i=s+1(1)s+d-1; otherwise, some

---

* R^+ *represents the set of real positive numbers.*

term will be positive and (5.2.1) will be strictly obtained ¤

(Note: This lemma generalises the one proved by Professor G.W. Stewart in Clark's report ([Cℓ], page 4). He proved inequality (5.2.1) when k=0 and s=0.)

Lemma 5.2.2: If $a_1,a_2,\ldots,a_n \in R^+$, $n\geq 2$ and

$$\frac{c_{s+1}a_{s+1}+c_{s+2}a_{s+2}+\ldots+c_{s+d_\ell}a_{s+d_\ell}}{g_\ell^{\times}(a_{s+1}+a_{s+2}+\ldots+a_{s+d_\ell})} \leq f_\ell(c,g) \quad , \tag{5.2.5}$$

where $s,\ell,d_i,v\in Z^+$, $s=\sum_{i=1}^{\ell-1} d_i$, $\ell=1(1)v$, $\sum_{i=1}^{v} d_i=n$, $\{c_{s+j},\ g_\ell\}\in R^+$, $c_{s+j}\neq c_{s+j+1}$

for $1\leq j\leq d_\ell-1$, and f_ℓ is a positive function, then

$$\frac{\sum_{\ell=1}^{v}\sum_{j=1}^{d_\ell}(c_{s+j}a_{s+j})}{\sum_{\ell=1}^{v} g_\ell^{\times}(\sum_{j=1}^{d_\ell} a_{s+j})} \leq \max_{1\leq\ell\leq v}\{f_\ell(c,g)\} \ . \tag{5.2.6}$$

Proof: From inequality (5.2.5) for $\ell=1(1)v$ we get the following inequalities respectively:

$$\left.\begin{aligned} c_1a_1+c_2a_2+\ldots+c_{d_1}a_{d_1} &\leq f_1(c,g)\times[g_1^{\times}(a_1+\ldots+a_{d_1})] \\ c_{d_1+1}a_{d_1+1}+\ldots+c_{d_1+d_2}a_{d_1+d_2} &\leq f_2(c,g)\times[g_2^{\times}(a_{d_1+1}+\ldots+a_{d_1+d_2})] \\ &\ \ \vdots \\ c_{n-d_v+1}a_{n-d_v+1}+\ldots+c_na_n &\leq f_v(c,g)\times[g_v^{\times}(a_{n-d_v+1}+\ldots+a_n)] \ . \end{aligned}\right\} \tag{5.2.7}$$

Since both sides of the inequalities (5.2.7) are positive the next inequality holds

$$\sum_{\ell=1}^{v}\sum_{j=1}^{d_\ell}(c_{s+j}a_{s+j}) \leq \max_{1\leq\ell\leq v}\{f_\ell(c,g)\}\left[\sum_{\ell=1}^{v} g_\ell^{\times}(\sum_{j=1}^{d_\ell} a_{s+j})\right] \ .$$

Moving the second factor of the right hand side to the left hand side we get eventually (5.2.6). Equality holds if v=1 (i.e., $n=d_1$) and

the equality condition of (5.2.5) for $\ell=1$ is fulfilled ¤

Lemma 5.2.3: If $a_1 \geq a_2 \geq \ldots \geq a_n$ and $b_1, b_2, \ldots, b_n$ are two sequences of numbers, then,

(i) a permutation $\alpha=(\alpha_1,\alpha_2,\ldots,\alpha_n)$ minimises the quantity

$$\sum_{j=1}^{n} a_j b_{\alpha_j} \tag{5.2.8}$$

if and only if $b_{\alpha_1} \leq b_{\alpha_2} \leq \ldots \leq b_{\alpha_n}$.

(ii) a permutation $\alpha'=(\alpha'_1,\alpha'_2,\ldots,\alpha'_n)$ maximises the quantity (5.2.8) if and only if $b_{\alpha'_1} \geq b_{\alpha'_2} \geq \ldots \geq b_{\alpha'_n}$.

Proof: In order to show (i), we shall prove that the permutation which minimises the quantity (5.2.8) is the one which orders the b_i, $1 \leq i \leq n$, in non-decreasing order.

Let us suppose that this is not true and the permutation which minimises (5.2.8) is given by:

$$\alpha^1 = (\alpha_1,\ldots,\alpha_{j-1},\alpha_{j+k},\alpha_{j+1},\ldots,\alpha_{j+k-1},\alpha_j,\alpha_{j+k+1},\ldots,\alpha_n).$$

Therefore,

$$a_j b_{\alpha_j} + a_{j+k} b_{\alpha_{j+k}} > a_j b_{\alpha_{j+k}} + a_{j+k} b_{\alpha_j} \tag{5.2.9}$$

or

$$(a_j - a_{j+k}) b_{\alpha_j} + (a_{j+k} - a_j) b_{\alpha_{j+k}} > 0$$

or

$$(a_j - a_{j+k})(b_{\alpha_j} - b_{\alpha_{j+k}}) > 0. \tag{5.2.10}$$

But the inequality (5.2.10) does not hold, since $a_j \geq a_{j+k}$ and $b_{\alpha_j} \leq b_{\alpha_{j+k}}$. This contradicts our assumption and establishes that the non-decreasing sequence of b_{α_j}, $j=1(1)n$ minimises the quantity $\sum_{j=1}^{n} a_j b_{\alpha_j}$.

A similar analysis can be used to show the second part (ii) of the lemma. Now, the equivalent inequalities for (5.2.9)-(5.2.10) will have the opposite direction¤

(Note: The first part of this lemma can be found in [Co] (p.136) with only a suggestion for the proof.)

Lemma 5.2.4: Let a task system $(J,[t_{ij}])$ with n independent jobs be scheduled on a heterogeneous multiprocessor system with m independent memories. If $\bar{\omega}_0$ corresponds to the optimal mean flow time of the above task system, then

$$\bar{\omega}_{OPT} \geq \tau_1 + \tau_2 + \ldots + \tau_n \ . \tag{5.2.11}$$

Proof: If c_j is the contribution of job J_j to an optimal schedule with respect to mean flow time, then

$$\bar{\omega}_{OPT} = c_1 + c_2 + \ldots + c_n \ .$$

Thus, if we show that $c_j \geq \tau_j$ for j=1(1)n, then (5.2.11) is proven.

From equation (5.1.1) we see that the contribution of a job J_j to mean flow time is

$$c_j = k_j t_{ij} \ , \tag{5.2.12}$$

where k_j is one greater than the number of tasks following J_j on i^{th} processor. However, since $k_j \geq 1$ and $t_{ij} \geq \tau_j$ then from the equation (5.2.12) we always get $c_j \geq \tau_j$.

Equality in (5.2.11) holds only if the number of jobs in the task system is less or equal to the number of processors (i.e. $n \leq m$), and the jobs minimum time requirement occurs on different processors. That means $k_j=1$ and $t_{ij}=\tau_j$ for j=1(1)n.

Lemma 5.2.5: Let a task system $(J,[t_{ij}])$ with n independent jobs, which are in a STF_{MIN} ordering (i.e., $\tau_n \leq \tau_{n-1} \leq \ldots \leq \tau_1$), be scheduled on a heterogeneous multiprocessor system with m independent memories. Then,

$$\bar{\omega}_{OPT} \geq (\tau_1+\tau_2+\ldots+\tau_m)+2(\tau_{m+1}+\ldots+\tau_{2m})+\ldots+\left\lceil\frac{n}{m}\right\rceil(\tau_{\left\lfloor\frac{n}{m}\right\rfloor m+1}+\ldots+\tau_{\left\lceil\frac{n}{m}\right\rceil m}) \ , \tag{5.2.13}$$

where $\left\lceil\frac{n}{m}\right\rceil m$ is the first integer greater or equal to n which can be

divided by m, and the tasks $J_{n+1},\ldots,J_{\lceil \frac{n}{m} \rceil m}$ have zero execution time requirements.

Proof: If the minimum time requirement of the jobs in consequent ranks of m tasks occur on different processors, then the optimal mean flow time of the above task system is given by,

$$\bar{\omega}_{OPT} = (\tau_1+\tau_2+\ldots+\tau_m)+2(\tau_{m+1}+\ldots+\tau_{2m})+\ldots+\lceil \frac{n}{m} \rceil (\tau_{\lceil \frac{n}{m} \rceil m+1}+\ldots+\tau_{\lceil \frac{n}{m} \rceil m}). \quad (5.2.14)$$

This is true because of $\bar{\omega}=\sum_{j=1}^{n} k_j t_{ij}$, $\tau_j = \min_{1\leq i\leq \ell_j} \{t_{ij}\}$ for $j=1(1)n$, $1\leq \ell_j \leq m$ and Lemma 5.2.3.

Unfortunately, the above assumption is satisfied only occasionally. This happens because, even in an optimal schedule, the memory constraint may force some jobs of the task system belonging to the r^{th} rank, $1\leq r\leq \lceil \frac{n}{m} \rceil$ to run in a consequent rank. This means that the STF ordering will not hold any more and hence the value of the optimal mean flow time will increase and will become greater than the right hand side of (5.2.14) (see Lemma 5.2.3).

Also, since it is unlikely the minimum time requirements of all the jobs in consequent ranks of m tasks to occur on different processors, the contribution of some jobs to the optimal schedule, with respect to mean flow time, will be greater than the corresponding one, which has been considered in the evaluation of (5.2.14) (i.e, $t_{ij}\geq \tau_j$ for $i=1(1)\ell_j$, $1\leq \ell_j \leq m$ and $j=1(1)n$). Thus, the value of optimal mean flow time will be greater than the right hand side of (5.2.14) in this case as well.

Therefore, inequality (5.2.13) is always held□

Lemma 5.2.6: Let a task system $(J,[t_{ij}])$ with n independent jobs, which are in a STF_{MAX} ordering, (i.e. $\sigma_n\leq\sigma_{n-1}\leq\ldots\leq\sigma_1$) bé scheduled on a heterogeneous multiprocessor system with m independent memories. Then,

(i) for non-identical processors:

$$\bar{\omega}_{OPT} > \frac{1}{\lambda}[(\sigma_1+\sigma_2+\ldots+\sigma_m)+2(\sigma_{m+1}+\ldots+\sigma_{2m})+\ldots+\left\lceil\frac{n}{m}\right\rceil(\sigma_{\left\lfloor\frac{n}{m}\right\rfloor m+1}+\ldots+\sigma_{\left\lceil\frac{n}{m}\right\rceil m})] \; ; \tag{5.2.15}$$

and (ii) for uniform processors:

$$\bar{\omega}_{OPT} > \frac{1}{\beta}[(\sigma_1+\sigma_2+\ldots+\sigma_m)+2(\sigma_{m+1}+\ldots+\sigma_{2m})+\ldots+\left\lceil\frac{n}{m}\right\rceil(\sigma_{\left\lfloor\frac{n}{m}\right\rfloor m+1}+\ldots+\sigma_{\left\lceil\frac{n}{m}\right\rceil m})], \tag{5.2.15'}$$

where $\left\lceil\frac{n}{m}\right\rceil m$ is as described in Lemma 5.2.5, $J_{n+1},\ldots,J_{\left\lceil\frac{n}{m}\right\rceil m}$ have zero execution time requirements, $\lambda = \max_{1\leq j\leq n}\left\{\frac{\sigma_j}{\tau_j}\right\}$ and $\beta = \max_{1\leq i\leq m}\{b_i\}/\min_{1\leq i\leq m}\{b_i\}$.

Proof: Let the permutation $\alpha=(\alpha_1,\alpha_2,\ldots,\alpha_n)$ produce a priority list $L=(J_{\alpha_n},J_{\alpha_{n-1}},\ldots,J_{\alpha_2},J_{\alpha_1})$ so that $\tau_{\alpha_n}\leq\tau_{\alpha_{n-1}}\leq\ldots\leq\tau_{\alpha_2}\leq\tau_{\alpha_1}$. When the jobs are in such order, then according to Lemma 5.2.5 the optimal mean flow time is always

$$\bar{\omega}_{OPT} \geq (\tau_{\alpha_1}+\ldots+\tau_{\alpha_m})+2(\tau_{\alpha_{m+1}}+\ldots+\tau_{\alpha_{2m}})+\ldots+\left\lceil\frac{n}{m}\right\rceil(\tau_{\alpha_{\left\lfloor\frac{n}{m}\right\rfloor m+1}}+\ldots+\tau_{\alpha_{\left\lceil\frac{n}{m}\right\rceil m}}).$$

But, $\tau_{\alpha_j} \geq \frac{\sigma_{\alpha_j}}{\lambda}$, $1\leq j\leq n$. Thus, the above inequality becomes

$$\bar{\omega}_{OPT} > \frac{1}{\lambda}[(\sigma_{\alpha_1}+\ldots+\sigma_{\alpha_m})+2(\sigma_{\alpha_{m+1}}+\ldots+\sigma_{\alpha_{2m}})+\ldots+\left\lceil\frac{n}{m}\right\rceil(\sigma_{\alpha_{\left\lfloor\frac{n}{m}\right\rfloor m+1}}+\ldots+\sigma_{\alpha_{\left\lceil\frac{n}{m}\right\rceil m}})].$$

The sum of the right hand side of the last inequality is minimised when σ_{α_i}, $i=1(1)\left\lceil\frac{n}{m}\right\rceil m$ are in non-decreasing order. (See Lemma 5.2.3) Therefore,

$$\bar{\omega}_{OPT} > \frac{1}{\lambda}[(\sigma_1+\ldots+\sigma_m)+2(\sigma_{m+1}+\ldots+\sigma_{2m})+\ldots+\left\lceil\frac{n}{m}\right\rceil(\sigma_{\left\lfloor\frac{n}{m}\right\rfloor m+1}+\ldots+\sigma_{\left\lceil\frac{n}{m}\right\rceil m})] \; ,$$

which proves part (i) of the Lemma. In the case of uniform processors it is obvious that λ should be substituted by β. So, using a similar analysis we can verify part (ii) as well. □

Lemma 5.2.7: Let a task system $(J,[t_{ij}])$ with n independent jobs, which are in a $LMST_{MIN}$ ordering (i.e. $\tau_{n_i}^{F_i} \lesssim \tau_{n_i-1}^{F_i} \lesssim \ldots \lesssim \tau_2^{F_i} \lesssim \tau_1^{F_i}$, $1 \lesssim i \lesssim m$), be scheduled on a heterogeneous multiprocessor system with m independent memories. Then,

$$\bar{\omega}_{OPT} \gtrsim (\tau_1^{F_1}+2\tau_2^{F_1}+\ldots+n_1\tau_{n_1}^{F_1})+[(\tau_1^{F_2}+\tau_2^{F_2})+2(\tau_3^{F_2}+\tau_4^{F_2})+\ldots+\left\lceil\frac{n_2}{2}\right\rceil(\tau_{\left\lfloor\frac{n_2}{2}\right\rfloor 2+1}^{F_2}+\tau_{\left\lceil\frac{n_2}{2}\right\rceil 2}^{F_2})]$$

$$+\ldots+[(\tau_1^{F_m}+\ldots+\tau_m^{F_m})+2(\tau_{m+1}^{F_m}+\ldots+\tau_{2m}^{F_m})+\ldots+\left\lceil\frac{n_m}{m}\right\rceil(\tau_{\left\lfloor\frac{n_m}{m}\right\rfloor m+1}^{F_m}+\ldots+\tau_{\left\lceil\frac{n_m}{m}\right\rceil m}^{F_m})], \tag{5.2.16}$$

where $\sum_{i=1}^{m} n_i = n$, $\left\lceil\frac{n_i}{i}\right\rceil i$ is the first integer greater than or equal to n_i, which can be divided by i, and the jobs $J_{n_i+1}^{i},\ldots,J_{\left\lceil\frac{n_i}{i}\right\rceil i}^{i}$, $1 \lesssim i \lesssim m$, have zero execution time requirements.

Proof: Let us suppose that the jobs J_j^1 (with $\ell_j=1$) have been scheduled on P_1 after any other job with $2 \lesssim \ell_j \lesssim m$. Therefore, according to Lemma 5.2.3, their contribution to the optimal mean flow time will be

$$\bar{\omega}_0^1 = \tau_1^{F_1}+2\tau_2^{F_1}+\ldots+n_1\tau_{n_1}^{F_1}, \tag{5.2.17}$$

provided $\tau_j^{F_1}$, $j=1(1)n_1$, are in non-increasing order. However, the condition made for the jobs with $\ell_j=1$ cannot be always satisfied even in an optimal schedule. So, some of the coefficients of $\tau_j^{F_1}$, $1 \lesssim j \lesssim n_1$, in the equality (5.2.17) will become greater than j. This means that the contribution of these jobs to the optimal mean flow time is greater than the right hand side of (5.2.17). Finally, we can say,

$$\bar{\omega}_0^1 \gtrsim \tau_1^{F_1}+2\tau_2^{F_1}+\ldots+n_1\tau_{n_1}^{F_1}. \tag{5.2.17'}$$

Also, if the jobs J_j^2 (with $\ell_j=2$) have been scheduled on P_1 and P_2 processors after any other jobs with $\ell_j \neq 2$ and their minimum time requirements in consequent ranks of two tasks occur on different processors, then their contribution to the optimal mean flow time will be

$$\bar{\omega}_0^2 = (\tau_1^{F_2}+\tau_2^{F_2})+2(\tau_3^{F_2}+\tau_4^{F_2})+\ldots+\left\lceil\frac{n_2}{2}\right\rceil(\tau_{\left\lfloor\frac{n_2}{2}\right\rfloor 2+1}^{F_2}+\tau_{\left\lceil\frac{n_2}{2}\right\rceil 2}^{F_2}) \quad . \tag{5.2.18}$$

Again, such a condition is occasionally satisfied and therefore, the contributions of jobs J_j^2, $1\leq j\leq n_2$, to the optimal mean flow time is always

$$\bar{\omega}_0^2 \geq (\tau_1^{F_2}+\tau_2^{F_2})+2(\tau_3^{F_2}+\tau_4^{F_2})+\ldots+\left\lceil\frac{n_2}{2}\right\rceil(\tau_{\left\lfloor\frac{n_2}{2}\right\rfloor 2+1}^{F_2}+\tau_{\left\lceil\frac{n_2}{2}\right\rceil 2}^{F_2}) \quad . \tag{5.2.18'}$$

Following the same way of thinking we can show that for the jobs J_j^3, $1\leq j\leq n_3$, we have

$$\bar{\omega}_0^3 \geq (\tau_1^{F_3}+\tau_2^{F_3}+\tau_3^{F_3})+2(\tau_4^{F_3}+\tau_5^{F_3}+\tau_6^{F_3})+\ldots+\left\lceil\frac{n_3}{3}\right\rceil(\tau_{\left\lfloor\frac{n_3}{3}\right\rfloor 3+1}^{F_3}+\ldots+\tau_{\left\lceil\frac{n_3}{3}\right\rceil 3}^{F_3}) \quad , \tag{5.2.19'}$$

and finally, for the jobs J_j^m, $1\leq j\leq n_m$,

$$\bar{\omega}_0^m \geq (\tau_1^{F_m}+\ldots+\tau_m^{F_m})+2(\tau_{m+1}^{F_m}+\ldots+\tau_{2m}^{F_m})+\ldots+\left\lceil\frac{n_m}{m}\right\rceil(\tau_{\left\lfloor\frac{n_m}{m}\right\rfloor m+1}^{F_m}+\ldots+\tau_{\left\lceil\frac{n_m}{m}\right\rceil m}^{F_m}) \quad . \tag{5.2.20'}$$

But, $\bar{\omega}_{OPT}=\bar{\omega}_0^1+\bar{\omega}_0^2+\ldots+\bar{\omega}_0^m$. Thus, substituting $\bar{\omega}_0^1,\bar{\omega}_0^2,\ldots,\bar{\omega}_0^m$ by the right hand side of the inequalities (5.2.17'),(5.2.18'),...,(5.2.20') respectively, we get the inequality (5.2.16) which proves the lemma□

Lemma 5.2.8: Let a task system $(J,[t_{ij}])$ with n independent jobs, which are in a $LMST_{MAX}$ ordering (i.e., $\sigma_{n_i}^{F_i}\leq\sigma_{n_i-1}^{F_i}\leq\ldots\leq\sigma_1^{F_i}$, $1\leq i\leq m$), be scheduled on a heterogeneous multiprocessor system with m independent memories. Then, for non-identical processors,

$$\bar{\omega}_{OPT} > \frac{1}{\lambda_1}(\sigma_1^{F_1}+2\sigma_2^{F_1}+\ldots+n_1\sigma_{n_1}^{F_1})+\frac{1}{\lambda_2}[(\sigma_1^{F_2}+\sigma_2^{F_2})+2(\sigma_3^{F_2}+\sigma_4^{F_2})+\ldots+\left\lceil\frac{n_2}{2}\right\rceil(\sigma_{\left\lfloor\frac{n_2}{2}\right\rfloor 2+1}^{F_2}+\sigma_{\left\lceil\frac{n_2}{2}\right\rceil 2}^{F_2})]$$
$$+\ldots+\frac{1}{\lambda_m}[(\sigma_1^{F_m}+\ldots+\sigma_m^{F_m})+2(\sigma_{m+1}^{F_m}+\ldots+\sigma_{2m}^{F_m})+\ldots+\left\lceil\frac{n_m}{m}\right\rceil(\sigma_{\left\lfloor\frac{n_m}{m}\right\rfloor m+1}^{F_m}+\ldots+\sigma_{\left\lceil\frac{n_m}{m}\right\rceil m}^{F_m})] \tag{5.2.21}$$

where $\sum_{i=1}^{m} n_i=n$, $\left\lceil\frac{n_i}{i}\right\rceil i$ is as described in Lemma 5.2.7, jobs $J_{n_i+1}^i,\ldots,J_{\left\lceil\frac{n_i}{i}\right\rceil i}^i$, $1\leq i\leq m$ have zero execution time requirements and $\lambda_i=\max_{J_j\in F_i}\left\{\frac{\sigma_j}{\tau_j}\right\}$.

Proof: Let the permutation $\alpha=(\alpha^1_1,\alpha^1_2,\ldots,\alpha^1_{n_1},\alpha^2_1,\ldots,\alpha^2_{n_2},\ldots,\alpha^m_1,\ldots,\alpha^m_{n_m})$ produces a priority list

$$L = (J^1_{\alpha^1_{n_1}},J^1_{\alpha^1_{n_1}-1},\ldots,J^1_{\alpha^1_1},J^2_{\alpha^2_{n_2}},\ldots,J^2_{\alpha^2_1},\ldots,J^m_{\alpha^m_{n_m}},\ldots,J^m_{\alpha^m_1})$$

so that, $\tau^{F_i}_{\alpha^i_{n_i}} \leq \tau^{F_i}_{\alpha^i_{n_i}-1} \leq \ldots \leq \tau^{F_i}_{\alpha^i_1}$ for $1\leq i\leq m$.

According to Lemma 5.2.7 we have

$$\bar{\omega}_{OPT} \geq (\tau^{F_1}_{\alpha^1_1}+2\tau^{F_1}_{\alpha^1_2}+\ldots+n_1\tau^{F_1}_{\alpha^1_{n_1}})+[(\tau^{F_2}_{\alpha^2_1}+\tau^{F_2}_{\alpha^2_2})+2(\tau^{F_2}_{\alpha^2_3}+\tau^{F_2}_{\alpha^2_4})+\ldots+\left\lceil\frac{n_2}{2}\right\rceil(\tau^{F_2}_{\alpha^2_{\lfloor\frac{n_2}{2}\rfloor 2+1}}+\tau^{F_2}_{\alpha^2_{\lceil\frac{n_2}{2}\rceil 2}})]$$

$$+\ldots+[(\tau^{F_m}_{\alpha^m_1}+\ldots+\tau^{F_m}_{\alpha^m_m})+\ldots+\left\lceil\frac{n_m}{m}\right\rceil(\tau^{F_m}_{\alpha^m_{\lfloor\frac{n_m}{m}\rfloor m+1}}+\ldots+\tau^{F_m}_{\alpha^m_{\lceil\frac{n_m}{m}\rceil m}})] \quad . \qquad (5.2.22)$$

But, $\tau^{F_i}_{\alpha^i_j} \geq \frac{\sigma^{F_i}_{\alpha_j}}{\lambda_i}$ for $1\leq i\leq m$ and $1\leq\alpha^i_j\leq n_i$. Thus

$$(\tau^{F_i}_{\alpha^i_1}+\ldots+\tau^{F_i}_{\alpha^i_i})+\ldots+\left\lceil\frac{n_i}{i}\right\rceil(\tau^{F_i}_{\alpha^i_{\lfloor\frac{n_i}{i}\rfloor i+1}}+\ldots+\tau^{F_i}_{\alpha^i_{\lceil\frac{n_i}{i}\rceil i}}) >$$

$$\frac{1}{\lambda_i}[(\sigma^{F_i}_{\alpha^i_1}+\ldots+\sigma^{F_i}_{\alpha^i_i})+\ldots+\left\lceil\frac{n_i}{i}\right\rceil(\sigma^{F_i}_{\alpha^i_{\lfloor\frac{n_i}{i}\rfloor i+1}}+\ldots+\sigma^{F_i}_{\alpha^i_{\lceil\frac{n_i}{i}\rceil i}})].$$

However, the sum of the right hand side in the last inequality is minimised when $\sigma^{F_i}_{\alpha^i_j}$, $1\leq\alpha^i_j\leq\left\lceil\frac{n_i}{i}\right\rceil i$, are in non-increasing order. So,

$$(\tau^{F_i}_{\alpha^i_1}+\ldots+\tau^{F_i}_{\alpha^i_i})+\ldots+\left\lceil\frac{n_i}{i}\right\rceil(\tau^{F_i}_{\alpha^i_{\lfloor\frac{n_i}{i}\rfloor i+1}}+\ldots+\tau^{F_i}_{\alpha^i_{\lceil\frac{n_i}{i}\rceil i}}) >$$

$$\frac{1}{\lambda_i}[(\sigma^{F_i}_1+\ldots+\sigma^{F_i}_i)+\ldots+\left\lceil\frac{n_i}{i}\right\rceil(\sigma^{F_i}_{\lfloor\frac{n_i}{i}\rfloor i+1}+\ldots+\sigma^{F_i}_{\lceil\frac{n_i}{i}\rceil i})] \quad , \qquad (5.2.23)$$

where $1\leq i\leq m$.

Therefore, the inequality (5.2.22) because of the inequalities (5.2.23) results in the inequality (5.2.21), which proves the Lemma¤

Lemma 5.2.9: Let a task system $(J,[t_{ij}])$ with n independent jobs be scheduled on a heterogeneous multiprocessor system with m independent memories. Also, let $\bar{\omega}_{QAD}$ be the mean flow time of the schedule constructed by the Q.A.D. algorithm, when the priority list is formed by any heuristic ordering rule. Then, for uniform or non-identical processors, we have

$$\bar{\omega}_{QAD} \leq \tau_1 + 2\tau_2 + \ldots + n\tau_n \ .$$

Proof: Let c_j, $1 \leq j \leq n$, be the contribution of job J_j to $\bar{\omega}_{QAD}$. So, $\bar{\omega}_{QAD} = c_1 + c_2 + \ldots + c_n$. Therefore, if we show that $c_j \leq j\tau_j$ for j=1(1)n then the lemma is proven.

From the definition of the mean flow time, when job J_j is scheduled on the i^{th} processor, we have

$$c_j = h_i t_{ij} \ ,$$

where h_i is one greater than the number of tasks following J_j on the i^{th} processor.

Now, suppose that τ_j occurs on the k^{th} processor, $1 \leq k \leq \ell_j$ (i.e. $\tau_j = t_{kj}$). From the nature of the Q.A.D. algorithm, we have

$$c_j = h_i t_{ij} \leq h_k t_{kj} \ .$$

But, since always $h_k \leq j$, from the last inequality we obtain $c_j \leq j\tau_j$, which proves the lemma. Furthermore, it is easy to see that the equality holds only if all τ_j, $1 \leq j \leq n$ occur on the same processor and the jobs have been scheduled on this processor as well¤

(Note: This lemma can apply to heterogeneous multiprocessor systems without any independent memories. However in that case, Clark [Cℓ] has shown this lemma to be true only when the priority list is formed by the LTF_{MIN} ordering rule.)

5.3 PRIORITY DRIVEN (P.D.) SCHEDULING ALGORITHMS

This section establishes worst-case performance bounds for the P.D. algorithm under several ordering procedures. As was stated in the introduction of the present chapter, first we shall establish an upper bound when an arbitrary ordering rule is used to construct the priority list.

Let $\bar{\omega}_{PD}$ be the mean flow time of the schedule constructed by the P.D. algorithm, when the priority list is formed by a heuristic ordering procedure, and $\bar{\omega}_{OPT}$ be the mean flow time of an optimal schedule for a given task system $(J,[t_{ij}])$, $1\leq i\leq m$, $1\leq j\leq n$.

Theorem 5.3.1: Let the priority list be in an arbitrary ordering (RAND). Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \max_{1\leq i\leq m} \{\lambda_i'\, n_i'\} \quad ;$$

and (ii) for uniform processors:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \max_{1\leq i\leq m} \{\beta_i'\, n_i'\} \quad ,$$

where $n=n_1'+n_2'+\ldots+n_m'$, $\lambda_i' = \max_{J_j\in G_i} \left\{\frac{\sigma_j}{\tau_j}\right\}$

$$\beta_i' = \max_{1\leq k\leq \nu_i} \{b_k\}/b_i \text{ and } \qquad \nu_i = \max_{J_j\in G_i} \{\ell_j\}$$

for i=1(1)m.

Proof: (i) Let $\bar{\omega}_1,\bar{\omega}_2,\ldots,\bar{\omega}_m$ be the contribution to the mean flow time of the jobs J_j, where $J_j\in G_i$ for i=1(1)m respectively. So, immediately we have

$$\bar{\omega}_{PD} = \bar{\omega}_1+\bar{\omega}_2+\ldots+\bar{\omega}_m \quad . \tag{5.3.1}$$

But, the contribution of the jobs scheduled on P_1 is always

$$\left.\begin{aligned}
\bar{\omega}_1 &< \lambda_1'(n_1'\tau_1^{G_1}+(n_1'-1)\tau_2^{G_1}+\ldots+2\tau_{n_1'-1}^{G_1}+\tau_{n_1'}^{G_1}) \quad .\\
\text{Similarly,}\quad \bar{\omega}_2 &< \lambda_2'(n_2'\tau_1^{G_2}+(n_2'-1)\tau_2^{G_2}+\ldots+2\tau_{n_2'-1}^{G_2}+\tau_{n_2'}^{G_2})\\
&\vdots\\
\text{and}\quad \bar{\omega}_m &< \lambda_m'(n_m'\tau_1^{G_m}+(n_m'-1)\tau_2^{G_m}+\ldots+2\tau_{n_{m-1}'}^{G_m}+\tau_{n_m'}^{G_m}) \quad .
\end{aligned}\right\} \tag{5.3.2}$$

Therefore, because of the inequalities (5.3.2) the equality (5.3.1)

becomes

$$\bar{\omega}_{PD}<\lambda_1'(n_1'\tau_1^{G_1}+(n_1'-1)\tau_2^{G_1}+\ldots+\tau_{n_1'}^{G_1})+\lambda_2'(n_2'\tau_1^{G_2}+(n_2'-1)\tau_2^{G_2}+\ldots+\tau_{n_2'}^{G_2})$$

$$+\ldots+\lambda_m'(n_m'\tau_1^{G_m}+(n_m'-1)\tau_2^{G_m}+\ldots+\tau_{n_m'}^{G_m}) \quad . \tag{5.3.3}$$

From Lemma 5.2.4 we have

$$\bar{\omega}_{OPT}\geq\tau_1^{G_1}+\tau_2^{G_1}+\ldots+\tau_{n_1'}^{G_1}+\tau_1^{G_2}+\ldots+\tau_{n_2'}^{G_2}+\ldots+\tau_1^{G_m}+\ldots+\tau_{n_m'}^{G_m} \quad , \tag{5.3.4}$$

since $n=n_1'+n_2'+\ldots+n_m'$.

Because λ_1', $\tau_j^{G_i}\in R^+$ for $j=1(1)n_i'$ and $i=1(1)m$, then from (5.3.3) and (5.3.4)

the following inequality can be derived

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \frac{\sum_{i=1}^{m}\lambda_1'\left[\sum_{j=1}^{n_i'}((n_i'+1-j)\tau_j^{G_i})\right]}{\sum_{i=1}^{m}\sum_{j=1}^{n_i'}\tau_j^{G_i}} \quad . \tag{5.3.5}$$

It is easy to see that

$$\left.\begin{aligned}
&\frac{\lambda_1'(n_1'\tau_1^{G_1}+(n_1'-1)\tau_2^{G_1}+\ldots+2\tau_{n_i'-1}^{G_1}+\tau_{n_1'}^{G_1})}{\tau_1^{G_1}+\tau_2^{G_1}+\ldots+\tau_{n_1'}^{G_1}} < \lambda_1'\,n_1' \\
&\qquad\vdots \\
&\frac{\lambda_m'(n_m'\tau_1^{G_m}+(n_m'-1)\tau_2^{G_m}+\ldots+\tau_{n_m'}^{G_m})}{\tau_1^{G_m}+\tau_2^{G_m}+\ldots+\tau_{n_m'}^{G_m}} < \lambda_m'\,n_m' \quad .
\end{aligned}\right\} \tag{5.3.6}$$

and

Thus, because of (5.3.6) we can obtain from Lemma 5.2.2 for $\nu=m$, $\ell=i$, $d_\ell=n_i'$ for $i=1(1)m$, $g_\ell=\frac{1}{\lambda_i'}$, and $c_{s+j}=n_i'+1-j$, $a_{s+j}=\tau_j^{G_i}$ for $j=1(1)n_i'$ the inequality

$$\frac{\sum_{i=1}^{m}\lambda_i'\left[\sum_{j=1}^{n_i'}((n_i'+1-j)\tau_j^{G_i})\right]}{\sum_{i=1}^{m}\sum_{j=1}^{n_i'}\tau_j^{G_i}} < \max_{1\leq i\leq m}\{\lambda_i'n_i'\} \quad . \tag{5.3.7}$$

From inequalities (5.3.5) and (5.3.7) one can show that

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \max_{1\leq i\leq m} \{\lambda_i! n_i!\} ,$$

which proves part (i) of the theorem.

(ii) In the case of uniform processors, instead of the inequalities (5.3.2), we have

$$\bar{\omega}_i < \left\lceil \max_{1\leq k\leq \nu_i} \{b_k\}/b_i \right\rceil (n_i! \tau_1^{G_i} + (n_i!-1)\tau_2^{G_i} + \ldots + \tau_{n_i!}^{G_i})$$

for i=1(1)m. Afterwards, using the same analysis we find

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \max_{1\leq i\leq m} \{\beta_i! n_i!\} ,$$

which proves part (ii)¤

However, this bound, when $\lambda_i!=1=\beta_i!$, $1\leq i\leq m$, agrees with the one found for the homogeneous multiprocessor system with independent memories. (See Theorem I.1, Appendix I.)

Between the two extremes of using an optimally chosen priority list and of using a completely arbitrary priority list, there is the possibility of using priority lists obtained by simple heuristic procedures. Through the remaining theorems of this section we shall see if the heuristic procedures chosen in this thesis to construct priority lists can produce better worst-case bounds, relative to the one already found in Theorem 5.3.1, when a completely arbitrary priority list is used.

Theorem 5.3.2: Let the priority list be in a LMF, LTF_{MIN}, LTF_{MAX}, $LMLT_{MIN}$ or $LMLT_{MAX}$ ordering. Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \max_{1\leq i\leq m} \{\lambda_i! n_i!\} ;$$

and (ii) for uniform processors:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \max_{1\leq i\leq m} \{\beta_i! n_i!\} ,$$

where λ_i' and β_i' are as defined in Theorem 5.3.1.

Proof: Inequalities (5.3.2) as well as (5.3.4) are valid when the priority list is constructed by the LMF, LTF_{MIN}, LTF_{MAX}, $LMLT_{MIN}$ or $LMLT_{MAX}$ ordering rule. This means that finally, the upper bounds of Theorem 5.3.1 are obtained.

Theorem 5.3.2 indicates that the simple ordering procedures LMF, LTF_{MIN}, LTF_{MAX}, $LMLT_{MIN}$ and $LMLT_{MAX}$ offer no improvement in a worst-case sense over an arbitrary ordering for the P.D. algorithm, when the mean flow time is the performance criterion.

Now, let us observe the bounds presented in Theorems 5.3.1 and 5.3.2. It is clear that, if the number of processors increases and the number of jobs in the task system remains constant or decreases then the value of the worst-case performance bound of the P.D. algorithm under the considered ordering rules decreases, since $n'_{max} = \max_{1\leq i\leq m} \{n_i'\}$ decreases as well. In a vice-versa situation the value of the worst-case bound increases. Nevertheless, this is not an absolutely accurate interpretation of the bound, because it also depends on the λ_i' variable. So, few increases or decreases in the number of processors or jobs in the task system might result in the behaviour of the algorithm to be slightly different to the one already presented.

Now, we turn to examine the STF ordering procedures.

Theorem 5.3.3: Let the priority list be in a STF_{MAX} ordering. Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{ \max_{1\leq i\leq m} \left\{\lambda_i' \left[\nu_i - \frac{(\nu_i - 1)}{2\left\lceil \frac{n_i'}{\nu_i} \right\rceil}\right]\right\} , \max_{1\leq i\leq m} \left\{\lambda_i' \left[\frac{n_i'+1}{2}\right]\right\}\right\} \quad ;$$

and (ii) for uniform processors:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{ \max_{1\leq i\leq m} \left\{\beta_i' \left[\nu_i - \frac{(\nu_i - 1)}{2\left\lceil \frac{n_i'}{\nu_i} \right\rceil}\right]\right\} , \max_{1\leq i\leq m} \left\{\beta_i' \left[\frac{n_i'+1}{2}\right]\right\}\right\} \quad ,$$

where λ_i', β_i' and ν_i are as defined in Theorem 5.3.1.

Proof: (i) Let $\bar{\omega}_1,\bar{\omega}_2,\ldots,\bar{\omega}_m$ be the contribution of the jobs J_j, where $J_j \in G_i$ for i=1(1)m respectively, to the mean flow time of the schedule which is constructed by a P.D. algorithm while the priority list is formed by the STF_{MAX} ordering rule. It is always

$$\left.\begin{aligned}
\bar{\omega}_1 &\leq n_1'\sigma_{n_1'}^{G_1} + (n_1'-1)\sigma_{n_1'-1}^{G_1} + \ldots + \sigma_1^{G_1} \,, \\
\bar{\omega}_2 &\leq n_2'\sigma_{n_2'}^{G_2} + (n_2'-1)\sigma_{n_2'-1}^{G_2} + \ldots + \sigma_1^{G_2} \,, \\
&\vdots \\
\text{and} \quad \bar{\omega}_m &\leq n_m'\sigma_{n_m'}^{G_m} + (n_m'-1)\sigma_{n_m'-1}^{G_m} + \ldots + \sigma_1^{G_m} \,,
\end{aligned}\right\} \tag{5.3.8}$$

where $\sigma_{n_i'}^{G_i} \leq \sigma_{n_{i-1}'}^{G_i} \leq \ldots \leq \sigma_1^{G_i}$ for i=1(1)m, due to the nature of the priority list.

But,

$$\bar{\omega}_{PD} = \bar{\omega}_1 + \bar{\omega}_2 + \ldots + \bar{\omega}_m$$

or because of the inequalities (5.3.8)

$$\bar{\omega}_{PD} \leq \sum_{i=1}^{m} (n_i'\sigma_{n_i'}^{G_i} + (n_i'-1)\sigma_{n_i'-1}^{G_i} + \ldots + \sigma_1^{G_i}) \,. \tag{5.3.9}$$

Furthermore, let $\bar{\omega}_0^1,\bar{\omega}_0^2,\ldots,\bar{\omega}_0^m$ be the contribution to the optimal mean flow time of the jobs J_j, where $J_j \in G_i$ for i=1(1)m respectively. Also, let us suppose that the jobs $J_j \in G_i$, $1 \leq i \leq m$, are the only ones in the task system. Then, from Lemma 5.2.6 we get

$$\left.\begin{aligned}
\bar{\omega}_0^1 &> \frac{1}{\lambda_1'}\left[(\sigma_1^{G_1}+\sigma_2^{G_1}+\ldots+\sigma_{\nu_1}^{G_1}) + 2(\sigma_{\nu_1+1}^{G_1}+\ldots+\sigma_{2\nu_1}^{G_1}) + \left\lceil\frac{n_1'}{\nu_1}\right\rceil\left(\sigma_{\left\lfloor\frac{n_1'}{\nu_1}\right\rfloor\nu_1+1}^{G_1}+\ldots+\sigma_{\left\lceil\frac{n_1'}{\nu_1}\right\rceil\nu_1}^{G_1}\right)\right] \\
\bar{\omega}_0^2 &> \frac{1}{\lambda_2'}\left[(\sigma_1^{G_2}+\sigma_2^{G_2}+\ldots+\sigma_{\nu_2}^{G_2}) + 2(\sigma_{\nu_2+1}^{G_2}+\ldots+\sigma_{2\nu_2}^{G_2}) + \ldots + \left\lceil\frac{n_2'}{\nu_2}\right\rceil\left(\sigma_{\left\lfloor\frac{n_2'}{\nu_2}\right\rfloor\nu_2+1}^{G_2}+\ldots+\sigma_{\left\lceil\frac{n_2'}{\nu_2}\right\rceil\nu_2}^{G_2}\right)\right] \\
&\vdots \\
\bar{\omega}_0^m &> \frac{1}{\lambda_m'}\left[(\sigma_1^{G_m}+\sigma_2^{G_m}+\ldots+\sigma_m^{G_m}) + 2(\sigma_{m+1}^{G_m}+\ldots+\sigma_{2m}^{G_m}) + \ldots + \left\lceil\frac{n_m'}{m}\right\rceil\left(\sigma_{\left\lfloor\frac{n_m'}{m}\right\rfloor m+1}^{G_m}+\ldots+\sigma_{\left\lceil\frac{n_m'}{m}\right\rceil m}^{G_m}\right)\right]
\end{aligned}\right\} \tag{5.3.10}$$

where $\sigma^{G_i}_{n_i'+1}=\ldots+\sigma^{G_i}_{\left\lceil \frac{n_i'}{\nu_i} \right\rceil \nu_i}=0$, $i \leq \nu_i \leq m$ for $i=1(1)m$. Now,

$$\bar{\omega}_{OPT} = \bar{\omega}_0^1 + \bar{\omega}_0^2 + \ldots + \bar{\omega}_0^m$$

or because of the inequalities (5.3.10)

$$\bar{\omega}_{OPT} > \sum_{i=1}^{m} \frac{1}{\lambda_i'} \left[(\sigma_1^{G_i}+\sigma_2^{G_i}+\ldots+\sigma_{\nu_i}^{G_i}) + 2(\sigma_{\nu_i+1}^{G_i}+\ldots+\sigma_{2\nu_i}^{G_i}) + \ldots + \left\lceil \frac{n_i'}{\nu_i} \right\rceil (\sigma_{\left\lfloor \frac{n_i'}{\nu_i} \right\rfloor \nu_i+1}^{G_i} + \ldots + \sigma_{\left\lceil \frac{n_i'}{\nu_i} \right\rceil \nu_i}^{G_i}) \right] \tag{5.3.11}$$

From inequalities (5.3.9) and (5.3.11) we have

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \frac{\sum_{i=1}^{m} \sum_{r=1}^{\lceil n_i'/\nu_i \rceil} \sum_{j=1}^{\nu_i} ((r-1)\nu_i+j)\sigma_{(r-1)\nu_i+j}^{G_i}}{\sum_{i=1}^{m} \frac{1}{\lambda_i'} \left[\sum_{r=1}^{\lceil n_i'/\nu_i \rceil} r \sum_{j=1}^{\nu_i} \sigma_{(r-1)\nu_i+j}^{G_i} \right]} \tag{5.3.12}$$

However, using Lemma 5.2.1 with $k=0$, $d=\nu_i$, $\ell=r$ and $a_{s+j}=\sigma_{(r-1)\nu_i+j}^{G_i}$, $1 \leq j \leq m$, for $r=1(1)\left\lceil \frac{n_i'}{\nu_i} \right\rceil$ and then applying Lemma 5.2.2 we get

$$\frac{\sum_{r=1}^{\lceil n_i'/\nu_i \rceil} \sum_{j=1}^{\nu_i} ((r-1)\nu_i+j)\sigma_{(r-1)\nu_i+j}^{G_i}}{\sum_{r=1}^{\lceil n_i'/\nu_i \rceil} r \sum_{j=1}^{\nu_i} \sigma_{(r-1)\nu_i+j}^{G_i}} < \nu_i - \frac{(\nu_i-1)}{2\left\lceil \frac{n_i'}{\nu_i} \right\rceil}$$

or multiplying both sides by λ_i'

$$\frac{\sum_{r=1}^{\lceil n_i'/\nu_i \rceil} \sum_{j=1}^{\nu_i} ((r-1)\nu_i+j)\sigma_{(r-1)\nu_i+j}^{G_i}}{\frac{1}{\lambda_i'} \sum_{r=1}^{\lceil n_i'/\nu_i \rceil} r \sum_{j=1}^{\nu_i} \sigma_{(r-1)\nu_i+j}^{G_i}} < \lambda_i' \left(\nu_i - \frac{(\nu_i-1)}{2\left\lceil \frac{n_i'}{\nu_i} \right\rceil} \right) \tag{5.3.13}$$

for $i=1(1)m$.

Again, applying Lemma 5.2.2 for the inequalities (5.3.13), we obtain

$$\frac{\sum_{i=1}^{m}\sum_{r=1}^{\lceil n_i'/\nu_i\rceil}\sum_{j=1}^{\nu_i}((r-1)\nu_i+j)\sigma^{G_i}_{(r-1)\nu_i+j}}{\sum_{i=1}^{m}\frac{1}{\lambda_i'}\left\lfloor\sum_{r=1}^{\lceil n_i'/\nu_i\rceil} r\sum_{j=1}^{\nu_i}\sigma^{G_i}_{(r-1)\nu_i+j}\right\rfloor} < \max_{1\le i\le m}\left\{\lambda_i'\left(\nu_i-\frac{(\nu_i-1)}{2\left\lceil\frac{n_i'}{\nu_i}\right\rceil}\right)\right\} \tag{5.3.14}$$

Finally, because of the inequalities (5.3.12) and (5.3.14)

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \max_{1\le i\le m}\left\{\lambda_i'\left(\nu_i-\frac{(\nu_i-1)}{2\left\lceil\frac{n_i'}{\nu_i}\right\rceil}\right)\right\} \quad . \tag{5.3.15}$$

Another way to find an upper bound is to use Lemma 5.2.4 to bound the optimal mean flow time, instead of Lemma 5.2.6 as used previously. So,

$$\bar{\omega}_0^1 > \frac{1}{\tau_{\alpha_1}^{G_1}+\tau_{\alpha_2}^{G_1}+\ldots+\tau_{\alpha_{n_1'}}^{G_1}}$$

or because of $\tau_{\alpha_j}^{G_1} > \frac{\sigma_{\alpha_j}^{G_1}}{\lambda_1'}$, $1\le j\le n_1'$, and Lemma 5.2.3

$$\left.\begin{aligned}\bar{\omega}_0^1 &> \frac{1}{\lambda_1'}(\sigma_1^{G_1}+\sigma_2^{G_1}+\ldots+\sigma_{n_1'}^{G_1}).\\ \text{Similarly,}\\ \bar{\omega}_0^2 &> \frac{1}{\lambda_2'}(\sigma_1^{G_2}+\sigma_2^{G_2}+\ldots+\sigma_{n_2'}^{G_2}),\\ &\vdots\\ \text{and}\quad \bar{\omega}_0^m &> \frac{1}{\lambda_m'}(\sigma_1^{G_m}+\sigma_2^{G_m}+\ldots+\sigma_{n_m'}^{G_m}),\end{aligned}\right\} \tag{5.3.10'}$$

where $\sigma_1^{G_i}\ge\sigma_2^{G_i}\ge\ldots\ge\sigma_{n_i'}^{G_i}$ for i=1(1)m.

Since $\bar{\omega}_{OPT} = \bar{\omega}_0^1 + \bar{\omega}_0^2 + \ldots + \bar{\omega}_0^m$, then

$$\bar{\omega}_{OPT} > \sum_{i=1}^{m}\frac{1}{\lambda_i'}(\sigma_1^{G_i}+\sigma_2^{G_i}+\ldots+\sigma_{n_i'}^{G_i}). \tag{5.3.11'}$$

From the inequalities (5.3.9) and (5.3.11') we get

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \frac{\sum_{i=1}^{m} \sum_{j=1}^{n_i'} j\sigma_j^{G_i}}{\sum_{i=1}^{m} \frac{1}{\lambda_i'} \left[\sum_{j=1}^{n_i' G_i} \sigma_j \right]} \quad . \tag{5.3.12'}$$

On the other hand, Lemma 5.2.1 for k=0, $d=n_i'$, s=0, ℓ=1 and $a_{s+j}=\sigma_j^{G_i}$ for j=1(1)n_i' will give us

$$\frac{\sum_{j=1}^{n_i'} j\sigma_j^{G_i}}{\sum_{j=1}^{n_i'} \sigma_j^{G_i}} \leq \frac{n_i'+1}{2}$$

or multiplying by λ_i' both sides

$$\frac{\sum_{j=1}^{n_i'} j\sigma_j^{G_i}}{\frac{1}{\lambda_i'} \sum_{j=1}^{n_i' G_i} \sigma_j} \leq \lambda_i' \frac{(n_i'+1)}{2}, \quad i=1(1)m.$$

Now, applying Lemma 5.2.2 for the above inequalities we obtain

$$\frac{\sum_{i=1}^{m} \sum_{j=1}^{n_i'} j\sigma_j^{G_i}}{\sum_{i=1}^{m} \frac{1}{\lambda_i'} \left[\sum_{j=1}^{n_i' G_i} \sigma_j \right]} < \max_{1 \leq i \leq m} \left\{ \lambda_i' \left(\frac{n_i'+1}{2} \right) \right\} . \tag{5.3.14'}$$

Thus, the inequality (5.3.12') because of (5.3.14') becomes

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \max_{1 \leq i \leq m} \left\{ \lambda_i' \left(\frac{n_i'+1}{2} \right) \right\} \quad . \tag{5.3.15'}$$

Therefore, because of the inequalities (5.3.15) and (5.3.15'), the worst-case bound of the P.D. algorithm when the priority list is formed by the STF_{MAX} rule is:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min \left\{ \max_{1 \leq i \leq m} \left\{ \lambda_i' \left(\nu_i - \frac{(\nu_i - 1)}{2 \left\lceil \frac{n_i'}{\nu_i} \right\rceil} \right) \right\}, \max_{1 \leq i \leq m} \left\{ \lambda_i' \left(\frac{n_i'+1}{2} \right) \right\}, \right\}$$

which proves part (i) of the theorem.

(ii) For the case of uniform processors the inequalities (5.3.8) must be substituted by

$$\bar{\omega}_1 \leq \left(\min_{1\leq k\leq \nu_1} \{b_k\}/b_1\right)\left(n_1'\sigma_{n_1'}^{G_1} + (n_1'-1)\sigma_{n_1'-1}^{G_1} + \ldots + \sigma_1^{G_1}\right)$$

$$\bar{\omega}_2 \leq \left(\min_{1\leq k\leq \nu_2} \{b_k\}/b_2\right)\left(n_2'\sigma_{n_2'}^{G_2} + (n_2'-1)\sigma_{n_2'-1}^{G_2} + \ldots + \sigma_1^{G_2}\right)$$

$$\vdots$$

$$\bar{\omega}_m \leq \left(\min_{1\leq k\leq \nu_m} \{b_k\}/b_m\right)\left(n_m'\sigma_{n_m'}^{G_m} + (n_m'-1)\sigma_{n_m'-1}^{G_m} + \ldots + \sigma_1^{G_m}\right)$$

and λ_i', $1\leq i\leq m$, in the inequalities (5.3.10) by $\max_{1\leq k\leq \nu_i} \{b_k\}/\min_{1\leq k\leq \nu_i} \{b_k\}$. Then, using the same analysis (i.e., the one used to prove part (i)) we get eventually,

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{\max_{1\leq i\leq m}\left\{\beta_i'\left[\nu_i - \frac{(\nu_i-1)}{2\left\lceil \frac{n_i'}{\nu_i}\right\rceil}\right]\right\}, \max_{1\leq i\leq m}\left\{\beta_i'\left[\frac{n_i'+1}{2}\right]\right\},\right\}$$

which proves part (ii) ¤

Theorem 5.3.4: Let the priority list be in a STF_{MIN} ordering. Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{\max_{1\leq i\leq m}\left\{\max_{1\leq r\leq \left\lceil \frac{n_i'}{\nu_i}\right\rceil}\left\{\lambda_i'^{r}\left[\nu_i - \frac{(\nu_i-1)}{2r}\right]\right\}\right\}, \max_{1\leq i\leq m}\left\{\lambda_i'\left[\frac{n_i'+1}{2}\right]\right\}\right\};$$

(ii) for uniform processors:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{\max_{1\leq i\leq m}\left\{\max_{1\leq r\leq \left\lceil \frac{n_i'}{\nu_i}\right\rceil}\left\{\beta_i'^{r}\left[\nu_i - \frac{(\nu_i-1)}{2r}\right]\right\}\right\}, \max_{1\leq i\leq m}\left\{\beta_i'\left[\frac{n_i'+1}{2}\right]\right\}\right\},$$

where λ_i', β_i' and ν_i are as defined in Theorem 5.3.1,

$$\lambda_i'^{r} = \max_{J_j \in G_i^r} \left\{ \frac{\sigma_j}{\tau_j} \right\} \quad \text{and} \quad \beta_i'^{r} = \max_{\substack{1 \le k \le \ell_j \\ J_j \in G_i^{r_j}}} \{b_k\}/b_i$$

for $r=1(1)\left\lceil \frac{n_i'}{\nu_i} \right\rceil$ and $i=1(1)m$.

Proof: Let $\bar{\omega}_1, \bar{\omega}_2, \ldots, \bar{\omega}_m$ be respectively the contribution of the jobs $J_j \in G_i$, $i=1(1)m$, to the mean flow time of the schedule which is constructed by the P.D. algorithm while the priority list was formed by the STF_{MIN} rule. Also, let $\bar{\omega}_0^1, \bar{\omega}_0^2, \ldots, \bar{\omega}_0^m$ be the contribution of jobs $J_j \in G_i$, $i=1(1)m$, to the the optimal mean flow time.

Then, for non-identical processors we have

$$\bar{\omega}_i < \lambda_i'^{1}(\tau_1^{G_i}+2\tau_2^{G_i}+\ldots+\nu_i\tau_{\nu_i}^{G_i})+\lambda_i'^{2}((\nu_i+1)\tau_{\nu_i+1}^{G_i}+\ldots+2\nu_i\tau_{2\nu_i}^{G_i})+$$

$$\ldots+\lambda_i'^{\left\lceil \frac{n_i'}{\nu_i} \right\rceil}\left((\left\lfloor \frac{n_i'}{\nu_i} \right\rfloor \nu_i+1)\tau_{\left\lfloor \frac{n_i'}{\nu_i} \right\rfloor \nu_i+1}^{G_i}+\ldots+\left\lceil \frac{n_i'}{\nu_i} \right\rceil \nu_i\, \tau_{\left\lceil \frac{n_i'}{\nu_i} \right\rceil \nu_i}^{G_i}\right), \qquad (5.3.16)$$

where $\tau_1^{G_i} \gtrsim \tau_2^{G_i} \gtrsim \ldots \gtrsim \tau_{n_i'}^{G_i}$ and $\tau_{n_i'+1}^{G_i}=\ldots=\tau_{\left\lceil \frac{n_i'}{\nu_i} \right\rceil \nu_i}^{G_i}=0$ for $i=1(1)m$;

assuming that the jobs $J_j \in G_i$, $1 \lesssim i \lesssim m$, are the only ones on the task system then because of Lemma 5.2.5,

$$\bar{\omega}_0^i \gtrsim (\tau_1^{G_i}+\tau_2^{G_i}+\ldots+\tau_{\nu_i}^{G_i})+2(\tau_{\nu_i+1}^{G_i}+\ldots+\tau_{2\nu_i}^{G_i})+\ldots+\left\lceil \frac{n_i'}{\nu_i} \right\rceil \left(\tau_{\left\lfloor \frac{n_i'}{\nu_i} \right\rfloor \nu_i+1}^{G_i}+\ldots+\tau_{\left\lceil \frac{n_i'}{\nu_i} \right\rceil \nu_i}^{G_i}\right) \qquad (5.3.17)$$

for $i=1(1)m$.

However, $\bar{\omega}_i$ and $\bar{\omega}_0^i$ for $i=1(1)m$ can also be bounded by

$$\bar{\omega}_i < \lambda_i'(\tau_1^{G_i}+2\tau_2^{G_i}+\ldots+n_i'\tau_{n_i'}^{G_i}) \qquad (5.3.18)$$

and

$$\bar{\omega}_0^i \gtrsim \tau_1^{G_i}+\tau_2^{G_i}+\ldots+\tau_{n_i'}^{G_i}, \qquad (5.3.19)$$

because of Lemma 5.2.4.

On the other hand, in the case of uniform processors the inequalities (5.3.16) and (5.3.18) must be substituted by

$$\bar{\omega}_i < \beta_i^{'1}(\tau_1^{G_i}+2\tau_2^{G_i}+\ldots+\nu_i\tau_{\nu_i}^{G_i})+\beta_i^{'2}((\nu_i+1)\tau_{\nu_1+1}^{G_i}+\ldots+2\nu_i\tau_{2\nu_i}^{G_i})+$$

$$\ldots+\beta_i^{'\left\lceil\frac{n_i'}{\nu_i}\right\rceil}\left((\left\lfloor\frac{n_i'}{\nu_i}\right\rfloor\nu_i+1)\tau_{\left\lfloor\frac{n_i'}{\nu_i}\right\rfloor\nu_i+1}^{G_i}+\ldots+\left\lceil\frac{n_i'}{\nu_i}\right\rceil\tau_{\left\lfloor\frac{n_i'}{\nu_i}\right\rfloor\nu_i}^{G_i}\right) \qquad (5.3.16')$$

and

$$\bar{\omega}_i < \beta_i'(\tau_1^{G_i}+2\tau_2^{G_i}+\ldots+n_i'\tau_{n_i'}^{G_i}) \qquad (5.3.18')$$

respectively, whereas (5.3.17) and (5.3.19) remain the same.

Now, following a similar analysis with the one used previously to prove Theorem 5.3.3 we find,

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{\max_{1\le i\le m}\left\{\max_{1\le r\le\left\lceil\frac{n_i'}{\nu_i}\right\rceil}\left\{\lambda_i^{'r}\left(\nu_i-\frac{(\nu_i-1)}{2r}\right)\right\}\right\},\ \max_{1\le i\le m}\left\{\lambda_i'\left(\frac{n_i'+1}{2}\right)\right\}\right\}$$

for non-identical processors and

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{\max_{1\le i\le m}\left\{\max_{1\le r\le\left\lceil\frac{n_i'}{\nu_i}\right\rceil}\left\{\beta_i^{'r}\left(\nu_i-\frac{(\nu_i-1)}{2r}\right)\right\}\right\},\ \max_{1\le i\le m}\left\{\beta_i'\left(\frac{n_i'+1}{2}\right)\right\}\right\}$$

for uniform processors, which proves Theorem 5.3.4[¤]

The bounds given by this theorem as well as by the previous one (Theorem 5.3.3) for $\lambda_i'=\lambda_i^{'r}=1=\beta_i'=\beta_i^{'r}$, $1\le r\le\left\lceil\frac{n_i'}{\nu_i}\right\rceil$ and $1\le i\le m$, agree with the corresponding worst-case bound found for the homogeneous multiprocessor system with independent memories. (See Theorem I.2, Appendix I.)

The second factor of the bounds of Theorem 5.3.3 and 5.3.4 might be less than the first one if $n'_{max}\le\left\lfloor\frac{3m}{2}\right\rfloor-1$, where $n'_{max}=\max_{1\le i\le m}\{n_i'\}$. This is because $\left(m-\frac{(m-1)}{2\left\lceil\frac{n_i'}{m}\right\rceil}\right)\ge\left(\frac{n_i'+1}{2}\right)$ when $n_i'\le\left\lfloor\frac{3m}{2}\right\rfloor-1$. The last statement can be verified by an induction process.

Generally, Theorem 5.3.3 and 5.3.4 indicate that the STF_{MAX} and STF_{MIN} ordering procedures, in contrast to the previous analysed ones, can offer improvement over the extreme performance of an arbitrary ordering P.D. algorithm, when the mean flow time is the performance criterion. The greater the value n'_{max} is the larger the difference between the guaranteed performance levels of the arbitrary and STF rules becomes.

Also, we can observe that the upper bounds of the P.D. algorithm under STF_{MAX} and STF_{MIN} rules may increase as the number of processors increases and $n'_{max} > \left\lfloor \frac{3m}{2} \right\rfloor - 1$, and hence their worst-case performance will deviate from the optimal one as m increases. However, we cannot be too rigorous, since their upper bounds depend on $\lambda_i^!$ and $\lambda_i^{!r}$ respectively. Finally, the extreme performance of the P.D. algorithm under the STF_{MIN} rule might be better, in some cases, than the corresponding one of the P.D. algorithm under STF_{MAX} rule, because $\lambda_i^{!r} \leqslant \lambda_i^!$ and $\beta_i^{!r} \leqslant \beta_i^!$.

Furthermore, we shall complete this section by analysing the P.D. algorithm under LMST ordering procedures.

Theorem 5.3.5: Let the priority list be in a $LMST_{MAX}$ ordering. Then, for non-identical processors:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{ \max\left\{ \max_{1\leqslant i\leqslant m-1}\left\{\lambda_i \max_{1\leqslant r\leqslant \left\lceil \frac{n_i}{i} \right\rceil}\left\{\frac{k}{r} + i - \frac{(i-1)}{2r}\right\}\right\}, \lambda_m\left(m - \frac{(m-1)}{2\left\lceil \frac{n_m}{m} \right\rceil}\right)\right\},\right.$$

$$\left.\max\left\{ \max_{1\leqslant i\leqslant m-1}\left\{\lambda_i\left(k + \frac{(n_i+1)}{2}\right)\right\}, \lambda_m\left(\frac{n_m+1}{2}\right)\right\}\right\},$$

where $k = \max_{1\leqslant i\leqslant m}\{n_i^! - n_i\}$ and $\lambda_i = \max_{J_j\in F_i}\left\{\frac{\sigma_j}{\tau_j}\right\}$, $1\leqslant i\leqslant m$, $1\leqslant j\leqslant n_i$.

Proof: Let $\bar{\omega}_1, \bar{\omega}_2, \ldots, \bar{\omega}_m$ be the contribution to the mean flow time of the jobs with $\ell_j = i$ for i=1(1)m respectively. So, we have

$$\bar{\omega}_{PD} = \bar{\omega}_1 + \bar{\omega}_2 + \ldots + \bar{\omega}_m \ . \qquad (5.3.20)$$

However, the contribution of jobs with $\ell_j=1$ is always

$$\bar{\omega}_1 < (k+1)\sigma_1^{F_1}+(k+2)\sigma_2^{F_1}+\ldots+(k+n_1)\sigma_{n_1}^{F_1} \quad .$$

Similarly, when $\ell_j=i$, $i=2(1)m-1$

$$\left.\begin{aligned}
\bar{\omega}_2 &< (k+1)\sigma_1^{F_2}+(k+2)\sigma_2^{F_2}+\ldots+(k+n_2)\sigma_{n_2}^{F_2} \\
&\vdots \\
\bar{\omega}_{m-1} &< (k+1)\sigma_1^{F_{m-1}}+(k+2)\sigma_2^{F_{m-1}}+\ldots+(k+n_{m-1})\sigma_{n_{m-1}}^{F_{m-1}} \\
\text{and} \quad \bar{\omega}_m &\leq \sigma_1^{F_m}+2\sigma_2^{F_m}+\ldots+n_m\sigma_{n_m}^{F_m} \quad .
\end{aligned}\right\} \qquad (5.3.21)$$

Therefore, because of the inequalities (5.3.21), equality (5.3.20) becomes

$$\begin{aligned}
\bar{\omega}_{PD} < {} & [(k+1)\sigma_1^{F_1}+\ldots+(k+n_1)\sigma_{n_1}^{F_1}]+[(k+1)\sigma_1^{F_2}+\ldots+(k+n_2)\sigma_{n_2}^{F_2}]+\ldots \\
& +[(k+1)\sigma_1^{F_{m-1}}+\ldots+(k+n_{m-1})\sigma_{n_{m-1}}^{F_{m-1}}]+[\sigma_1^{F_m}+\ldots+n_m\sigma_{n_m}^{F_m}] \quad . \qquad (5.3.22)
\end{aligned}$$

Since $\sigma_{n_i}^{F_i} \leq \sigma_{n_i-1}^{F_i} \leq \ldots \leq \sigma_2^{F_i} \leq \sigma_1^{F_i}$ for $i=1(1)m$, from Lemma 5.2.8 we get

$$\begin{aligned}
\bar{\omega}_{OPT} > {} & \frac{1}{\lambda_1}(\sigma_1^{F_1}+2\sigma_2^{F_1}+\ldots+n_1\sigma_{n_1}^{F_1})+\frac{1}{\lambda_2}\left[(\sigma_1^{F_2}+\sigma_2^{F_2})+2(\sigma_3^{F_2}+\sigma_4^{F_2})+\ldots+\left\lceil\frac{n_2}{2}\right\rceil\left(\sigma_{\left\lfloor\frac{n_2}{2}\right\rfloor 2+1}^{F_2}+\sigma_{\left\lceil\frac{n_2}{2}\right\rceil 2}^{F_2}\right)\right] \\
& +\ldots+\frac{1}{\lambda_m}\left[(\sigma_1^{F_m}+\ldots+\sigma_m^{F_m})+2(\sigma_{m+1}^{F_m}+\ldots+\sigma_{2m}^{F_m})+\ldots+\left\lceil\frac{n_m}{m}\right\rceil\left(\sigma_{\left\lfloor\frac{n_m}{m}\right\rfloor m+1}^{F_m}+\ldots+\sigma_{\left\lceil\frac{n_m}{m}\right\rceil m}^{F_m}\right)\right] \qquad (5.3.23)
\end{aligned}$$

where $\sigma_{n_i+1}^{F_i}=\ldots=\sigma_{\left\lceil\frac{n_i}{m}\right\rceil m}^{F_i}=0$ for $i=1(1)m$.

Therefore, from inequalities (5.3.22) and (5.3.23) we obtain

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \frac{\sum_{i=1}^{m-1}[(k+1)\sigma_1^{F_i}+(k+2)\sigma_2^{F_i}+\ldots+(k+n_i)\sigma_{n_i}^{F_i}]+(\sigma_1^{F_m}+2\sigma_2^{F_m}+\ldots+n_m\sigma_{n_m}^{F_m})}{\sum_{i=1}^{m}\frac{1}{\lambda_i}\left[\sum_{r=1}^{\lceil n_i/i\rceil} r(\sigma_{(r-1)i+1}^{F_i}+\ldots+\sigma_{ri}^{F_i})\right]}$$

or

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \frac{\sum_{i=1}^{m-1} \sum_{r=1}^{\lceil n_i/i \rceil} \sum_{j=1}^{i} (k+(r-1)i+j)\sigma^{F_i}_{(r-1)i+j} + \sum_{r=1}^{\lceil n_m/m \rceil} \sum_{j=1}^{m} ((r-1)m+j)\sigma^{F_m}_{(r-1)m+j}}{\sum_{i=1}^{m} \frac{1}{\lambda_i}\left[\sum_{r=1}^{\lceil n_i/i \rceil} r\left(\sum_{j=1}^{i} \sigma^{F_i}_{(r-1)i+j}\right)\right]} \tag{5.3.24}$$

Now, using Lemma 5.2.1 with k as being defined, $\ell=r$, $d=i$ and $a_{s+j}=\sigma^{F_i}_{(r-1)i+j}$, $1\leq j\leq i$, for $r=1(1)\left\lceil \frac{n_i}{i} \right\rceil$, and then for $i=1(1)m$ applying Lemma 5.2.2 and multiplying both sides of the resulting inequalities by λ_i, we get respectively,

$$\frac{\sum_{r=1}^{n_1} (k+r)\sigma^{F_1}_{r}}{\frac{1}{\lambda_1}\left[\sum_{r=1}^{n_1} \sigma^{F_1}_{r}\right]} < \lambda_1(k+1) ,$$

$$\frac{\sum_{r=1}^{\lceil n_2/2 \rceil} \sum_{j=1}^{2} [(k+(r-1)2+j)\sigma^{F_2}_{(r-1)2+j}]}{\frac{1}{\lambda_2}\left[\sum_{r=1}^{\lceil n_2/2 \rceil} r\left(\sum_{j=1}^{2} \sigma^{F_2}_{(r-1)2+j}\right)\right]} < \lambda_2\left[\max_{1\leq r\leq \left\lceil \frac{n_2}{2} \right\rceil} \left\{\frac{k}{r} + 2 - \frac{1}{2r}\right\}\right],$$

$$\vdots$$

$$\frac{\sum_{r=1}^{\lceil n_{m-1}/m-1 \rceil} \sum_{j=1}^{m-1} [(k+(r-1)(m-1)+j)\sigma^{F_{m-1}}_{(r-1)(m-1)+j}]}{\frac{1}{\lambda_{m-1}}\left[\sum_{r=1}^{\lceil n_{m-1}/m-1 \rceil} r\left(\sum_{j=1}^{m-1} \sigma^{F_{m-1}}_{(r-1)(m-1)+j}\right)\right]} < \lambda_{m-1}\left[\max_{1<r<\left\lceil \frac{n_{m-1}}{m-1} \right\rceil} \left\{\frac{k}{r}+m-1-\frac{(m-2)}{2r}\right\}\right]$$

and

$$\frac{\sum_{r=1}^{\lceil n_m/m \rceil} \sum_{j=1}^{m} [((r-1)m+j)\sigma^{F_m}_{(r-1)m+j}]}{\frac{1}{\lambda_m}\left[\sum_{r=1}^{\lceil n_m/m \rceil} r\left(\sum_{j=1}^{m} \sigma^{F_m}_{(r-1)m+j}\right)\right]} < \lambda_m\left(m - \frac{(m-1)}{2\left\lceil \frac{n_m}{m} \right\rceil}\right) . \tag{5.3.25}$$

Then, applying Lemma 5.2.2 for the inequalities (5.3.25) we get,

$$\frac{\sum_{i=1}^{m-1}\sum_{r=1}^{\lceil n_i/i\rceil}\sum_{j=1}^{i}\left[(k+(r-1)i+j)\sigma^{F_i}_{(r-1)i+j}\right] + \sum_{r=1}^{\lceil n_m/m\rceil}\sum_{j=1}^{m}\left[((r-1)m+j)\sigma^{F_m}_{(r-1)m+j}\right]}{\sum_{i=1}^{m}\frac{1}{\lambda_i}\left[\sum_{r=1}^{\lceil n_i/i\rceil} r\left(\sum_{j=1}^{i}\sigma^{F_i}_{(r-1)i+j}\right)\right]} <$$

$$\max\left\{\max_{1\le i\le m-1}\left\{\lambda_i \max_{1\le r\le\left\lceil\frac{n_i}{i}\right\rceil}\left\{\frac{k}{r}+i-\frac{(i-1)}{2r}\right\}\right\},\ \lambda_m\left(m-\frac{(m-1)}{2\left\lceil\frac{n_m}{m}\right\rceil}\right)\right\}\ .$$

Because of the last inequality, (5.3.24) becomes

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \max\left\{\max_{1\le i\le m-1}\left\{\lambda_i \max_{1<r<\left\lceil\frac{n_i}{i}\right\rceil}\left\{\frac{k}{r}+i-\frac{(i-1)}{2r}\right\}\right\},\ \lambda_m\left(m-\frac{(m-1)}{2\left\lceil\frac{n_m}{m}\right\rceil}\right)\right\}\ . \tag{5.3.26}$$

Nevertheless, the optimal mean flow time, except the bound (5.3.23), can also be bounded from below by:

$$\bar{\omega}_{OPT} > \sum_{i=1}^{m}\frac{1}{\lambda_i}(\sigma_1^{F_i}+\sigma_2^{F_i}+\ldots+\sigma_{n_i}^{F_i})\ .$$

This lower bound for $\bar{\omega}_{Opt}$ can be verified in exactly the same way as the bound (5.3.11'), which has been found in Theorem 5.3.3. Afterwards, following a similar analysis as the one which has been used to prove (5.3.26) we obtain

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \max\left\{\max_{1\le i\le m-1}\left\{\lambda_i\left(k+\frac{(n_i+1)}{2}\right)\right\},\ \lambda_m\left(\frac{n_m+1}{2}\right)\right\}. \tag{5.3.27}$$

Therefore, because of the inequalities (5.3.26) and (5.3.27) the worst-case performance bound of the P.D. algorithm when the priority list is formed by the $LMST_{MAX}$ rule is:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{\max\left\{\max_{1\le i\le m-1}\left\{\lambda_i \max_{1\le r\le\left\lceil\frac{n_i}{i}\right\rceil}\left\{\frac{k}{r}+i-\frac{(i-1)}{2r}\right\}\right\},\ \lambda_m\left(m-\frac{(m-1)}{2\left\lceil\frac{n_m}{m}\right\rceil}\right)\right\},\right.$$

$$\left.\max\left\{\max_{1\le i\le m-1}\left\{\lambda_i\left(k+\frac{(n_i+1)}{2}\right)\right\},\ \lambda_m\left(\frac{n_m+1}{2}\right)\right\}\right\}\ ,$$

which proves the theorem ¤

We can see that the second factor of the above bound can be less than the first one, whenever:

(1) $n_{max} \leq \left\lfloor \frac{3p}{2} \right\rfloor - 1$, where $p = \left\{ s : \lambda_s \left(k + s - \frac{s-1}{2\left\lceil \frac{n_s}{s} \right\rceil} \right) = \max_{1 \leq i \leq m-1} \left\{ \lambda_i \left(k + i - \frac{i-1}{2\left\lceil \frac{n_i}{i} \right\rceil} \right) \right\} \right\}$

and $2k-(i-1) \leq 0$ or $k \leq \left\lfloor \frac{i-1}{2} \right\rfloor$, $1 \leq i \leq m$; and

(2) $n_{max} \leq p-1$, where $p = \left\{ s : \lambda_s \left(k + \frac{s+1}{2} \right) = \max_{1 \leq i \leq m-1} \left\{ \lambda_i \left(k + \frac{i+1}{2} \right) \right\} \right\}$

and $2k-(i-1) > 0$ or $k \geq \left\lceil \frac{i-1}{2} \right\rceil$, $1 \leq i \leq m$.

Recall that $n_{max} = \max_{1 \leq i \leq m} \{n_i\}$.

Note that for the uniform processors case, $LMST_{MAX}$ and $LMST_{MIN}$ orderings are identical. So, the bound of the next theorem for uniform processors is also the worst-case performance bound for the $LMST_{MAX}$ ordering rule.

Theorem 5.3.6: Let the priority list be in a $LMST_{MIN}$ ordering. Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min \left\{ \max \left\{ \max_{1 \leq i \leq m-1} \left\{ \max_{1 \leq r \leq \left\lceil \frac{n_i}{i} \right\rceil} \left\{ \lambda_i^r \left(\frac{k}{r} + i - \frac{(i-1)}{2r} \right) \right\} \right\} , \right. \right.$$

$$\max_{1 \leq r \leq \left\lceil \frac{n_m}{m} \right\rceil} \left\{ \lambda_m^r \left(m - \frac{(m-1)}{2r} \right) \right\} ,$$

$$\left. \left. \max \left\{ \max_{1 \leq i \leq m-1} \left\{ \lambda_i \left(k + \frac{(n_i+1)}{2} \right) \right\}, \lambda_m \left(\frac{n_m+1}{2} \right) \right\} \right\} ;$$

(ii) for uniform processors:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min \left\{ \max \left\{ \max_{1 \leq i \leq m-1} \left\{ \beta_i \max_{1 \leq r \leq \left\lceil \frac{n_i}{i} \right\rceil} \left\{ \frac{k}{r} + i - \frac{(i-1)}{2r} \right\} \right\} , \beta_m \left(m - \frac{(m-1)}{2\left\lceil \frac{n_m}{m} \right\rceil} \right) \right\} \right.$$

$$\left. \max \left\{ \max_{1 \leq i \leq m-1} \left\{ \beta_i \left(k + \frac{(n_i+1)}{2} \right) \right\}, \beta_m \left(\frac{n_m+1}{2} \right) \right\} \right\},$$

where k,λ_i are as defined in Theorem 5.3.5, $\lambda_i^r = \max_{J_j \in F_i^r} \left\{\frac{\sigma_j}{\tau_j}\right\}$,

$\beta_i = \max_{1 \leq s \leq i} \{b_s\}/b_i$ for i=1(1)m.

Proof: Let $\bar{\omega}_1, \bar{\omega}_2, \ldots, \bar{\omega}_m$ be the contribution to the mean flow time of the jobs with $\ell_j=i$ for i=1(1)m respectively. So we have

$$\bar{\omega}_{PD} = \bar{\omega}_1 + \bar{\omega}_2 + \ldots + \bar{\omega}_m \tag{5.3.28}$$

However, the contribution of jobs with $\ell_j=i$ for i=1(1)m is always,

$$\left.\begin{aligned}
\bar{\omega}_1 &< (k+1)\tau_1^{F_1} + (k+2)\tau_2^{F_1} + \ldots + (k+n_1)\tau_{n_1}^{F_1}, \\
\bar{\omega}_2 &< \lambda_2^1[(k+1)\tau_1^{F_2} + (k+2)\tau_2^{F_2}] + \lambda_2^2[(k+3)\tau_3^{F_2} + (k+4)\tau_4^{F_2}] + \ldots \\
&\quad + \lambda_2^{\left\lceil \frac{n_2}{2} \right\rceil}\left[\left(k + \left\lfloor \frac{n_2}{2} \right\rfloor 2+1\right)\tau_{\left\lfloor \frac{n_2}{2} \right\rfloor 2+1}^{F_2} + \left(k + \left\lceil \frac{n_2}{2} \right\rceil 2\right)\tau_{\left\lceil \frac{n_2}{2} \right\rceil 2}^{F_2}\right], \\
&\vdots \\
\bar{\omega}_{m-1} &< \lambda_{m-1}^1[(k+1)\tau_1^{F_{m-1}} + \ldots + (k+m-1)\tau_{m-1}^{F_{m-1}}] + \ldots + \lambda_{m-1}^{\left\lceil \frac{n_{m-1}}{m-1} \right\rceil}\left[\left(k + \left\lfloor \frac{n_{m-1}}{m-1} \right\rfloor (m-1)+1\right)\tau_{\left\lfloor \frac{n_{m-1}}{m-1} \right\rfloor (m-1)+1}^{F_{m-1}} + \ldots + \left(k + \left\lceil \frac{n_{m-1}}{m-1} \right\rceil (m-1)\right)\tau_{\left\lceil \frac{n_{m-1}}{m-1} \right\rceil (m-1)}^{F_{m-1}}\right], \\
\bar{\omega}_m &< \lambda_m^1(\tau_1^{F_m} + \ldots + m\tau_m^{F_m}) + \ldots + \lambda_m^{\left\lceil \frac{n_m}{m} \right\rceil}\left[\left(\left\lfloor \frac{n_m}{m} \right\rfloor m+1\right)\tau_{\left\lfloor \frac{n_m}{m} \right\rfloor m+1}^{F_m} + \ldots + \left\lceil \frac{n_m}{m} \right\rceil m\tau_{\left\lceil \frac{n_m}{m} \right\rceil m}^{F_m}\right].
\end{aligned}\right\} \tag{5.3.29}$$

Also, from Lemma 5.2.7 we have

$$\begin{aligned}
\bar{\omega}_{OPT} \geq\ & (\tau_1^{F_1} + 2\tau_2^{F_1} + \ldots + n_1\tau_{n_1}^{F_1}) + \left[(\tau_1^{F_2} + \tau_2^{F_2}) + 2(\tau_3^{F_2} + \tau_4^{F_2}) + \ldots + \left\lceil \frac{n_2}{2} \right\rceil \left(\tau_{\left\lfloor \frac{n_2}{2} \right\rfloor 2+1}^{F_2} + \tau_{\left\lceil \frac{n_2}{2} \right\rceil 2}^{F_2}\right)\right] \\
&+ \ldots + \left[(\tau_1^{F_m} + \ldots + \tau_m^{F_m}) + 2(\tau_{m+1}^{F_m} + \ldots + \tau_{2m}^{F_m}) + \ldots + \left\lceil \frac{n_m}{m} \right\rceil \left(\tau_{\left\lfloor \frac{n_m}{m} \right\rfloor m+1}^{F_m} + \ldots + \tau_{\left\lceil \frac{n_m}{m} \right\rceil m}^{F_m}\right)\right].
\end{aligned} \tag{5.3.30}$$

However, $\bar{\omega}_i$ and $\bar{\omega}_0^i$ for i=1(1)m can also be bounded by

$$\left.\begin{aligned} \bar{\omega}_i &< \lambda_i\left((k+1)\tau_1^{F_i}+(k+2)\tau_2^{F_i}+\ldots+(k+n_i)\tau_{n_i}^{F_i}\right), \quad 1\leq i\leq m-1, \\ \bar{\omega}_m &< \lambda_m\left(\tau_1^{F_m}+2\tau_2^{F_m}+\ldots+n_m\tau_{n_m}^{F_m}\right) \end{aligned}\right\} \tag{5.3.31}$$

and

$$\bar{\omega}_0^i \geq \tau_1^{F_i}+\tau_2^{F_i}+\ldots+\tau_{n_i}^{F_i} \tag{5.3.32}$$

respectively.

On the other hand, for the case of uniform processors,

$$\left.\begin{aligned} \bar{\omega}_i &< \beta_i\left((k+1)\tau_1^{F_i}+(k+2)\tau_2^{F_i}+\ldots+(k+n_i)\tau_{n_i}^{F_i}\right), \ 1\leq i\leq m-1, \\ \bar{\omega}_m &< \beta_m\left(\tau_1^{F_m}+2\tau_2^{F_m}+\ldots+n_m\tau_{n_m}^{F_m}\right) \end{aligned}\right\} \tag{5.3.33}$$

and $\bar{\omega}_0^i$ is bounded by (5.3.30) or (5.3.32).

Furthermore, following a similar analysis with the one used for Theorem 5.3.5, either for the case of non-identical or uniform processors, we obtain respectively,

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{ \max\left\{ \max_{1\leq i\leq m-1} \left\{ \max_{1\leq r\leq \left\lceil \frac{n_i}{i} \right\rceil} \left\{ \lambda_i^r \left[\frac{k}{r} + i - \frac{(i-1)}{2r} \right] \right\} \right\}, \right.\right.$$

$$\left.\left. \max_{1\leq r\leq \left\lceil \frac{n_m}{m} \right\rceil} \left\{ \lambda_m^r \left[m - \frac{(m-1)}{2r} \right] \right\} \right\}, \quad \max\left\{ \max_{1\leq i\leq m-1} \left\{ \lambda_i \left[k + \frac{(n_i+1)}{2} \right] \right\}, \lambda_m \left[\frac{n_m+1}{2} \right] \right\} \right\}$$

and

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{ \max\left\{ \max_{1\leq i\leq m-1} \left\{ \beta_i \max_{1\leq r\leq \left\lceil \frac{n_i}{i} \right\rceil} \left\{ \frac{k}{r} + i - \frac{(i-1)}{2r} \right\} \right\}, \right.\right.$$

$$\left.\left. \beta_m \left[m - \frac{(m-1)}{2\left\lceil \frac{n_m}{m} \right\rceil} \right] \right\}, \quad \max\left\{ \max_{1\leq i\leq m-1} \left\{ \beta_i \left[k + \frac{(n_i+1)}{2} \right] \right\}, \beta_m \left[\frac{n_m+1}{2} \right] \right\} \right\},$$

which proves the theorem ¤

If $n_{max}\leq \left\lfloor \frac{3p}{2} \right\rfloor -1$ while $2k-(i-1)\leq 0$ or if $n_{max}\leq p-1$ while $2k-(i-1)>0$ then, as for the bound of Theorem 5.3.5, the second factor of the bound for uniform

processors in Theorem 5.3.6 is less, whilst the corresponding one for the bound for non-identical processors in the same theorem may be less than the first factor.

The worst-case performance bounds of the last two theorems for $\lambda_i=\lambda_i^r=1=\beta_i$, $1\leq i\leq m$, agree with the corresponding one found for the homogeneous multiprocessor system with independent memories. Also, Theorems 5.3.5 and 5.3.6 reveal that the utilisation of memory, in contributing to the extreme performance of the P.D. algorithm under the STF ordering rules when the mean flow time is chosen as the performance criterion, does not offer improvement, except for special cases only. Generally, if k is very small LMST might have better worst-case performance bounds than those given by STF ordering rules. More exactly, when k=0, i.e. $\sum_{J_j\in F_i} t_{ij} \geq \sum_{J_j\in F_{i+1}} t_{(i+1)j}$ for $i=1(1)m-1$, we can distinguish the following cases:

(1) If n_i, $1\leq i\leq m$, are not in decreasing order then the upper bounds of the Theorems 5.3.5 and 5.3.6 become:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{\max_{1\leq i\leq m}\left\{\lambda_i\left[i-\frac{(i-1)}{2\left\lceil\frac{n_i}{i}\right\rceil}\right]\right\},\ \max\left\{\lambda_i\left[\frac{n_i+1}{2}\right]\right\}\right\},$$

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{\max_{1\leq i\leq m}\left\{\max_{1\leq r\leq\left\lceil\frac{n_i}{i}\right\rceil}\left\{\lambda_i^r\left[i-\frac{(i-1)}{2r}\right]\right\}\right\},\ \max\left\{\lambda_i\left[\frac{n_i+1}{2}\right]\right\}\right\}$$

for non-identical processors respectively and,

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{\max_{1\leq i\leq m}\left\{\beta_i\left[i-\frac{(i-1)}{2\left\lceil\frac{n_i}{i}\right\rceil}\right]\right\},\ \max\left\{\beta_i\left[\frac{n_i+1}{2}\right]\right\}\right\}$$

for uniform processors.

We can see that even for k=0 the worst-case performance bounds of the P.D. algorithm under the LMST ordering rules are not clearly better than the corresponding ones when the STF rules are used to build up the priority list. Again, these bounds for $\lambda_i=\lambda_i^r=1=\beta_i$ agree with the bound found in this case

for the homogeneous multiprocessor system with independent memories.

(See Theorem I.3, Appendix I.)

(2) If n_i, $1 \leq i \leq m$ are in decreasing order i.e., $n_1 \geq n_2 \geq \ldots \geq n_m$ we have

(a) when the priority list is formed by the $LMST_{MIN}$ procedure

$$\left.\begin{aligned} \bar{\omega}_{PD} &< \sum_{i=1}^{m} \lambda_i \left(n_i \tau_1^{F_i} + (n_i - 1)\tau_2^{F_i} + \ldots + \tau_{n_i}^{F_i}\right) \\ \text{and} \\ \bar{\omega}_{OPT} &\geq \sum_{i=1}^{m} \left(n_i \tau_1^{F_i} + (n_i - 1)\tau_2^{F_i} + \ldots + \tau_{n_i}^{F_i}\right), \end{aligned}\right\} \qquad (5.3.32)$$

where $\tau_1^{F_i} \leq \tau_2^{F_i} \leq \ldots \leq \tau_{n_i}^{F_i}$ for i=1(1)m.

(b) when the priority list is formed by the $LMST_{MAX}$ procedure

$$\left.\begin{aligned} \bar{\omega}_{PD} &< \sum_{i=1}^{m} \left(n_i \sigma_1^{F_i} + (n_i - 1)\sigma_2^{F_i} + \ldots + \sigma_{n_i}^{F_i}\right) \\ \text{and} \\ \bar{\omega}_{OPT} &> \sum_{i=1}^{m} \frac{1}{\lambda_i} \left(n_i \sigma_1^{F_i} + (n_i - 1)\sigma_2^{F_i} + \ldots + \sigma_{n_i}^{F_i}\right), \end{aligned}\right\} \qquad (5.3.33)$$

where $\sigma_1^{F_i} \leq \sigma_2^{F_i} \leq \ldots \leq \sigma_{n_i}^{F_i}$ for i=1(1)m.

Furthermore, following the techniques used in previous theorems and with the help of Lemma 5.2.2 we can obtain $\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \lambda$ for both ordering procedures.

Similarly, for uniform processors $\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} \leq \max_{1 \leq i \leq m} \left\{ \max_{1 \leq s \leq i} \{b_s\}/b_i \right\}$. From the last bound it is apparent that if the speeds of the processors b_i, $1 \leq i \leq m$ are in an increasing order i.e., $b_1 \leq b_2 \leq \ldots \leq b_m$, then the LMST ordering rule offers an optimal schedule. For $\lambda = \beta_i = 1$ we get optimal schedules as found in the homogeneous multiprocessor system with independent memories.

However, the observations made for the upper bounds of the P.D. algorithm under STF_{MAX} and STF_{MIN} ordering procedures in comparison to the arbitrary one, can also apply for the $LMST_{MAX}$ and $LMST_{MIN}$ rules.

Finally, one could summarise this section by making the following remarks:

- the extreme performance of the P.D. algorithm for each of the considered ordering procedures, when the mean flow time performance criterion is used, is well presented;
- the bounds are widely varied from one heuristic ordering rule to another; and
- there is a lack of examples which can attain the values of the proven bounds.

5.4 TWO-PHASE PRIORITY DRIVEN (P.D.*) SCHEDULING ALGORITHMS

In the previous section we found worst-case performance bounds for a variety of P.D. algorithms. Here, we shall analyse the corresponding P.D.* algorithms. We know, that the two-phase algorithms without changing the completion time of the schedules, arrange the jobs, which have been allocated on the same processor, according to their time requirements in an increasing order. However, since it is obvious that such an arrangement tries to minimise the mean flow time of the task system, intuitively P.D.* algorithms are expected to perform better than the corresponding P.D. ones. At this stage one could raise the question: can the P.D.* algorithms offer an improvement over the worst-case performance of the corresponding P.D. algorithms? This is what we shall find out in this section through the establishment of a number of theorems.

Let $\bar{\omega}_{PD*}$ be the mean flow time of the schedule constructed by the P.D.* algorithm, when the priority list is formed by a heuristic ordering rule, and $\bar{\omega}_{OPT}$ be the mean flow time of an optimal schedule for a given task system $(J,[t_{ij}])$, $1 \leq i \leq m$ and $1 \leq j \leq n$.

Theorem 5.4.1: Let the priority list be in an arbitrary (RAND), STF_{MAX} or LTF_{MAX} ordering. Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{PD*}}{\bar{\omega}_{OPT}} < \min\left\{ \max_{1\leq i\leq m}\left\{\lambda_i^!\left(\nu_i - \frac{(\nu_i-1)}{2\left\lceil \frac{n_i^!}{\nu_i}\right\rceil}\right)\right\} , \max_{1\leq i\leq m}\left\{\lambda_i^!\left(\frac{n_i^!+1}{2}\right)\right\}\right\} ; \qquad (5.4.1)$$

and (ii) for uniform processors:

$$\frac{\bar{\omega}_{PD*}}{\bar{\omega}_{OPT}} < \min\left\{ \max_{1\leq i\leq m}\left\{\beta_i^!\left(\nu_i - \frac{(\nu_i-1)}{2\left\lceil \frac{n_i^!}{\nu_i}\right\rceil}\right)\right\} , \max_{1\leq i\leq m}\left\{\beta_i^!\left(\frac{n_i^!+1}{2}\right)\right\}\right\} , \qquad (5.4.2)$$

where $\lambda_i^!, \beta_i^!$ and ν_i are as defined in Theorems 5.3.1.

Proof: If the priority list is in an arbitrary, STF_{MAX} or LTF_{MAX} ordering and the P.D.* algorithm is used to do the scheduling then $\bar{\omega}$ is bounded by

$$\bar{\omega}_{PD^*} > \sum_{i=1}^{m} (n_i^! \sigma_{n_i^!}^{G_i} + (n_i^! - 1)\sigma_{n_i^!-1}^{G_i} + \ldots + \sigma_1^{G_i}) \quad . \tag{5.4.3}$$

Further, from Lemmas 5.2.6 and 5.2.4 we can obtain

$$\bar{\omega}_{OPT} > \sum_{i=1}^{m} \frac{1}{\lambda_i^!} \left[(\sigma_1^{G_i} + \sigma_2^{G_i} + \ldots + \sigma_{\nu_i}^{G_i}) + 2(\sigma_{\nu_i+1}^{G_i} + \ldots + \sigma_{2\nu_i}^{G_i}) + \ldots + \left\lceil \frac{n_i^!}{\nu_i} \right\rceil \left(\sigma_{\left\lfloor \frac{n_i^!}{\nu_i} \right\rfloor \nu_i + 1}^{G_i} + \ldots + \sigma_{\left\lceil \frac{n_i^!}{\nu_i} \right\rceil \nu_i}^{G_i} \right) \right] \tag{5.4.4}$$

and

$$\bar{\omega}_{OPT} > \sum_{i=1}^{m} \frac{1}{\lambda_i^!} (\sigma_1^{G_i} + \sigma_2^{G_i} + \ldots + \sigma_{n_i^!}^{G_i}) \tag{5.4.5}$$

respectively, where $\sigma_1^{G_i} \geq \sigma_2^{G_i} \geq \ldots \geq \sigma_{n_i^!}^{G_i}$ and $\sigma_{n_i^!+1}^{G_i} = \ldots = \sigma_{\left\lceil \frac{n_i^!}{\nu_i} \right\rceil \nu_i}^{G_i} = 0$ for i=1(1)m.

Now, for the pairs of inequalities (5.4.3)-(5.4.4) and (5.4.3)-(5.4.5) if we apply exactly the same analysis as the one used in Theorem 5.3.3, we will obtain the worst-case bound (5.4.1).

On the other hand for the uniform processors case, replacing the inequality (5.4.3) by

$$\bar{\omega}_{PD^*} \leq \sum_{i=1}^{m} \left[\left(\min_{1 \leq k \leq \nu_i} \{b_k\}/b_i \right) (n_i^! \sigma_{n_i^!}^{G_i} + (n_i^! - 1)\sigma_{n_i^!-1}^{G_i} + \ldots + \sigma_1^{G_i}) \right]$$

and $\lambda_i^!$ by $\max_{1 \leq k \leq \nu_i} \{b_k\} / \min_{1 \leq k \leq \nu_i} \{b_k\}$ in the inequalities (5.4.4) and (5.4.5) and working similarly as in the case for non-identical processors, we eventually obtain the worst-case bound (5.4.2), which completes the theorem ¤

Theorem 5.4.2: Let the priority list be in a STF_{MIN} or LTF_{MIN} ordering. Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{PD^*}}{\bar{\omega}_{OPT}} < \min \left\{ \max_{1 \leq i \leq m} \left\{ \max_{1 \leq r \leq \left\lceil \frac{n_i^!}{\nu_i} \right\rceil} \left\{ \lambda_i^{!r} \left(\nu_i - \frac{(\nu_i - 1)}{2r} \right) \right\} \right\} , \max_{1 \leq i \leq m} \left\{ \lambda_i^! \left(\frac{n_i^! + 1}{2} \right) \right\} \right\}; \tag{5.4.6}$$

and (ii) for uniform processors:

$$\frac{\bar{\omega}_{PD^*}}{\bar{\omega}_{OPT}} < \min \left\{ \max_{1 \leq i \leq m} \left\{ \max_{1 \leq r \leq \left\lceil \frac{n_i^!}{\nu_i} \right\rceil} \left\{ \beta_i^{!r} \left(\nu_i - \frac{(\nu_i - 1)}{2r} \right) \right\} \right\} , \max_{1 \leq i \leq m} \left\{ \beta_i^! \left(\frac{n_i^! + 1}{2} \right) \right\} \right\}, \tag{5.4.7}$$

where $\lambda_i', \beta_i', \nu_i$, and $\lambda_i'^r, \beta_i'^r$ are as defined in Theorems 5.3.1 and 5.3.4 respectively.

Proof: For the case of non-identical processors $\bar{\omega}$ is bounded by

$$\bar{\omega}_{PD} < \sum_{i=1}^{m}\Bigg[\lambda_i'^1(\tau_1^{G_i}+2\tau_2^{G_i}+\ldots+\nu_i\tau_{\nu_i}^{G_i})+\lambda_i'^2\Big((\nu_i+1)\tau_{\nu_i+1}^{G_i}+\ldots+2\nu_i\tau_{2\nu_i}^{G_i}\Big)+$$

$$\ldots+\lambda_i'^{\left\lceil\frac{n_i'}{\nu_i}\right\rceil}\Bigg(\left(\left\lfloor\frac{n_i'}{\nu_i}\right\rfloor\nu_i+1\right)\tau_{\left\lfloor\frac{n_i'}{\nu_i}\right\rfloor\nu_i+1}^{G_i}+\ldots+\left\lceil\frac{n_i'}{\nu_i}\right\rceil\nu_i\,\tau_{\left\lceil\frac{n_i'}{\nu_i}\right\rceil\nu_i}^{G_i}\Bigg)\Bigg] \qquad (5.4.8)$$

or

$$\bar{\omega}_{OPT} < \sum_{i=1}^{m}\lambda_i'(\tau_1^{G_i}+2\tau_2^{G_i}+\ldots+n_i'\tau_{n_i'}^{G_i}), \qquad (5.4.9)$$

where $\tau_1^{G_i}\geq\tau_2^{G_i}\geq\ldots\geq\tau_{n_i'}^{G_i}$, and $\tau_{n_i'+1}^{G_i}=\ldots=\tau_{\left\lceil\frac{n_i'}{\nu_i}\right\rceil\nu_i}^{G_i}=0$ for i=1(1)m.

From Lemmas 5.2.5 and 5.2.4 we can obtain respectively,

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m}\Bigg[(\tau_1^{G_i}+\tau_2^{G_i}+\ldots+\tau_{\nu_i}^{G_i})+2(\tau_{\nu_i+1}^{G_i}+\ldots+\tau_{2\nu_i}^{G_i})+\ldots+\left\lceil\frac{n_i'}{\nu_i}\right\rceil\Big(\tau_{\left\lfloor\frac{n_i'}{\nu_i}\right\rfloor\nu_i+1}^{G_i}+\ldots+\tau_{\left\lceil\frac{n_i'}{\nu_i}\right\rceil\nu_i}^{G_i}\Big)\Bigg] \qquad (5.4.10)$$

and

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m}(\tau_1^{G_i}+\tau_2^{G_i}+\ldots+\tau_{n_i'}^{G_i}), \qquad (5.4.11)$$

Now, for the case of uniform processors $\bar{\omega}$ is bounded as in (5.4.8) and (5.4.9), but with $\lambda_i'^r$, $1\leq r\leq\left\lceil\frac{n_i'}{\nu_i}\right\rceil$ and λ_i' being substituted by $\beta_i'^r$ and β_i' respectively, whilst $\bar{\omega}_{OPT}$ is bounded as in (5.4.10) and (5.4.11).

Furthermore, following a similar analysis, as in Theorem 5.3.3, for the pairs of inequalities (5.4.8)-(5.4.10), (5.4.9)-(5.4.11) for non-identical and uniform processors we finally achieve the worst-case bounds (5.4.6) and (5.4.7) respectively. □

Theorem 5.4.3: Let the priority list be in a LMF, $LMST_{MAX}$ or $LMLT_{MAX}$ ordering. Then, for non-identical processors:

$$\frac{\bar{\omega}_{PD^*}}{\bar{\omega}_{OPT}} < \min\left\{\max\left\{\max_{1\leq i\leq m-1}\left\{\lambda_i \max_{1\leq r\leq\left\lceil\frac{n_i!}{i}\right\rceil}\left\{\frac{k}{r}+i-\frac{(i-1)}{2r}\right\}\right\},\ \lambda_m\left\{m-\frac{(m-1)}{2\left\lceil\frac{n_m}{m}\right\rceil}\right\}\right\},\right.$$

$$\left.\max\left\{\max_{1\leq i\leq m-1}\left\{\lambda_i\left\{k+\frac{(n_i+1)}{2}\right\}\right\},\ \lambda_m\left\{\frac{n_m+1}{2}\right\}\right\}\right.,$$

$$\left.\max_{1\leq i\leq m}\left\{\lambda_i^!\left\{\nu_i-\frac{(\nu_i-1)}{2\left\lceil\frac{n_i^!}{\nu_i}\right\rceil}\right\}\right\},\ \max_{1\leq i\leq m}\left\{\lambda_i^!\left\{\frac{n_i^!+1}{2}\right\}\right\}\right\} \qquad (5.4.12)$$

where $\lambda_i^!, \nu_i$ and λ_i, k are as defined in Theorems 5.3.1 and 5.3.5 respectively.

Proof: For the priority list being in such an ordering, i.e., LMF, $LMST_{MAX}$ or $LMLT_{MAX}$, $\bar{\omega}$ is bounded by

$$\bar{\omega}_{PD^*} < \sum_{i=1}^{m-1}\left[(k+1)\sigma_1^{F_i}+(k+2)\sigma_2^{F_i}+\ldots+(k+n_i)\sigma_{n_i}^{F_i}\right]+(\sigma_1^{F_m}+2\sigma_2^{F_m}+\ldots+n_m\sigma_{n_m}^{F_m}), \qquad (5.4.13)$$

where $\sigma_1^{F_i}\geq\sigma_2^{F_i}\geq\ldots\geq\sigma_{n_i}^{F_i}$ for i=1(1)m.

Now, from Lemma 5.2.8 we have

$$\bar{\omega}_{OPT} > \sum_{i=1}^{m}\frac{1}{\lambda_i}\left[\sum_{r=1}^{\lceil n_i/i\rceil} r(\sigma_{(r-1)i+1}^{F_i}+\ldots+\sigma_{ri}^{F_i})\right], \qquad (5.4.14)$$

where $\sigma_{n_i+1}^{F_i}=\ldots=\sigma_{\left\lceil\frac{n_i}{i}\right\rceil i}^{F_i}=0$ for i=1(1)m, whereas from Lemma 5.2.4 we can obtain

$$\bar{\omega}_{OPT} > \sum_{i=1}^{m}\frac{1}{\lambda_i}(\sigma_1^{F_i}+\sigma_2^{F_i}+\ldots+\sigma_{n_i}^{F_i}). \qquad (5.4.15)$$

Furthermore, following the same analysis, as in theorem 5.3.5, for the pairs of inequalities (5.4.13)-(5.4.14) and (5.4.13)-(5.4.15) we obtain respectively

$$\frac{\bar{\omega}_{PD^*}}{\bar{\omega}_{OPT}} < \max\left\{\max_{1\leq i\leq m-1}\left\{\lambda_i \max_{1\leq r\leq\left\lceil\frac{n_i}{i}\right\rceil}\left\{\frac{k}{r}+i-\frac{(i-1)}{2r}\right\}\right\},\ \lambda_m\left\{m-\frac{(m-1)}{2\left\lceil\frac{n_m}{m}\right\rceil}\right\}\right\}$$

and

$$\frac{\bar{\omega}_{PD^*}}{\bar{\omega}_{OPT}} < \max\left\{\max_{1\leq i\leq m-1}\left\{\lambda_i\left\{k+\frac{(n_i+1)}{2}\right\}\right\},\ \lambda_m\left\{\frac{n_m+1}{2}\right\}\right\}.$$

In addition, the inequalities (5.4.3),(5.4.4) and (5.4.5) of the Theorem 5.4.1 are held even if the priority list is in the LMF, $LMST_{MAX}$ or $LMLT_{MAX}$ ordering. This is true, since for each job J_j, $1 \leq j \leq n$, which is scheduled on the i^{th} processor with (k-1) jobs following it on that processor, $t_{ij} \leq \sigma_k^{F_i}$, where $\sigma_1^{F_i} \geq \sigma_2^{F_i} \geq \ldots \geq \sigma_{n_i}^{F_i}$ for i=1(1)m. So, the upper bound for non-identical processors on Theorem 5.4.1 is eventually obtained.

Finally, combining this bound with the ones found previously in this proof we get the worst-case bound (5.4.12), which proves the theorem □

Theorem 5.4.4: Let the priority list be in a $LMST_{MIN}$ or $LMLT_{MIN}$ ordering. Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{PD^*}}{\bar{\omega}_{OPT}} < \min\left\{\max\left\{\max_{1\leq i\leq m-1}\left\{\max_{1\leq r\leq\left\lceil\frac{n_i}{i}\right\rceil}\left\{\lambda_i^r\left(\frac{k}{r}+i-\frac{(i-1)}{2r}\right)\right\}\right\}, \max_{1\leq r\leq\left\lceil\frac{n_m}{m}\right\rceil}\left\{\lambda_m^r\left(m-\frac{(m-1)}{2r}\right)\right\}\right\}\right.$$

$$\max\left\{\max_{1\leq i\leq m-1}\left\{\lambda_i\left(k+\frac{(n_i+1)}{2}\right)\right\}, \lambda_m\left(\frac{n_m+1}{2}\right)\right\},$$

$$\left.\max_{1\leq i\leq m}\left\{\lambda_i'\left(\nu_i-\frac{(\nu_i-1)}{2\left\lceil\frac{n_i'}{\nu_i}\right\rceil}\right)\right\}, \max_{1\leq i\leq m}\left\{\lambda_i'\left(\frac{n_i'+1}{2}\right)\right\}\right\}; \qquad (5.4.16)$$

(ii) for uniform processors:

$$\frac{\bar{\omega}_{PD^*}}{\bar{\omega}_{OPT}} < \min\left\{\max\left\{\max_{1\leq i\leq m-1}\left\{\beta_i\max_{1\leq r\leq\left\lceil\frac{n_i}{i}\right\rceil}\left\{\frac{k}{r}+i-\frac{(i-1)}{2r}\right\}\right\}, \beta_m\left(m-\frac{(m-1)}{2\left\lceil\frac{n_m}{m}\right\rceil}\right)\right\}\right.,$$

$$\max\left\{\max_{1\leq i\leq m-1}\left\{\beta_i\left(k+\frac{(n_i+1)}{2}\right)\right\}, \beta_m\left(\frac{n_m+1}{2}\right)\right\},$$

$$\left.\max_{1\leq i\leq m}\left\{\beta_i'\left(\nu_i-\frac{(\nu_i-1)}{2\left\lceil\frac{n_i'}{\nu_i}\right\rceil}\right)\right\}, \max_{1\leq i\leq m}\left\{\beta_i'\left(\frac{n_i'+1}{2}\right)\right\}\right\}, \qquad (5.4.17)$$

where λ_i', β_i' and ν_i, k and λ_i, and λ_i^r, β_i are as defined in the Theorems 5.3.1, 5.3.5 and 5.3.6 respectively.

Proof: The mean flow time of the schedules constructed by the P.D.* algorithm when the priority list is in a $LMST_{MIN}$ or $LMLT_{MIN}$ ordering, for the case of non-identical processors, is bounded by

$$\bar{\omega}_{PD*} < \sum_{i=1}^{m} \sum_{r=1}^{\lceil n_i/i \rceil} \lambda_i^r \left[(k+(r-1)i+1)\tau_{(r-1)i+1}^{F_i} + \ldots + (k+ri)\tau_{ri}^{F_i} \right] , \qquad (5.4.18)$$

where $\tau_1^{F_i} \geq \tau_2^{F_i} \geq \ldots \geq \tau_{n_i}^{F_i}$ and $\tau_{n_i+1}^{F_i} = \ldots = \tau_{\lceil \frac{n_i}{i} \rceil i}^{F_i} = 0$ for i=1(1)m, or

$$\bar{\omega}_{PD*} < \sum_{i=1}^{m-1} \lambda_i \left[(k+1)\tau_1^{F_i} + (k+2)\tau_2^{F_i} + \ldots + (k+n_i)\tau_{n_i}^{F_i} \right] + \lambda_m (\tau_1^{F_m} + 2\tau_2^{F_m} + \ldots + \tau_{n_m}^{F_m}) . \qquad (5.4.19)$$

On the other hand, for uniform processors we always have

$$\bar{\omega}_{PD*} < \sum_{i=1}^{m-1} \beta_i \left[(k+1)\tau_1^{F_i} + (k+2)\tau_2^{F_i} + \ldots + (k+n_i)\tau_{n_i}^{F_i} \right] + \beta_m (\tau_1^{F_m} + 2\tau_2^{F_m} + \ldots + n_m \tau_{n_m}^{F_m}) . \qquad (5.4.20)$$

However, from Lemmas 5.2.7 and 5.2.4 we have respectively,

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m} \sum_{r=1}^{\lceil n_i/i \rceil} r(\tau_{(r-1)i+1}^{F_i} + \ldots + \tau_{ri}^{F_i}) \qquad (5.4.21)$$

and

$$\bar{\omega}_{OPT} \sum_{i=1}^{m} (\tau_1^{F_i} + \tau_2^{F_i} + \ldots + \tau_{n_i}^{F_i}) . \qquad (5.4.22)$$

Now, for the pairs of the inequalities (5.4.18)-(5.4.21), (5.4.19)-(5.4.22) and (5.4.20)-(5.4.21), (5.4.20)-(5.4.22) following a similar analysis as in Theorem 5.3.5, we obtain the first two factors of the bounds (5.4.16) and (5.4.17) respectively. Furthermore, the inequalities (5.4.3), (5.4.4) and (5.4.5) of Theorem 5.4.1 are also held when the priority list is in a $LMST_{MIN}$ or $LMLT_{MIN}$ ordering. (Recall that $t_{ij} \leq \sigma_k^{F_i}$.) Therefore, combining the obtained two factors with the bounds of Theorem 5.4.1 we finally get the bounds (5.4.16) and (5.4.17) which prove the theorem. □

The above found worst-case bound for uniform processors also applies for the cases where the priority list is in the LMF, $LMST_{MAX}$ or $LMLT_{MAX}$ ordering.

The bounds of all the Theorems 5.4.1, 5.4.2, 5.4.3 and 5.4.4 for $\lambda_i=\lambda_i^!=\lambda_i^{!r}=1=\beta_i=\beta_i^!=\beta_i^r$ agree with the bounds which have been found for the homogeneous multiprocessor system with independent memories. (See Theorems I.4 and I.5, Appendix I)

The theorems already mentioned in the present section indicate that the P.D.* algorithms can offer improvement to the worst-case performance bounds over the corresponding P.D. algorithms only when the priority list was formed by the RAND, LMF, LTF_{MIN}, LTF_{MAX}, $LMLT_{MIN}$ or the $LMLT_{MAX}$ ordering rule. For the remaining ordering procedures either very little or no improvement has been achieved. However, the remarks made for the bounds in the section 5.3, which appear here, are also valid.

Furthermore, in this section we observe again the informative style of the proven worst-case bounds as well as the lack of examples to attain their values. Nevertheless, in contrast to the conclusion made in the previous section, the worst-case performance bounds of the P.D.* algorithms are very close to each other when simple heuristic procedures are used to construct the priority list.

5.5 QUICK AND DIRTY (Q.A.D.) SCHEDULING ALGORITHMS

We recall that the philosophy of the Q.A.D. algorithms is to allocate each job of the task system on such a processor, so that its contribution to the mean flow time is the least possible. Such a behaviour makes us believe that they should perform better than the corresponding P.D. algorithms. However, whether or not the Q.A.D. algorithms offer an improvement over the worst-case performance of the corresponding P.D. algorithms we shall find out in the remainder of this section. Further, since the Q.A.D. algorithms are right justified while the P.D. ones are left justified, the STF and LMST ordering rules correspond to the LTF and LMLT respectively, and vice versa.

For this section, let $\bar{\omega}_{QAD}$ be the mean flow time of the schedule constructed by the Q.A.D. algorithm, when the priority list is formed by a heuristic ordering procedure, and $\bar{\omega}_{OPT}$ be the mean flow time of an optimal schedule for a given task system $(J,[t_{ij}])$, $1 \leq i \leq m$, $1 \leq j \leq n$.

As in the previous sections (5.3 and 5.4), we start the analysis by examining an arbitrary ordering priority list.

Theorem 5.5.1: Let the priority list be in an arbitrary ordering. Then, for non-identical or uniform processors,

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < n'_{max} \quad ,$$

where $n'_{max} = \max_{1 \leq i \leq m} \{n'_i\}$.

Proof: Let S and S_o be the schedules which correspond to $\bar{\omega}_{QAD}$ and $\bar{\omega}_{OPT}$ respectively. Also, let c_j and c_j^o be the contribution to $\bar{\omega}_{QAD}$ and $\bar{\omega}_{OPT}$ respectively of the job J_j, $1 \leq j \leq n$. So,

$$\left.\begin{aligned} c_j &= h_i t_{ij} \\ \text{and} \qquad c_j^o &= h_k^o t_{kj} \ , \end{aligned}\right\} \ , \tag{5.5.1}$$

where h_i and h_k^o are integers and actually, one greater than the number of jobs following J_j on the i^{th} processor of the S schedule and the k^{th} processor of the S_o schedule respectively.

However,

$$\bar{\omega}_{QAD} = \sum_{j=1}^{n} c_j \quad ,$$

$$\bar{\omega}_{OPT} = \sum_{j=1}^{n} c_j^o$$

and therefore,

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} = \frac{\sum_{j=1}^{n} c_j}{\sum_{j=1}^{n} c_j^o} \quad .$$

Now, it is obvious that if we show

$$\frac{c_j}{c_j^o} \leq n'_{max}, \quad 1 \leq j \leq n \quad , \tag{5.5.2}$$

then the proof will be almost completed.

Suppose that the inequality (5.5.2) is false for some j; then,

$$\frac{c_j}{c_j^o} > n'_{max}$$

and because of the equalities (5.5.1)

$$h_i t_{ij} > n'_{max} h_k^o t_{kj} \quad . \tag{5.5.3}$$

However,

$$h_i t_{ij} = \min_{1 \leq g \leq \ell_j} \{h_g t_{gj}\} \quad ,$$

where $1 \leq \ell_j \leq m$, and therefore

$$h_i t_{ij} \leq h_k t_{kj} \quad , \tag{5.5.4}$$

where h_k can be defined analogous to h_i in the equations (5.5.1).

Combining the inequalities (5.5.3) and (5.5.4) we obtain

$$h_k t_{kj} \geq h_i t_{ij} > n'_{max} h_k^o t_{kj}$$

or

$$h_k > n'_{max} h_k^o \quad .$$

This is not true since h_k and h_k^o are integers greater or equal to 1 and $h_k \leq n'_{max}$. So, this contradiction proves the inequality (5.5.2).

Consequently,

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < n'_{max} \quad . \tag{5.5.5}$$

The equality has been rejected from (5.5.5) since it cannot be $c_j=n'_{max}c^o_j$ for all $1\leq j\leq n$.

In addition, the bound is a best possible one. This can be realised by considering the Example 5.1 which is incorporated into the proof of the following theorem.

(Note: The above proven upper bound applies also to the case of the heterogeneous multiprocessor system without private memories. In addition, comparing this bound with the one presented by Clark [Cl] for such a system, (see Table 3.9), we conclude that the present one is more informative.)

Theorem 5.5.2: Let the priority list be in a LMF, STF_{MIN}, STF_{MAX}, $LMST_{MIN}$ or $LMST_{MAX}$ ordering. Then, for non-identical or uniform processors,

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < n'_{max} \quad .$$

Proof: Since the upper bound of Theorem 5.5.1 was proved for an arbitrary priority list, that bound is also an upper bound for any particular priority list when the Q.A.D. algorithm is used to construct the schedule. Moreover, there are examples which can cause the Q.A.D. algorithm under the LMF, STF_{MAX}, STF_{MIN}, $LMST_{MAX}$ or $LMST_{MIN}$ ordering rules to deviate from optimal performance by the amount allowed by the proven worst-case bound. In fact the following example is one of them.

Example 5.1: Let the task system $(J,[t_{ij}])$ be defined by the set of independent jobs $J=\{J_1,J_2,\ldots,J_n\}$ and the $(m\times n)$ matrix

$$[t_{ij}]= \begin{array}{c} \overbrace{}^{n_1}\ \overbrace{}^{n_2} \qquad\qquad \overbrace{}^{n_{m-1}}\ \overbrace{}^{n_m} \\ \begin{bmatrix} \varepsilon\ldots\varepsilon & \varepsilon\ldots\varepsilon & \ldots & \varepsilon\ldots\varepsilon & \varepsilon\ldots\varepsilon & X \\ \infty\ldots\infty & X\ldots X & \ldots & X\ldots X & X\ldots X & X+Z \\ \vdots & \infty\ldots\infty & \ldots & X\ldots X & X\ldots X & X+Z \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ \vdots & \vdots & & \vdots & \vdots & \vdots \\ \vdots & \vdots & & X\ldots X & X\ldots X & X+Z \\ \infty\ldots\infty & \infty\ldots\infty & \ldots & \infty\ldots\infty & X\ldots X & X+Z \end{bmatrix} \end{array} ,$$

$$(J_1, \ldots J_{n_1+n_2}, \quad . \quad . \quad . \quad , J_n)$$

where $n=n_1+n_2+\ldots+n_m$, $\quad \varepsilon, X, Z \in R^+$ and $\quad \varepsilon << X << Z$.

Clearly, the priority list $L=(J_1,J_2,\ldots,J_n)$ is a LMF, STF_{MIN}, STF_{MAX}, $LMST_{MIN}$ or $LMST_{MAX}$ ordering and also an arbitrary ordering. The schedule resulting from the priority list L is given in Fig. 5.1, whereas the corresponding optimal one is shown in Fig. 5.2.

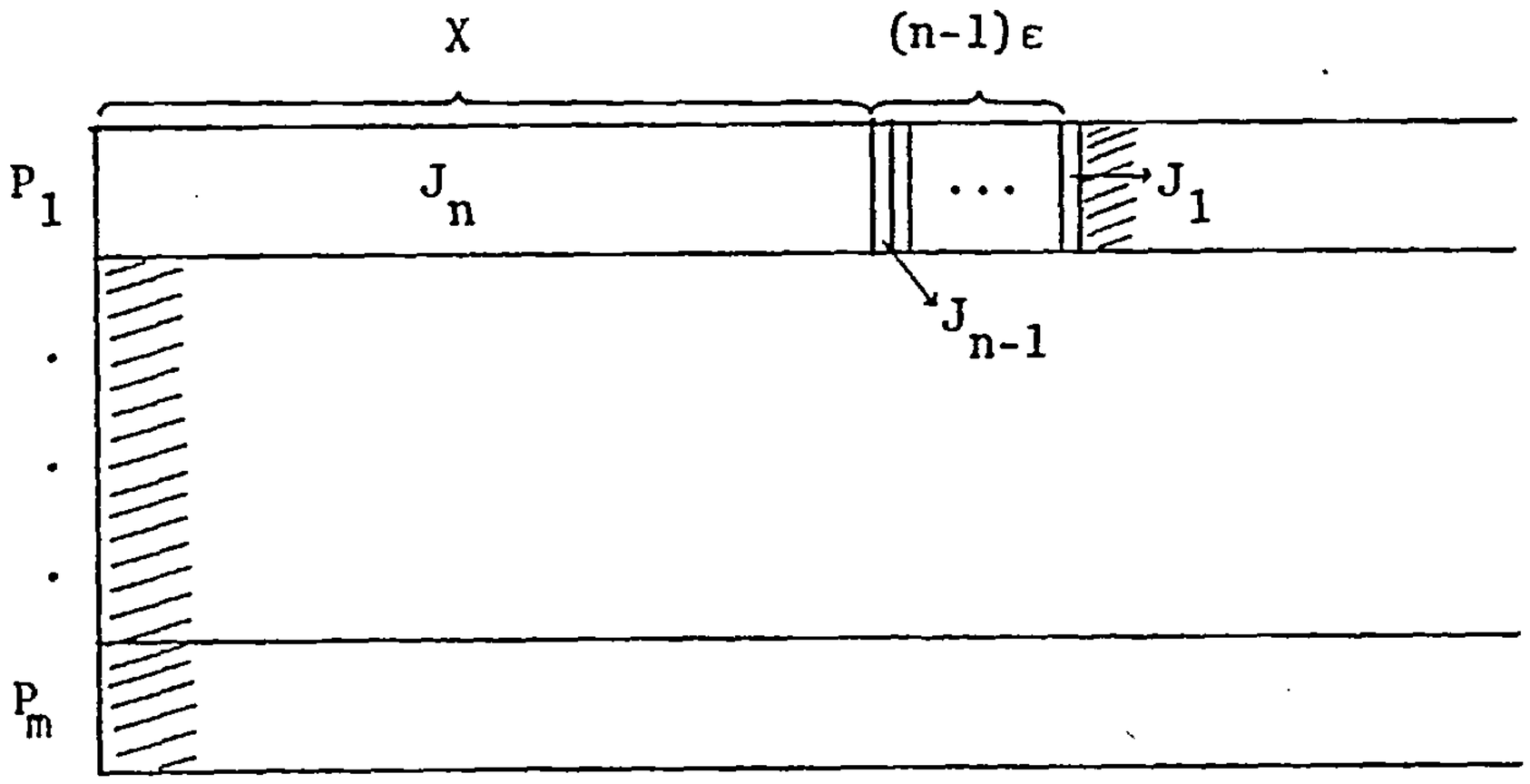


FIGURE 5.1: Worst-case schedule to illustrate Theorem 5.5.2

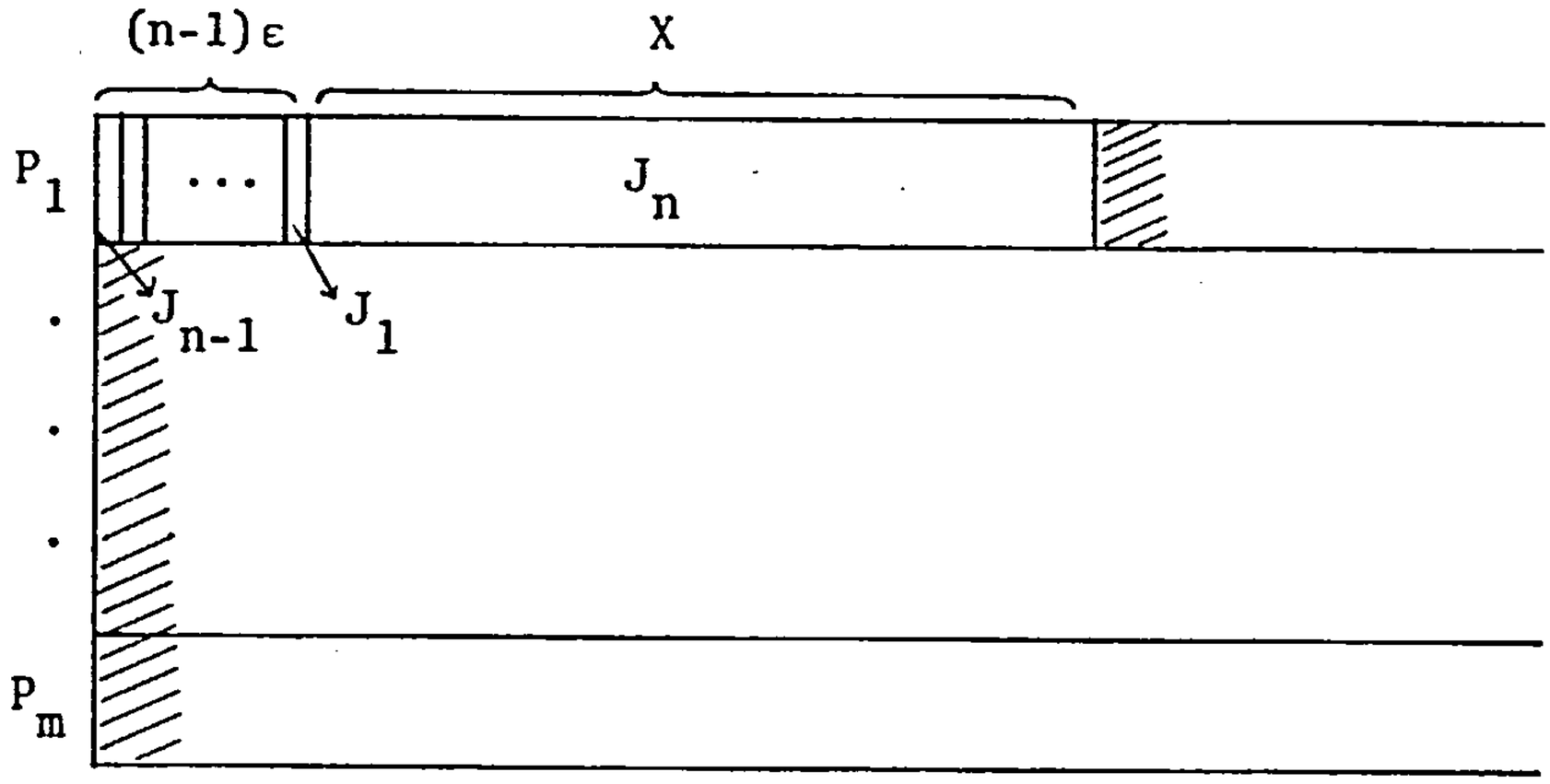


FIGURE 5.2: Optimal schedule to illustrate Theorem 5.5.2

The ratio of the mean flow times of these two schedules is:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} = \frac{nX+f_1(\varepsilon)}{X+f_2(\varepsilon)}$$

or

$$\lim_{\varepsilon \to 0} \frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} = \frac{nX}{X} = n = n'_{max}$$

which is the value predicted by Theorem 5.5.2.

The above mentioned task system, when J_n is replaced by another J_n with time requirement $(X, X^2/\varepsilon, \ldots, X^2/\varepsilon)$, can be used as an example to show that n'_{max} is also a best possible bound for the case of uniform processors ¤

The bound of Theorems 5.5.1 and 5.5.2 agrees with the one found for the homogeneous multiprocessor system with independent memories. (See Theorem II.1, Appendix II.) Therefore, the use of a heterogeneous multiprocessor system instead of a homogeneous one does not worsen the guaranteed performance levels of the Q.A.D. algorithm when the priority list is in a RAND, LMF, STF_{MIN}, STF_{MAX}, $LMST_{MIN}$ or $LMST_{MAX}$ ordering.

Nevertheless, Theorem 5.5.2 indicates that the use of the heuristic procedures LMF, STF_{MIN}, STF_{MAX}, $LMST_{MIN}$ or $LMST_{MAX}$ to pre-order the priority list does not offer any improvement in a worst-case sense over an arbitrary ordering Q.A.D. algorithm.

In addition, comparing Q.A.D. with the corresponding P.D. algorithms when the priority list is constructed by one of the above mentioned heuristic procedures, we can realise that the Q.A.D. algorithms have a better worst-case performance. However, if λ_i', $1 \leq i \leq m$, is very close to 1 and the n'_{max} of the schedule constructed by a P.D. algorithm is less than the n'_{max} of the schedule corresponding to a Q.A.D. one, then P.D. algorithms appear to have a better worst-case performance. Finally, the remarks made for the upper bound of Theorem 5.3.1, i.e., how the bound behaves as the number of processors or tasks in the task system increases or decreases, are valid for the bound given in Theorem 5.5.1 as well; in fact, they are more meaningful in this theorem.

Now, we continue the analysis of the Q.A.D. algorithm for the cases where LTF ordering rules are used to form the priority list.

Theorem 5.5.3: Let the priority list be in a LTF_{MAX} ordering. Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \min \left\{ \max_{1 \leq r \leq \left\lceil \frac{n}{m} \right\rceil} \left\{ \frac{\lambda}{\mu_r} \left[m - \frac{(m-1)}{2r} \right] \right\}, \max_{1 \leq i \leq m} \left\{ \lambda_i' \left[\nu_i - \frac{(\nu_i - 1)}{2 \left\lceil \frac{n_i'}{\nu_i} \right\rceil} \right] \right\}, \right.$$

$$\left. \max_{1 \leq i \leq m} \left\{ \lambda_i' \left[\frac{n_i'+1}{2} \right] \right\}, \; n'_{max} \right\} \; ; \qquad (5.5.6)$$

(ii) for uniform processors:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \min \left\{ \max_{1 \leq r \leq \left\lceil \frac{n}{m} \right\rceil} \left\{ \frac{\beta}{\beta_\mu^r} \left[m - \frac{(m-1)}{2r} \right] \right\}, \max_{1 \leq i \leq m} \left\{ \beta_i' \left[\nu_i - \frac{(\nu_i - 1)}{2 \left\lceil \frac{n_i'}{\nu_i} \right\rceil} \right] \right\}, \right.$$

$$\left. \max_{1 \leq i \leq m} \left\{ \beta_i' \left[\frac{n_i'+1}{2} \right] \right\}, \; n'_{max} \right\} \; , \qquad (5.5.7)$$

where λ_i', β_i' and ν_i, n'_{max} and λ, β are as defined in Theorems 5.3.1, 5.5.1

and Lemma 5.2.6 respectively, $\mu_r = \min_{J_j \in D_r} \left\{\frac{\sigma_j}{\tau_j}\right\}$ and $\beta_\mu^r = \min_{J_j \in D_r}\left\{ \max_{1\leq k\leq \ell_j} \{b_k\} / \min_{1\leq k\leq \ell_j} \{b_k\}\right\}$

$1 \leq r \leq \left\lceil \frac{n}{m} \right\rceil$.

Proof: From Lemma 5.2.9 we have

$$\bar{\omega}_{QAD} < \tau_1 + 2\tau_2 + \ldots + n\tau_n \ .$$

Nevertheless, since $\tau_j \leq \frac{\sigma_j}{\mu_r}$, where $J_j \in D_r$, $1<r< \left\lceil \frac{n}{m} \right\rceil$ and $1 \leq j \leq n$, the above inequality becomes

$$\bar{\omega}_{QAD} < \sum_{r=1}^{\lceil n/m \rceil} \left[\frac{1}{\mu_r} \left(((r-1)m+1)\sigma_{(r-1)m+1} + \ldots + rm\sigma_{rm} \right) \right] , \qquad (5.5.8)$$

where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$ and $\sigma_{n+1} = \ldots = \sigma_{\left\lceil \frac{n}{m} \right\rceil m} = 0$.

However, from Lemma 5.2.6

$$\bar{\omega}_{OPT} > \frac{1}{\lambda} \left[\sum_{r=1}^{\lceil n/m \rceil} r(\sigma_{(r-1)m+1} + \ldots + \sigma_{rm}) \right] . \qquad (5.5.9)$$

Also, $\bar{\omega}_{QAD}$ and $\bar{\omega}_{OPT}$ can be bounded as

$$\bar{\omega}_{QAD} < \sum_{i=1}^{m} (\sigma_1^{G_i} + 2\sigma_2^{G_i} + \ldots + n_i!\sigma_{n_i!}^{G_i}) \qquad (5.5.10)$$

and

$$\bar{\omega}_{OPT} > \sum_{i=1}^{m} \frac{1}{\lambda_i!} \left[\sum_{r=1}^{\lceil n_i!/\nu_i \rceil} \left(((r-1)\nu_i+1)\sigma_{(r-1)\nu_i+1}^{G_i} + \ldots + r\nu_i \sigma_{r\nu_i}^{G_i} \right) \right] \qquad (5.5.11)$$

or

$$\bar{\omega}_{OPT} > \sum_{i=1}^{m} \frac{1}{\lambda_i!} (\sigma_1^{G_i} + \sigma_2^{G_i} + \ldots + \sigma_{n_i!}^{G_i}) , \qquad (5.5.12)$$

where $\sigma_1^{G_i} \geq \sigma_2^{G_i} \geq \ldots \geq \sigma_{n_i!}^{G_i}$ and $\sigma_{n_{i+1}!}^{G_i} = \ldots = \sigma_{\left\lceil \frac{n_i!}{\nu_i} \right\rceil \nu_i}^{G_i} = 0$, for $1 \leq i \leq m$.

Now, considering the pairs of the inequalities (5.5.8)-(5.5.9), (5.5.10)-(5.5.11) and (5.5.10)-(5.5.12) and following a similar analysis as in Theorem 5.3.3 we obtain respectively:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \max_{1\leq r\leq \left\lceil \frac{n}{m} \right\rceil} \left\{ \frac{\lambda}{\mu_r} \left[m - \frac{(m-1)}{2r} \right] \right\} ,$$

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \max_{1<i<m} \left\{\lambda_i' \left(\nu_i - \frac{(\nu_i-1)}{2\left\lceil \frac{n_i'}{\nu_i} \right\rceil}\right)\right\} ,$$

and

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \max_{1\leq i\leq m} \left\{\lambda_i' \left(\frac{n_i'+1}{2}\right)\right\} .$$

Then, combining the above inequalities with the inequality $(\bar{\omega}_{QAD}/\bar{\omega}_{OPT})<n'_{max}$, which is true since it holds for an arbitrary priority list, we obtain the worst-case bound for non-identical processors given in (5.5.6).

For uniform processors we obtain the worst-case bound (5.5.7), by replacing the inequalities (5.5.8), (5.5.9), (5.5.11) and (5.5.12) with

$$\bar{\omega}_{QAD} < \sum_{r=1}^{\lceil n/m \rceil} \left\lfloor \frac{1}{\beta_\mu^r} \left(((r-1)m+1)\sigma_{(r-1)m+1} + \ldots + rm\sigma_{rm} \right) \right\rfloor , \qquad (5.5.8')$$

$$\bar{\omega}_{OPT} > \frac{1}{\beta} \left[\sum_{r=1}^{\lceil n/m \rceil} r(\sigma_{(r-1)m+1} + \ldots + \sigma_{rm}) \right] , \qquad (5.5.9')$$

$$\bar{\omega}_{OPT} > \sum_{i=1}^{m} \frac{1}{\beta_i'} \left[\sum_{r=1}^{\lceil n_i'/m \rceil} \left(((r-1)\nu_i+1)\sigma_{(r-1)\nu_i+1}^{G_i} + \ldots + r\nu_i\sigma_{r\nu_i}^{G_i} \right) \right] \qquad (5.5.11')$$

and

$$\bar{\omega}_{OPT} > \sum_{i=1}^{m} \frac{1}{\beta_i'} (\sigma_1^{G_i} + \sigma_2^{G_i} + \ldots + \sigma_{n_i'}^{G_i}) \qquad (5.5.12')$$

respectively and following a similar analysis.

Moreover, the bound n'_{max} is a best possible one. This is realised by the following example.

Example 5.2: Let the task system $(J,[t_{ij}])$ be defined by a set of independent jobs $J=\{J_1,J_2,\ldots,J_n\}$ and the $(m\times n)$ matrix

$$[t_{ij}] = \begin{bmatrix} \overbrace{\varepsilon \ldots \varepsilon}^{n_2} & \ldots & \overbrace{\varepsilon \ldots \varepsilon}^{n_{m-1}} & \overbrace{\varepsilon \ldots \varepsilon}^{n_m} & X \\ X+Z \ldots X+Z & \ldots & X+Z \ldots X+Z & X+Z \ldots X+Z & \infty \\ \infty \ldots \infty & \ldots & X+Z \ldots X+Z & X+Z \ldots X+Z & \infty \\ \vdots \quad \vdots & & \vdots \quad \vdots & \vdots \quad \vdots & \vdots \\ \vdots \quad \vdots & & \vdots \quad \vdots & \vdots \quad \vdots & \vdots \\ \vdots \quad \vdots & & X+Z \ldots X+Z & X+Z \ldots X+Z & \infty \\ \infty \ldots \infty & \ldots & \infty \ldots \infty & \varepsilon \ldots \varepsilon & \infty \end{bmatrix},$$

$$(J_1 \ldots, J_{n_2}, \ \ldots \ , J_{\sum_{i=2}^{m-2} n_i + 1} \ldots, J_{\sum_{i=2}^{m-1} n_i}, \ldots, J_{n-1}, J_n)$$

where $n=n_2+n_3+\ldots+n_m$, $\sum_{i=2}^{m-1} n_i > n_m - 1$, $\varepsilon, X, Z \in R^+$ and $\varepsilon << X << Z$.

We can see that the priority list $L=(J_1,J_2,\ldots,J_n)$ is a LTF_{MAX} ordering. The schedule resulting from this priority is shown in Fig.5.3 whereas the corresponding optimal one in Fig.5.4.

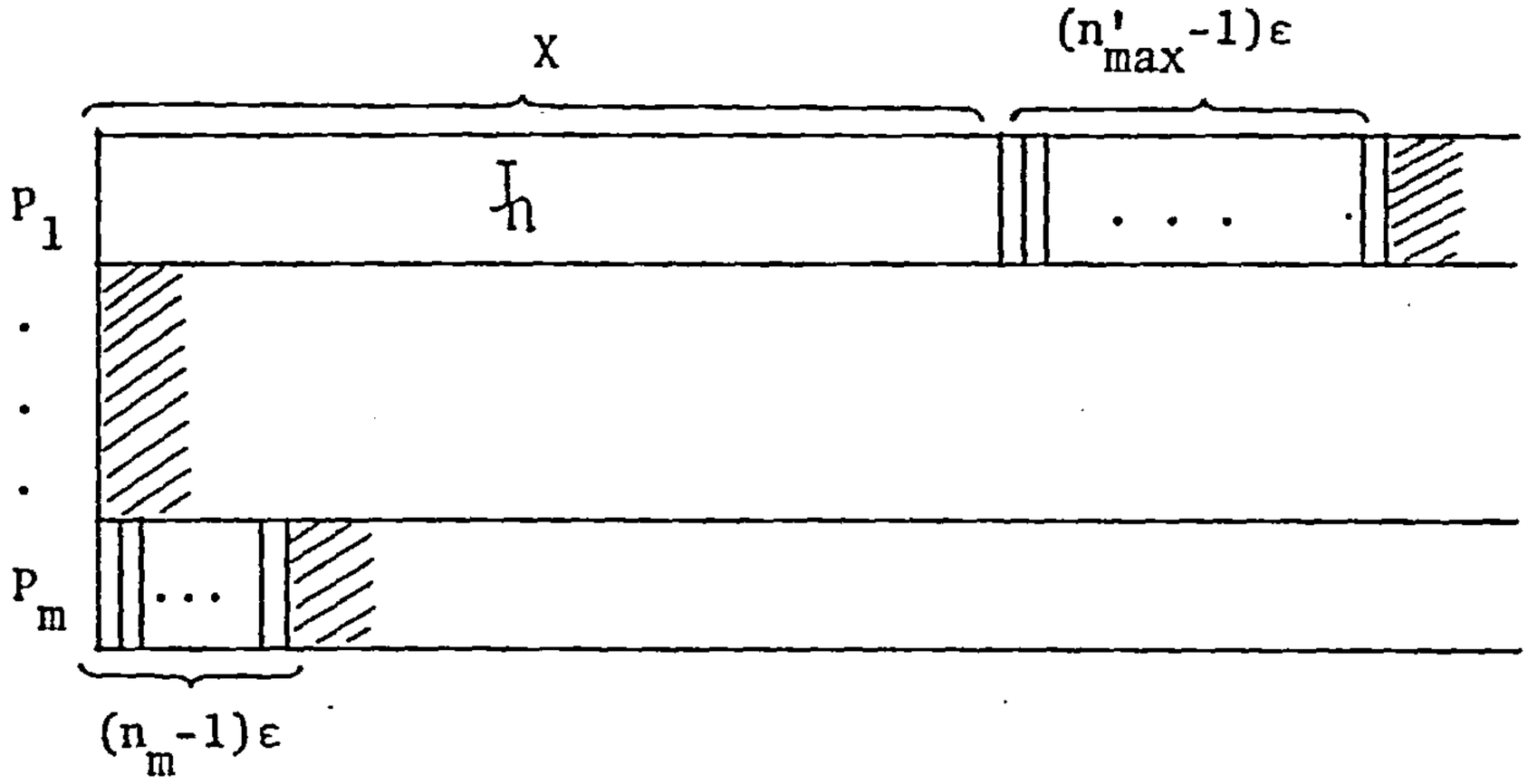


FIGURE 5.3: The schedule resulting from L using the Q.A.D. algorithm

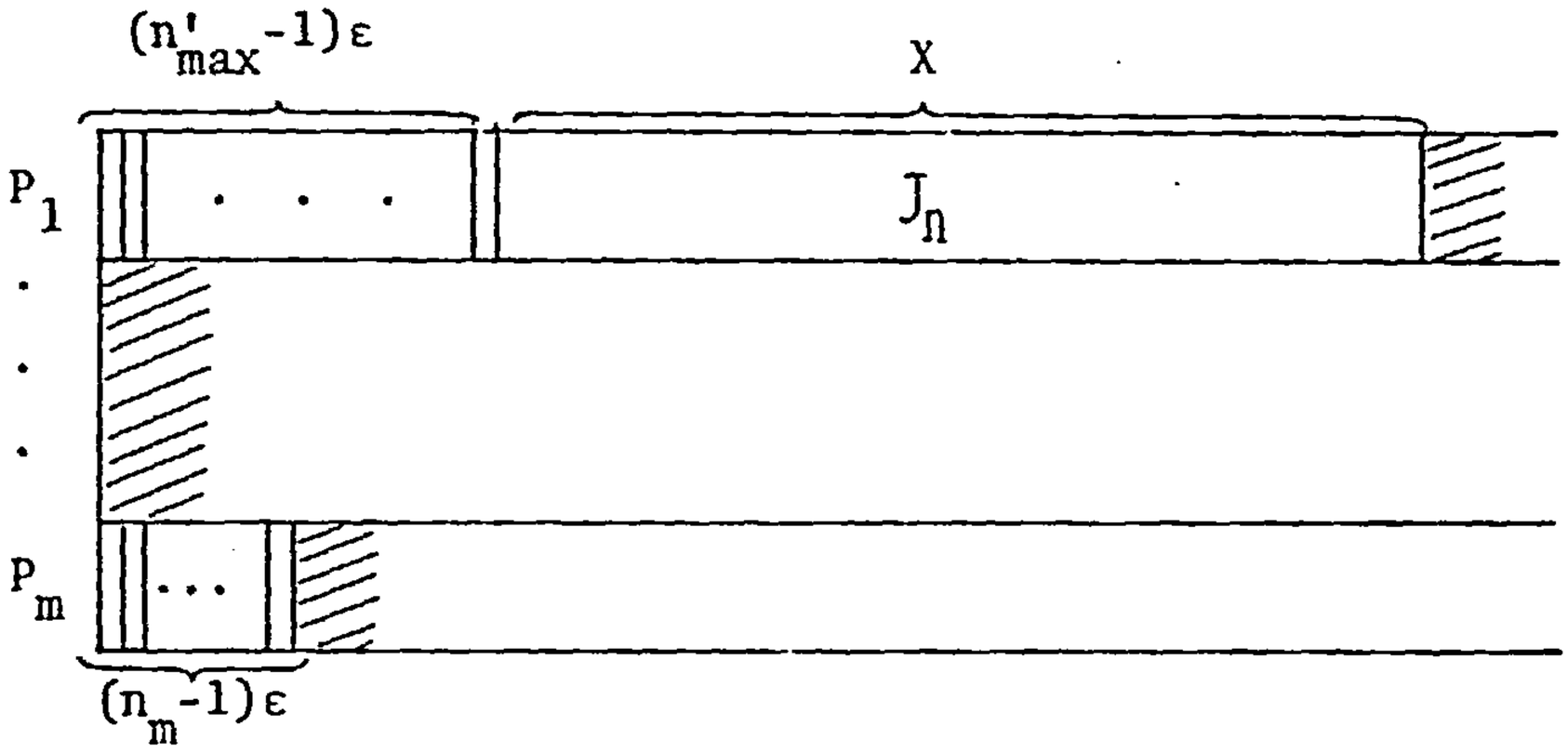


FIGURE 5.4: The optimal schedule for the given task system $(J,[t_{ij}])$ in Example 5.2.

The ratio of the mean flow times of these two schedules is:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} = \frac{n'_{max}X+f_1(\varepsilon)+f'_1(\varepsilon)}{X+f_2(\varepsilon)+f'_1(\varepsilon)}$$

or

$$\lim_{\varepsilon\to 0} \frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} = \frac{n'_{max}X}{X} = n'_{max} \quad ,$$

where $n'_{max} = \left(\sum_{i=1}^{m-1} n_i\right) + 1$.

Further, a similar task system, with the (X+Z) time requirements of the jobs $J_1, J_2, \ldots, J_{n-1}$ being replaced by (2X), can be used to show that n'_{max} is also a best possible bound for the case of uniform processors.

For $\lambda=\lambda'_i=1=\beta=\beta'_i$, $1\leq i\leq m$ and hence $\mu_r=1=\beta^r_\mu$, $1\leq r\leq \left\lceil \frac{n}{m} \right\rceil$, the given bounds in Theorem 5.5.3 agree with the corresponding worst-case bound found for the homogeneous multiprocessor system with independent memories. (See Theorem II.2, Appendix II.)

Furthermore, although n'_{max} is a best possible bound, we cannot accept it as the worst-case performance bound of the Q.A.D. algorithm under the LTF_{MAX} rule, because it is based on pathological examples like the one given, i.e. Example 5.2. So, other bounds have been derived, based on the nature of the priority list and its advantages, which although cannot be reached by examples, in most of the cases they are better than n'_{max}. In addition, these bounds show the ability of the LTF_{MAX} rule to produce better worst-case performance bounds than the one provided by the previously examined procedures. However, comparing the bounds offered by the latest theorem with the corresponding ones of Theorem 5.3.3, we can see that the Q.A.D. algorithm under the LTF_{MAX} rule may have better guaranteed performance levels than the P.D. algorithm under STF_{MAX} rule.

Theorem 5.5.4: Let the priority list be in a LTF_{MIN} ordering. Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \min\left\{\left(m - \frac{(m-1)}{2\left\lceil\frac{n}{m}\right\rceil}\right), \max_{1\leq i\leq m}\left\{\max_{1\leq r\leq\left\lceil\frac{n_i'}{\nu_i}\right\rceil}\left\{\lambda_i'^{r}\left(\nu_i - \frac{(\nu_i-1)}{2r}\right)\right\}\right\},\right.$$

$$\left.\max_{1\leq i\leq m}\left\{\lambda_i'\left(\frac{n_i'+1}{2}\right)\right\}, n'_{max}\right\}; \qquad (5.5.13)$$

(ii) for uniform processors:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \min\left\{\left(m - \frac{(m-1)}{2\left\lceil\frac{n}{m}\right\rceil}\right), \max_{1\leq i\leq m}\left\{\max_{1\leq r\leq\left\lceil\frac{n_i'}{\nu_i}\right\rceil}\left\{\beta_i'^{r}\left(\nu_i - \frac{(\nu_i-1)}{2r}\right)\right\}\right\},\right.$$

$$\left.\max_{1\leq i\leq m}\left\{\beta_i'\left(\frac{n_i'+1}{2}\right)\right\}, n'_{max}\right\}, \qquad (5.5.14)$$

where λ_i', β_i' and ν_i, n'_{max} and $\lambda_i'^{r}, \beta_i'^{r}$ are as defined in Theorems 5.3.1, 5.5.1 and 5.3.4 respectively.

Proof: From Lemmas 5.2.9 and 5.2.5 we have

$$\bar{\omega}_{QAD} \leq \tau_1 + 2\tau_2 + \ldots + n\tau_n \qquad (5.5.15)$$

and

$$\bar{\omega}_{OPT} \geq \sum_{r=1}^{\lceil n/m\rceil}\left[r(\tau_{(r-1)m+1} + \ldots + \tau_{rm})\right], \qquad (5.5.16)$$

where $\tau_1\geq\tau_2\geq\ldots\geq\tau_n$ and $\tau_{n+1}=\ldots=\tau_{\left\lceil\frac{n}{m}\right\rceil m}=0$.

In addition, $\bar{\omega}_{QAD}$ and $\bar{\omega}_{OPT}$ can be bounded by:

$$\bar{\omega}_{QAD} < \sum_{i=1}^{m}\sum_{r=1}^{\lceil n_i'/\nu_i\rceil}\left[\lambda_i'^{r}\left((r-1)\nu_i+1)\overset{G_i}{\tau}_{(r-1)\nu_i+1} + \ldots + r\nu_i\overset{G_i}{\tau}_{r\nu_i}\right)\right] \qquad (5.5.17)$$

and

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m}\sum_{r=1}^{\lceil n_i'/\nu_i\rceil}\left[r(\overset{G_i}{\tau}_{(r-1)\nu_i+1} + \ldots + \overset{G_i}{\tau}_{r\nu_i})\right] \qquad (5.5.18)$$

where $\overset{G_i}{\tau}_1\geq\overset{G_i}{\tau}_2\geq\ldots\geq\overset{G_i}{\tau}_{n_i'}$ and $\overset{G_i}{\tau}_{n_i'+1}=\ldots=\overset{G_i}{\tau}_{\left\lceil\frac{n_i'}{\nu_i}\right\rceil\nu_i}=0$, $1\leq i\leq m$; or

$$\bar{\omega}_{QAD} < \sum_{i=1}^{m}\left[\lambda_i'(\overset{G_i}{\tau}_1 + 2\overset{G_i}{\tau}_2 + \ldots + \overset{G_i}{\tau}_{n_i'})\right] \qquad (5.5.19)$$

and

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m} \left(\tau_1^{G_i} + \tau_2^{G_i} + \ldots + \tau_{n_i'}^{G_i}\right) . \qquad (5.5.20)$$

Consequently, considering the pairs of the inequalities (5.5.15)-(5.5.16), (5.5.17)-(5.5.18) and (5.5.19)-(5.5.20) and following a similar analysis as in Theorem 5.3.3 we get respectively:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \left(m - \frac{(m-1)}{2\left\lceil \frac{n}{m} \right\rceil}\right) ,$$

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \max_{1 \leq i \leq m} \left\{ \max_{1 \leq r \leq \left\lceil \frac{n_i'}{\nu_i} \right\rceil} \left\{ \lambda_i'^{r} \left(\nu_i - \frac{(\nu_i - 1)}{2r} \right) \right\} \right\} ,$$

and

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \max_{1 \leq i \leq m} \left\{ \lambda_i' \left(\frac{n_i' + 1}{2} \right) \right\} .$$

Now, combining the above bounds with the bound $(\bar{\omega}_{QAD}/\bar{\omega}_{OPT}) < n'_{max}$, which comes from Theorem 5.5.1, we obtain the upper bound (5.5.13). However, working similarly we can obtain the bound (5.5.14) for the case of uniform processors.

Although the bound $\left(m - \frac{(m-1)}{2\left\lceil \frac{n}{m} \right\rceil}\right)$ seems much better than the others, for $\lambda_i'^{r}$ or $\beta_i'^{r}$ very close to 1 or $n'_{max} \leq m-1$ this might not be the upper bound of the Q.A.D. algorithm under the LTF_{MIN} rule. However, except those cases, where the λ_i' and β_i' are very close to 1, this bound is much better than the worst-case performance bounds of the Q.A.D. algorithm under the LTF_{MAX} rule (see Theorem 5.5.3) and hence, even better than the corresponding ones of the P.D. algorithm under the STF_{MIN} or STF_{MAX} rules (see Theorems 5.3.3 and 5.3.4). Furthermore, it should be noticed that if the equality holds in (5.5.15) then this Q.A.D. algorithm produces optimal schedules.

Finally, if $\lambda_i' = \lambda_i'^{r} = 1 = \beta_i' = \beta_i'^{r}$ we get agreement with the bound found for

the homogeneous multiprocessor system with independent memories. (See Theorem II.2, Appendix II.)

Since we have already established worst-case performance bounds for the RAND, LMT, STF, LMST and LTF ordering rules we now turn to examine the LMLT ordering rules, which will complete the analysis of the Q.A.D. algorithm.

Theorem 5.5.5: Let the priority list be in a $LMLT_{MAX}$ ordering. Then, for non-identical processors:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \min \left\{ \max_{1\leq i\leq m} \left\{ \lambda_i \left(i - \frac{(i-1)}{2\left\lceil \frac{n_i}{i} \right\rceil} \right) \right\} , \max_{1\leq i\leq m} \left\{ \lambda_i \left(\frac{n_i+1}{2} \right) \right\}, \; n'_{max} \right\} , \tag{5.5.21}$$

where λ_i and n'_{max} are as defined in Theorems 5.3.5 and 5.5.1 respectively.

Proof: The quantities $\bar{\omega}_{QAD}$ and $\bar{\omega}_{OPT}$ are bounded by the following upper and lower bounds respectively:

$$\bar{\omega}_{QAD} < \sum_{i=1}^{m} (\sigma_1^{F_i} + 2\sigma_2^{F_i} + \ldots + n_i \sigma_{n_i}^{F_i}) \tag{5.2.22}$$

and

$$\bar{\omega}_{OPT} > \sum_{i=1}^{m} \frac{1}{\lambda_i} \left[\sum_{r=1}^{\lceil n_i/i \rceil} r(\sigma_{(r-1)i+1}^{F_i} + \ldots + \sigma_{ri}^{F_i}) \right] \tag{5.2.23}$$

or

$$\bar{\omega}_{OPT} > \sum_{i=1}^{m} \frac{1}{\lambda_i} (\sigma_1^{F_i} + \sigma_2^{F_i} + \ldots + \sigma_{n_i}^{F_i}) , \tag{5.2.24}$$

where $\sigma_1^{F_i} \geq \sigma_2^{F_i} \geq \ldots \geq \sigma_{n_i}^{F_i}$ and $\sigma_{n_i+1}^{F_i} = \ldots = \sigma_{\left\lceil \frac{n_i}{i} \right\rceil i}^{F_i} = 0$ for i=1(1)m.

Now, for the pairs of inequalities (5.5.22)-(5.5.23) and (5.5.22)-(5.5.24) we follow a similar analysis as the one used in Theorem 5.3.5 and we get respectively:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \max_{1\leq i\leq m} \left\{ \lambda_i \left(i - \frac{(i-1)}{2\left\lceil \frac{n_i}{i} \right\rceil} \right) \right\}$$

and

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \max_{1\leq i\leq m} \left\{ \lambda_i \left(\frac{n_i+1}{2} \right) \right\} .$$

Furthermore, combining the above bounds with the $(\bar{\omega}_{QAD}/\bar{\omega}_{OPT})<n'_{max}$, which obviously holds (see Theorem 5.5.1), we obtain the upper bound 5.3.2 1. Moreover, the bound n'_{max} is a best possible one and this is realised by the following example.

Example 5.3: Let the task system $(J,[t_{ij}])$ be defined by the set of independent jobs $J=\{J_1,J_2,\ldots,J_n\}$ and the (m×n) matrix:

$$[t_{ij}] = \begin{array}{c} \overbrace{\quad}^{n_2} \quad\quad\quad \overbrace{\quad}^{n_{m-2}}\ \overbrace{\quad}^{n_{m-1}}\ \overbrace{\quad}^{n_m} \\ \left[\begin{array}{ccccccc} \varepsilon\ldots\varepsilon & . & . & . & \varepsilon\ldots\varepsilon & X & \varepsilon\ldots\varepsilon \\ X\ldots X & . & . & . & X\ldots X & X+Z & X\ldots X \\ \infty\ldots\infty & . & . & . & X\ldots X & X+Z & X\ldots X \\ & & & & X\ldots X & X+Z & . \quad . \\ & & & & \vdots \quad \vdots & \vdots & . \quad . \\ & & & & X\ldots X & X+Z & . \quad . \\ & & & & \infty\ldots\infty & X+Z & X\ldots X \\ \infty\ldots\infty & . & . & . & \infty\ldots\infty & \infty & \varepsilon\ldots\varepsilon \end{array}\right] \\ (J_1, \quad . \quad . \quad . \quad , J_n) \end{array},$$

where $n=\sum_{i=2}^{m} n_i$, $\sum_{i=2}^{m-1} n_i > n_m$, $\varepsilon,X,Z \in R^+$ and $\varepsilon<<X<<Z$.

Clearly, the priority list $L=(J_1,J_2,\ldots,J_n)$ is a $LMLT_{MAX}$ ordering. The schedule resulting from this priority list and the corresponding optimal schedule are shown in Fig.5.5 and 5.6 respectively.

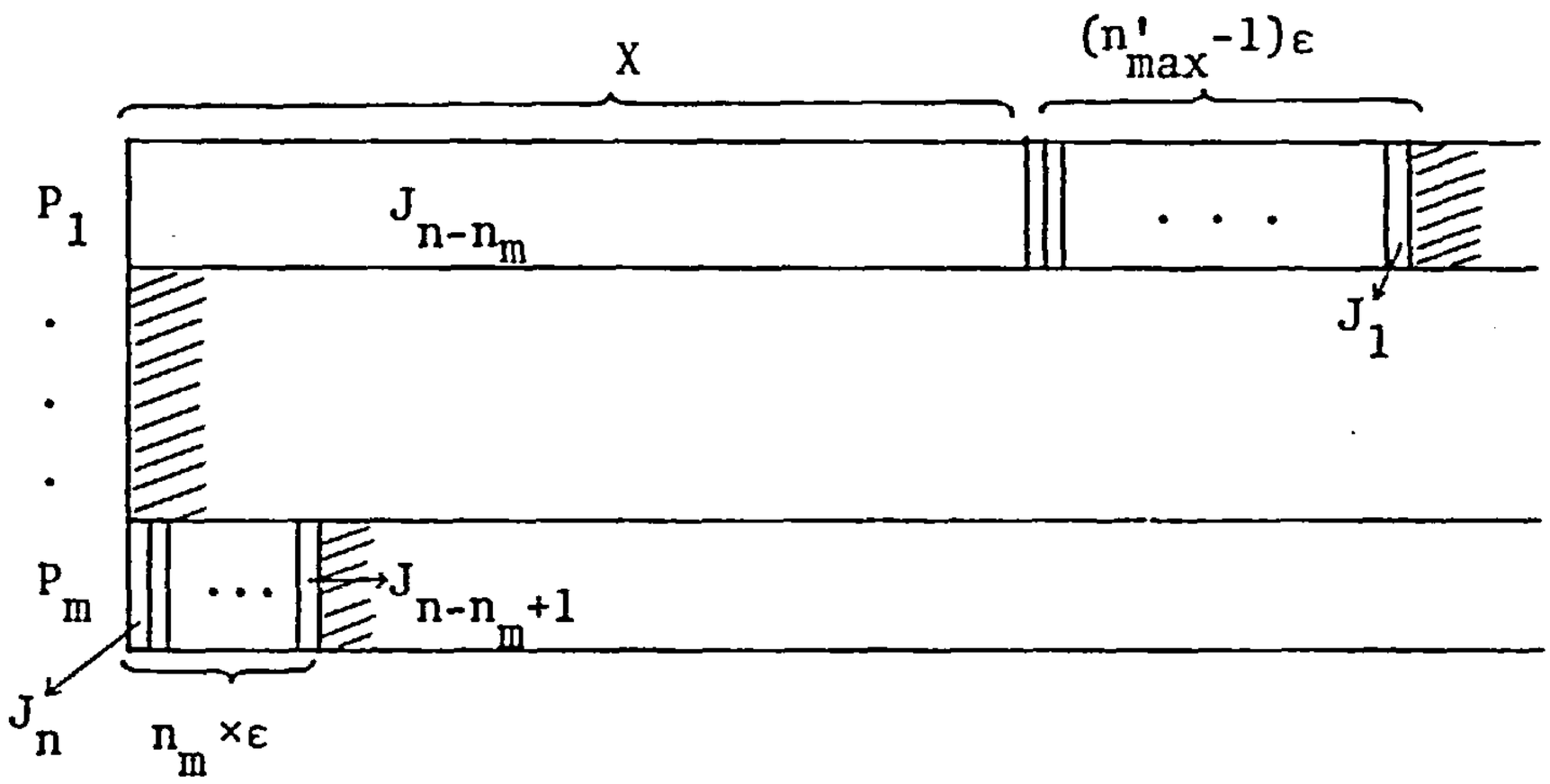


FIGURE 5.5: Worst-case schedule illustrating Theorem 5.5.5.

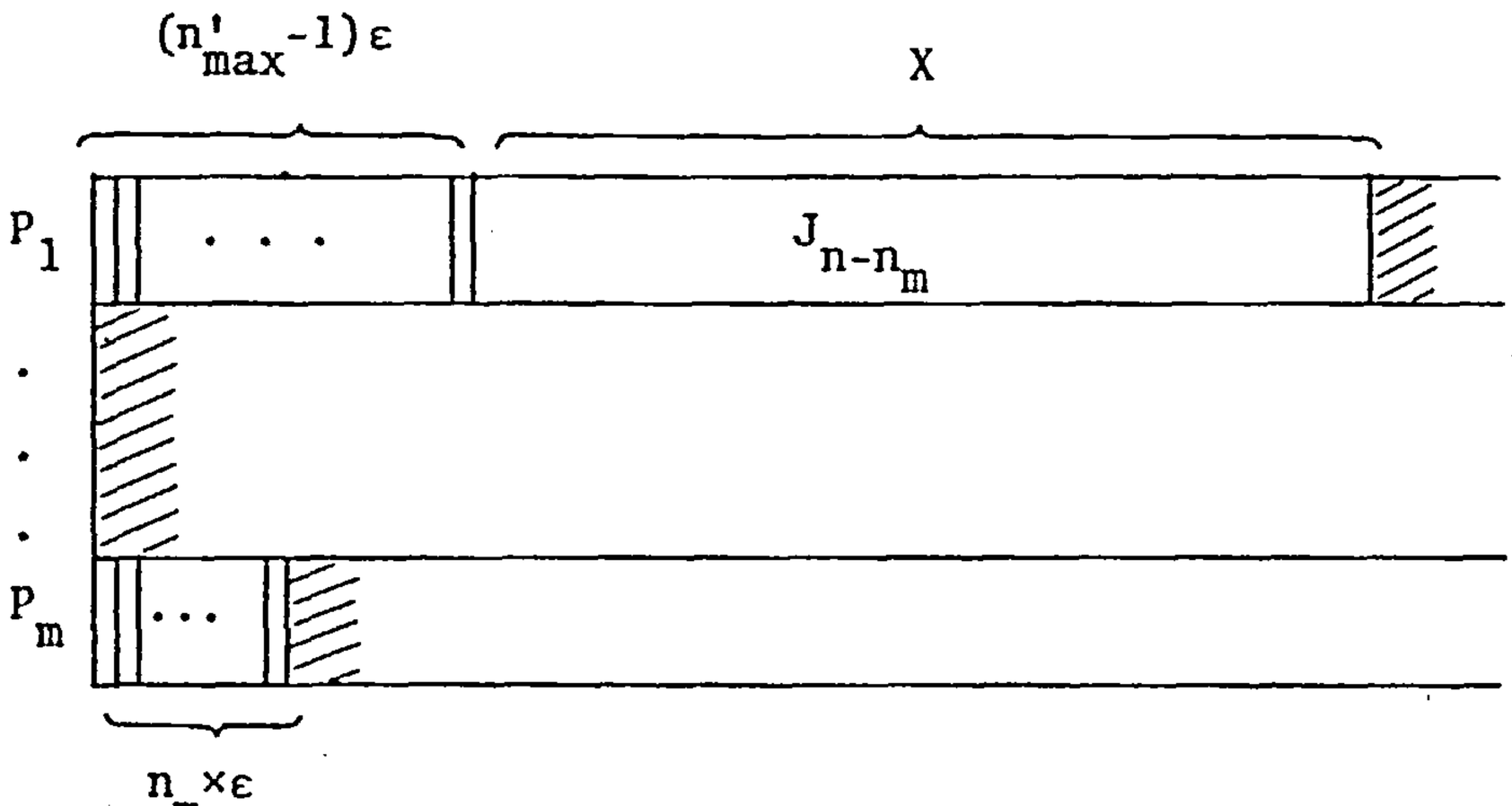


FIGURE 5.6: Optimal schedule illustrating Theorem 5.5.5.

The ratio of the mean flow times of these two schedules is:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} = \frac{n'_{max}X+f_1(\varepsilon)+f'_1(\varepsilon)}{X+f_2(\varepsilon)+f'_1(\varepsilon)}$$

or

$$\lim_{\varepsilon\to 0} \frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} = \frac{n'_{max}X}{X} = n'_{max} \quad \square$$

As it was previously stated the $LMLT_{MAX}$ and $LMLT_{MIN}$ orderings are exactly the same for the case of uniform processors. So, the corresponding bound of the next theorem applies here as well.

Theorem 5.5.6: Let the priority list be in a $LMLT_{MIN}$ ordering. Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \min\left\{\max_{1\le i\le m}\left\{\max_{1\le r\le\left\lceil\frac{n_i}{i}\right\rceil}\left\{\lambda_i^r\left[i-\frac{(i-1)}{2r}\right]\right\}\right\}, \max_{1\le i\le m}\left\{\lambda_i\left[\frac{n_i+1}{2}\right]\right\}, n'_{max}\right\}; \tag{5.5.25}$$

(ii) for uniform processors:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \min\left\{\max_{1\le i\le m}\left\{\beta_i \max_{1\le r\le\left\lceil\frac{n_i}{i}\right\rceil}\left\{i-\frac{(i-1)}{2r}\right\}\right\}, \max_{1\le i\le m}\left\{\beta_i\left[\frac{n_i+1}{2}\right]\right\}, n'_{max}\right\}, \tag{5.5.26}$$

where λ_i, λ_i^r and β_i, and n'_{max} are as defined in Theorems 5.3.5, 5.3.6 and 5.5.1 respectively.

Proof: The values of $\bar{\omega}_{QAD}$ and $\bar{\omega}_{OPT}$ are bounded respectively as follows:

$$\bar{\omega}_{QAD} < \sum_{i=1}^{m} \sum_{r=1}^{\lceil n_i/i \rceil} \left[\lambda_i^r \left(((r-1)i+1) \tau^{F_i}_{(r-1)i+1} + \ldots + ri\tau^{F_i}_{ri} \right) \right] \qquad (5.5.27)$$

or

$$\bar{\omega}_{QAD} < \sum_{i=1}^{m} [\lambda_i (\tau^{F_i}_1 + 2\tau^{F_i}_2 + \ldots + n_i \tau^{F_i}_{n_i})] \qquad (5.5.28)$$

and

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m} \sum_{r=1}^{\lceil n_i/i \rceil} [r(\tau^{F_i}_{(r-1)i+1} + \ldots + \tau^{F_i}_{ri})] \qquad (5.5.29)$$

or

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m} (\tau^{F_i}_1 + \ldots + \tau^{F_i}_{n_i}) \ , \qquad (5.5.30)$$

where $\tau^{F_i}_1 \geq \tau^{F_i}_2 \geq \ldots \geq \tau^{F_i}_{n_i}$ and $\tau^{F_i}_{n_i+1} = \ldots = \tau^{F_i}_{\lceil \frac{n_i}{i} \rceil i} = 0$ for i=1(1)m.

For the pairs of inequalities (5.5.27)-(5.5.29) and (5.5.28)-(5.5.50) an analysis similar to the one used in Theorem 5.3.5 will result in the following upper bounds respectively:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \max_{1 \leq i \leq m} \left\{ \max_{1 \leq r \leq \left\lceil \frac{n_i}{i} \right\rceil} \left\{ \lambda_i^r \left(i - \frac{(i-1)}{2r} \right) \right\} \right\}$$

and

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < \max_{1 \leq i \leq m} \left\{ \lambda_i \left(\frac{n_i+1}{2} \right) \right\} \ .$$

Those two bounds together with the $(\bar{\omega}_{QAD}/\bar{\omega}_{OPT}) < n'_{max}$, which is obviously because of Theorem 5.5.1, result in the upper bound (5.5.25) for the case of non-identical processors. However, replacing the inequalities (5.5.27) and (5.5.28) by $\bar{\omega}_{QAD} < \sum_{i=1}^{m} \beta_i (\tau^{F_i}_1 + 2\tau^{F_i}_2 + \ldots + n_i \tau^{F_i}_{n_i})$, and following a similar analysis as above, we obtain the worst-case bound for uniform processors (5.5.26).

Finally, since the priority list L of the Example 5.3 is also a $LMLT_{MIN}$ ordering, the example can also be used to show that n'_{max} is a best

possible bound for case (i) of this theorem. Moreover, if the (X+Z) time requirements of the J_{n-n_m} job are replaced by (X^2/ϵ) then, the same example shows that n'_{max} is a best possible bound for the case of uniform processors as well□

As in Theorem 5.5.3, although n'_{max} is a best possible bound for the $LMLT_{MAX}$ or $LMLT_{MIN}$ ordering rule, we cannot accept it as the worst-case performance bound since we have found bounds which appear to be better than n'_{max} in many cases.

Now, comparing the bounds of Theorems 5.5.5 and 5.5.6 with the corresponding ones of Theorems 5.5.3 and 5.5.4 we can see that the utilisation of memory, in contributing to the worst-case performance of the Q.A.D. algorithm under the LTF rules, when the mean flow time is chosen as the performance criterion, does not offer any clear improvement. In particular, although for the LTF_{MAX} rule the utilisation of memory may improve the guaranteed performance levels, for the LTF_{MIN} rule the utilisation of memory worsen the worst-case bounds if λ_i^r or β_i^r are not very close to 1. However, the worst-case performance bounds may be better when the priority list is in the $LMLT_{MIN}$ ordering rather than in $LMLT_{MAX}$ ordering, since $\lambda_i^r \leq \lambda_i$.

On the other hand, the extreme performance bounds which have been derived for the QAD algorithm under the LMLT ordering rules are better than the corresponding ones found for the P.D. algorithm under the LMST rules. Moreover, this is true even for schedules where $n'_i=n_i$, $1\leq i\leq m$ (i.e. k=0). Whenever such schedules are derived then, no matter what the values of n_i are, for non-identical processors $(\bar{\omega}_{QAD}/\bar{\omega}_{OPT})<\lambda$ whereas for the uniform processors case $\bar{\omega}_{QAD}=\bar{\omega}_{OPT}$ (optimal schedules).

Also, we should notice the agreement of these bounds with the corresponding one found for the homogeneous multiprocessor system with independent memories, as $\lambda_i^r=\lambda_i=1=\beta_i$, $1\leq i\leq m$. (See Theorem II.3, Appendix II.)

Finally, concluding this section we shall point out that although the n'_{max} bound is a best possible one for all but the LTF_{MIN} rule, for the LTF_{MAX}, $LMLT_{MAX}$ and $LMLT_{MIN}$ rules, when the Q.A.D. algorithm is used, other bounds have also been found, which appear to be better than n'_{max} in many cases. This means, that best possible bounds based on pathological situations might not be accepted as the worst-case performance bounds of the algorithm when better and more informative bounds can be found for a number of special cases. Here, as for the P.D. scheduling algorithms, the worst-case performance bounds of Q.A.D. algorithms can vary widely when simple heuristic procedures are used to form the priority list.

5.6 TWO-PHASE QUICK AND DIRTY (Q.A.D.*) SCHEDULING ALGORITHMS

The aims of the present section are similar to the ones for which section 5.4 was established, i.e., whether or not the Q.A.D.* algorithms can offer an improvement, in a worst-case sense, over the corresponding Q.A.D. algorithms.

Let $\bar{\omega}_{QAD*}$ be the mean flow time of the schedule constructed by the Q.A.D.* algorithm, when the priority list is formed by a heuristic ordering procedure, and $\bar{\omega}_{OPT}$ be the mean flow time of an optimal schedule for a given task system $(\mathcal{J},[t_{ij}])$, $1 \leq i \leq m$, $1 \leq j \leq n$.

In the remainder, we present a number of theorems, which provide worst-case performance bounds for those Q.A.D.* algorithms whose value of $\bar{\omega}_{QAD*}$ can be bounded better than the value of $\bar{\omega}_{QAD}$ for the corresponding Q.A.D. algorithm.

Theorem 5.6.1: Let the priority list be in an arbitrary or STF_{MAX} ordering. Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{QAD*}}{\bar{\omega}_{OPT}} < \min \left\{ \max_{1\leq i\leq m} \left\{ \lambda_i^! \left[\nu_i - \frac{(\nu_i - 1)}{2\left\lceil \frac{n_i}{\nu_i} \right\rceil} \right] \right\} , \max_{1\leq i\leq m} \left\{ \lambda_i^! \left[\frac{n_i^! + 1}{2} \right] \right\} , n'_{max} \right\} ; \tag{5.6.1}$$

(ii) for uniform processors:

$$\frac{\bar{\omega}_{QAD*}}{\bar{\omega}_{OPT}} < \min \left\{ \max_{1\leq i\leq m} \left\{ \beta_i^! \left[\nu_i - \frac{(\nu_i - 1)}{2\left\lceil \frac{n_i}{\nu_i} \right\rceil} \right] \right\} , \max_{1\leq i\leq m} \left\{ \beta_i^! \left[\frac{n_i^! + 1}{2} \right] \right\} , n'_{max} \right\} , \tag{5.6.2}$$

where $\lambda_i^!, \beta_i^!$, ν_i and n'_{max} are as defined in Theorems 5.3.1 and 5.5.1.

Proof: When the priority list is in a RAND or STF_{MAX} ordering then the values of $\bar{\omega}_{QAD*}$ and $\bar{\omega}_{OPT}$ are bounded as their corresponding ones in Theorem 5.4.1. This means that the upper bounds (5.4.1) and (5.4.2) are valid even if the Q.A.D.* algorithm is used to construct the schedules and the priority list is in the RAND or STF_{MAX} ordering. However, because of Theorem 5.5.1 we

also have $(\bar{\omega}_{QAD*}/\bar{\omega}_{OPT})<n'_{max}$. Therefore, combining these upper bounds we obtain the worst-case bounds (5.6.1) and (5.6.2) for non-identical and uniform processors respectively.

Theorem 5.6.2: Let the priority list be in a STF_{MIN} ordering. Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{QAD*}}{\bar{\omega}_{OPT}} < \min\left\{ \max_{1\le i\le m} \left\{ \max_{1\le r\le \left\lceil \frac{n'_i}{\nu_i} \right\rceil} \left\{ \lambda'^{r}_{i} \left[\nu_i - \frac{(\nu_i - 1)}{2r} \right] \right\} \right\} , \max_{1\le i\le m} \left\{ \lambda'_i \left[\frac{n'_i + 1}{2} \right] \right\}, n'_{max} \right\}; \quad (5.6.6)$$

(ii) for uniform processors:

$$\frac{\bar{\omega}_{QAD*}}{\bar{\omega}_{OPT}} < \min\left\{ \max_{1\le i\le m} \left\{ \max_{1\le r\le \left\lceil \frac{n'_i}{\nu_i} \right\rceil} \left\{ \beta'^{r}_{i} \left[\nu_i - \frac{(\nu_i - 1)}{2r} \right] \right\} \right\} , \max_{1\le i\le m} \left\{ \beta'_i \left[\frac{n'_i + 1}{2} \right] \right\}, n'_{max} \right\}; \quad (5.6.7)$$

where $\lambda'_i, \beta'_i, \nu_i$ and $\lambda'^{r}_{i}, \beta'^{r}_{i}$ are as defined in Theorems 5.3.1 and 5.3.4 respectively.

Proof: Here, the values of the quantities $\bar{\omega}_{QAD*}$ and $\bar{\omega}_{OPT}$ are bounded as their corresponding ones in Theorem 5.4.2. Therefore, following a similar analysis we will find the first two factors of the worst-case bounds (5.6.6) and (5.6.7). Furthermore, $(\bar{\omega}_{QAD*}/\bar{\omega}_{OPT})<n'_{max}$ as indicated in the previous theorem. Thus, we finally obtain the bounds (5.6.6) and (5.6.7) for non-identical and uniform processors respectively.

Theorem 5.6.3: Let the priority list be in a $LMST_{MAX}$ or LMF ordering. Then, for non-identical processors,

$$\frac{\bar{\omega}_{QAD*}}{\bar{\omega}_{OPT}} < \min\left\{ \max\left\{ \max_{1\le i\le m-1} \left\{ \lambda_i \max_{1\le r\le \left\lceil \frac{n'_i}{i} \right\rceil} \left\{ \frac{k}{r} + i - \frac{(i-1)}{2r} \right\} \right\} , \lambda_m \left[m - \frac{(m-1)}{2\left\lceil \frac{n_m}{m} \right\rceil} \right] \right\} ,$$

$$\max\left\{ \max_{1\le i\le m-1} \left\{\lambda_i \left[k + \frac{n_i+1}{2}\right]\right\},\ \lambda_m \left[\frac{n_m+1}{2}\right]\right\},\ n'_{max} \quad ,$$

$$\max_{1\le i\le m}\left\{\lambda'_i \left[\nu_i - \frac{(\nu_i-1)}{2\left\lceil \frac{n'_i}{\nu_i}\right\rceil}\right]\right\},\quad \max_{1\le i\le m}\left\{\lambda'_i \left[\frac{n'_i+1}{2}\right]\right\}\ \Bigg\} \quad , \qquad (5.6.8)$$

where λ'_i, ν_i and λ_i,k are as defined in Theorems 5.3.1 and 5.3.5 respectively.

Proof: The values of the quantities $\bar{\omega}_{QAD*}$ and $\bar{\omega}_{OPT}$ are bounded as their corresponding ones in Theorem 5.4.3. Therefore, the upper bound of that theorem is held here as well. However, because of Theorem 5.5.1 we also have $(\bar{\omega}_{QAD*}/\bar{\omega}_{OPT})<n'_{max}$. Now, combining these two upper bounds we obtain the worst-case performance bound (5.6.8), which proves the theorem.

Theorem 5.6.4: Let the priority list be in a $LMST_{MIN}$ ordering. Then,

(i) for non-identical processors:

$$\frac{\bar{\omega}_{QAD*}}{\bar{\omega}_{OPT}} < \min\left\{\max\left\{ \max_{1\le i\le m-1}\left\{ \max_{1\le r\le\left\lceil\frac{n_i}{i}\right\rceil}\left\{\lambda_i^r\left[\frac{k}{r}+i-\frac{(i-1)}{2r}\right]\right\}\right\},\ \max_{1\le r\le\left\lceil\frac{n_m}{m}\right\rceil}\left\{\lambda_m^r\left[m-\frac{(m-1)}{2r}\right]\right\}\right\},\right.$$

$$\max\left\{ \max_{1\le i\le m-1}\left\{\lambda_i\left[k+\frac{(n_i+1)}{2}\right]\right\},\ \lambda_m\left[\frac{n_m+1}{2}\right]\right\},\ n'_{max} \quad ,$$

$$\left.\max_{1\le i\le m}\left\{\lambda'_i\left[\nu_i-\frac{(\nu_i-1)}{2\left\lceil\frac{n'_i}{\nu_i}\right\rceil}\right]\right\},\quad \max_{1\le i\le m}\left\{\lambda'_i\left[\frac{n'_i+1}{2}\right]\right\}\right\} \quad ; \qquad (5.6.9)$$

(ii) for uniform processors:

$$\frac{\bar{\omega}_{QAD*}}{\bar{\omega}_{OPT}} < \min\left\{\max\left\{ \max_{1\le i\le m-1}\left\{\beta_i \max_{1\le r\le\left\lceil\frac{n_i}{i}\right\rceil}\left\{\frac{k}{r}+i-\frac{(i-1)}{2r}\right\}\right\},\ \beta_m\left[m-\frac{(m-1)}{2\left\lceil\frac{n_m}{m}\right\rceil}\right]\right\},\right.$$

$$\max\left\{ \max_{1\leq i\leq m-1} \left\{\beta_i\left[k + \frac{n_i+1}{2}\right]\right\} , \quad \beta_m\left[\frac{n_m+1}{2}\right]\right\} \quad , n'_{max} \quad ,$$

$$\max\left\{\beta'_i\left[\nu_i - \frac{(\nu_i-1)}{2\left\lceil\frac{n'_i}{\nu_i}\right\rceil}\right]\right\} , \quad \max_{1\leq i\leq m}\left\{\beta'_i\left[\frac{n_i+1}{2}\right]\right\}\right\} , \qquad (5.6.10)$$

where λ'_i, β'_i and ν_i, k and λ_i, λ_i^r and β_i are as defined in Theorems 5.3.1, 5.3.5, and 5.3.6.

Proof: When the priority list is in $LMST_{MIN}$ ordering then the values of $\bar{\omega}_{QAD*}$ and $\bar{\omega}_{OPT}$ are bounded as their corresponding ones in Theorem 5.4.4. Thus, the upper bounds of that theorem are held here as well. In addition, $(\bar{\omega}_{QAD*}/\bar{\omega}_{OPT}) < n'_{max}$ as can be resulted in from Theorem 5.5.1. Thus, combining these upper bounds we obtain the worst-case bounds (5.6.9) and (5.6.10) for non-identical and uniform processors respectively.□

We recall that when the priority list is in LMF or $LMST_{MAX}$ ordering then for the case of uniform processors the upper bound (5.6.10) is held.

Finally, if the priority list is in LTF_{MAX}, LTF_{MIN}, $LMLT_{MAX}$ or $LMLT_{MIN}$ ordering, the value of $\bar{\omega}_{QAD*}$ is bounded as when the Q.A.D. algorithm, under these ordering rules, is used to construct the schedules, i.e., the values of $\bar{\omega}_{QAD}$ in Theorems 5.5.3, 5.5.4, 5.5.5 and 5.5.6 respectively. Therefore, the Q.A.D.* algorithm under the LTF or LMLT ordering rules does not offer any improvement over the worst-case performance bounds of the corresponding Q.A.D. algorithms. The only exception is that the bound n'_{max} is not a best possible one any more.

Before we finish the present section, one could notice that the Q.A.D.* algorithm produces informative guaranteed performance levels, which are better than the corresponding Q.A.D. one, only when the priority list is in the RAND, STF, LMF or LMST ordering. Also, not one of the worst-case performance bounds found for the Q.A.D.* algorithms is a best possible one.

Furthermore, although the bounds of the Q.A.D.* algorithms are close to each other, they are not so close as the corresponding ones of the P.D.* algorithms. However, generally the worst-case bounds of the Q.A.D.* algorithms may be better than the corresponding bounds of the P.D.* algorithms.

Moreover, the bounds of all the theorems 5.6.1, 5.6.2, 5.6.3 and 5.6.4 for $\lambda_i=\lambda_i'=\lambda_i'^{r}=1=\beta_i=\beta_i'=\beta_i^{r}$ agree with the bounds which have been found for the homogeneous multiprocessor system with independent memories. (See Theorems II.4 and II.5, Appendix II.)

CHAPTER 6

DETERMINISTIC ANALYSIS OF HEURISTIC SCHEDULING ALGORITHMS - COMPLETION TIME PERFORMANCE CRITERION

6.1 INTRODUCTION

In the previous chapter we have analysed a variety of non-preemptive algorithms under several heuristic ordering procedures and we have proven worst-case performance bounds for each of them, when the mean flow time is chosen as the performance criterion. However, in order to achieve completely the objectives of this thesis, as far as the deterministic analysis is concerned, guaranteed performance levels must also be established for these algorithms when the completion time is the performance criterion. As a matter of fact, this is the aim of the present chapter.

We recall, that the completion time (or maximum finishing time) of a given schedule S is the total time it takes to complete the execution of all jobs of the task system according to that schedule i.e., $\omega=\max_{1\leq i\leq n}\{f_i(S)\}$.

The following two sections are dedicated to P.D. and Q.A.D. algorithms. The remainder P.D.* or Q.A.D.* algorithms are not considered since $\omega_{PD}=\omega_{PD*}$ and $\omega_{QAD}=\omega_{QAD*}$.

However, before we start analysing the algorithms let us define U_i to be the busy time of the processor P_i, $1\leq i\leq m$ (i.e., $U_i = \sum_{J_j\in G_i} t_{ij}$) and I_i to be the corresponding idle time of the P_i processor (i.e., $I_i=\omega-U_i$).

6.2 PRIORITY DRIVEN (P.D.) SCHEDULING ALGORITHMS

As it was indicated the completion time will be the performance criterion throughout this chapter. However, first we shall derive worst-case performance bounds for the P.D. algorithm under the heuristic ordering rules already used in the previous chapter.

Let ω_{PD} be the completion time of the schedule constructed by the P.D. algorithm, when the priority list is formed by a heuristic ordering procedure, and ω_{OPT} be the length of the optimal schedule for a given task system $(\mathcal{J},[t_{ij}])$; $1 \leq i \leq m$, $1 \leq j \leq n$.

Theorem 6.2.1: Let the priority list be in an arbitrary ordering (RAND). Then,

(i) for non-identical processors:

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \lambda\left[r + 2 + \frac{u}{2^r(\ell+1)}\right] \leq \lambda\left[2 + \log_2\left(\frac{m}{\ell+1}\right)\right]$$

for $2 \leq \ell \leq m-1$,

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 1 + \lambda\left(r + 1 + \frac{u}{2^{r+1}}\right) \leq 1 + \lambda\log_2(m)$$

for $\ell=1$, and

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \lambda\left(2 - \frac{1}{m}\right) \quad \text{for } \ell=m;$$

(ii) for uniform processors:

$$\max\left\{\frac{\omega_{PD}}{\omega_{OPT}}\right\} \geq 1 + \sum_{j=\ell+1}^{m} \frac{b_j}{\sum_{i=1}^{j-1} b_i} \quad \text{for } 1 \leq \ell \leq m-1 \text{ and}$$

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 1 + \frac{b_{max}}{b_{min}} - \frac{b_{max}}{\sum_{i=1}^{m} b_i} \quad \text{for } \ell=m,$$

where λ is as defined in Lemma 5.2.6, $\ell=\ell_w$ and J_w is the longest job to finish at time ω_{PD}, $m=2^r(\ell+1)+u$, and r is the greatest integer such that $2^r(\ell+1) \leq m$.

Proof: (i) When the non-identical processors characterise the heterogeneous

multiprocessor system then, three guaranteed performance levels will be derived which depend on the value of the parameter ℓ.

(1) Let $2 \leq \ell \leq m-1$. In such a case the proof will be by contradiction, which is similar to the one used for the corresponding algorithm in [Kaf] and [KS2]. Suppose that there is a task system for which $\omega_{PD} > \lambda[r+2+\frac{(m-2^r(\ell+1))}{2^r(\ell+1)}]\omega_{OPT}$. We shall analyse the structure of such a schedule and derive several conditions that must be satisfied by the given task system.

The schedule produced by the P.D. algorithm is divided from right-to-left into r+3 intervals $(d_1,d_2,\ldots,d_{r+3})$. Each of the first r+2 intervals has length $\lambda\omega_{OPT}$ and hence they occupy the time period $[\omega_{PD}-\lambda(r+2)\omega_{OPT},\omega_{PD}]$ in the schedule. However, the i^{th} interval is the half open time period $[\omega_{PD}-\lambda i\omega_{OPT},\omega_{PD}-\lambda(i-1)\omega_{OPT})$, $1 \leq i \leq r+2$ while the final interval d_{r+3} occupies the time period $[0,\omega_{PD}-\lambda(r+2)\omega_{OPT})$.

Now, let us assume:

a_i to be the number of processors containing no idle time in the interval d_i,

x_i to be the largest label ($\max\{\ell_j\}$) of any task beginning in the interval d_i, and J_j

T_i to be the total busy time of the processors in the closed interval $[\omega_{PD}-\lambda i\omega_{OPT},\omega_{PD}]$.

The following relationships apply to the interval d_1,

$$a_1 \geq 1 , \tag{6.2.1}$$

$$T_1 \geq \lambda\omega_{OPT} , \tag{6.2.2}$$

and $$x_1 \geq \ell . \tag{6.2.3}$$

Relation (6.2.1) is true since every interval must have at least one processor which is continuously active. This also implies relation (6.2.2) since the length of d_1 is $\lambda\omega_{OPT}$. Nevertheless, since no job can have a time requirement greater than $\lambda\omega_{OPT}$, at least one task begins in d_1. Thus, relation (6.2.3) follows immediately as the task J_w,

which finishes at ω_{PD}, has the label $\ell_w=\ell$ and its time requirement cannot exceed the $\lambda\omega_{OPT}$ units of time.

Further, we derive three recursive formulae defining a_i,T_i, and x_i for $2\leq i\leq r+2$. First,

$$a_i \geq x_{i-1} \quad . \tag{6.2.4}$$

This can be realised since, the existance of at least one job in the interval d_{i-1} which can fit into the x_{i-1} largest memories means that the corresponding processors cannot be idle before the beginning of that job. Consequently, these processors cannot be idle during the interval d_i.

The second formula follows immediately from the definitions of a_i and T_i,

$$T_i \geq T_{i-1} + a_i\lambda\omega_{OPT} \quad . \tag{6.2.5}$$

This can also be written as:

$$T_i \geq a_i\lambda\omega_{OPT} + a_{i-1}\lambda\omega_{OPT}+\ldots+a_1\lambda\omega_{OPT}$$

or

$$T_i \geq \lambda\omega_{OPT}\left(\sum_{j=1}^{i} a_j\right) \quad . \tag{6.2.6}$$

Finally, the formula which defines x_i is,

$$x_i \geq \sum_{j=1}^{i} a_j \quad . \tag{6.2.7}$$

However, we need first to show that the inequality

$$x_i \geq x_{i-1} \tag{6.2.8}$$

holds. In the case where the inequality (6.2.8) was not true, there would exist a job in d_{i-1} which could fit into some memory where no task beginning in d_i could fit into. This would imply that this task could have commenced earlier and hence the demand scheduling principle would have been violated. Therefore, the inequality (6.2.8) must hold in any valid schedule.

Consider the interval $[\omega_{PD}-\lambda i\omega_{OPT},\omega_{PD}]$. By definition T_i is the total busy time of the processor in this interval. On the other hand,

inequality (6.2.8) implies that the memories which can contain the tasks in this interval are the x_i largest ones. Thus,

$$\omega_{OPT} \geq T_i/\lambda x_i$$

or

$$x_i \geq T_i/\lambda\omega_{OPT} .$$

Substituting T_i from (6.2.6) in the last inequality we obtain

$$x_i \geq \sum_{j=1}^{i} a_j ,$$

which proves the inequality (6.2.7).

Moreover, using the induction process the following inequalities can be shown to be true for $2 \leq i \leq r-2$:

$$x_i \geq 2^{i-2}(\ell+1) \qquad (6.2.9)$$

and

$$T_i \geq 2^{i-2}(\ell+1)\lambda\omega_{OPT} . \qquad (6.2.10)$$

Initial step: From (6.2.7) for i=2 we have

$$x_2 \geq \sum_{j=1}^{2} a_j = a_1 + a_2 \geq \ell+1 = 2^0(\ell+1) ,$$

while from (6.2.5), we have

$$T_2 \geq T_1 + a_2\lambda\omega_{OPT} \geq \lambda\omega_{OPT} + \ell\lambda\omega_{OPT}$$

or

$$T_2 \geq (\ell+1)\lambda\omega_{OPT} = 2^0(\ell+1)\lambda\omega_{OPT} ,$$

because $a_1 \geq 1$, $a_2 \geq x_1 \geq \ell$ and $T_1 \geq \lambda\omega_{OPT}$. Therefore, the inequalities (6.2.9) and (6.2.10) hold for i=2.

Induction step: Assuming that the inequalities (6.2.9) and (6.2.10) hold for any $i<k$, where $3 \leq k \leq r+2$, we will show that they hold for i=k as well.

We have

$$x_k \geq \sum_{j=1}^{k} a_j = a_1 + a_2 + \sum_{j=3}^{k} a_j$$

or

$$x_k \geq 1 + \ell + \sum_{j=3}^{k} x_{j-1} . \qquad (6.2.11)$$

According to the induction assumption,

$$x_i \geq 2^{i-2}(\ell+1), \qquad \text{for } 2 \leq i \leq k.$$

Thus, the inequality (6.2.11) becomes

$$x_k \geq (1+\ell) + \sum_{j=3}^{k} 2^{j-3}(\ell+1)$$

or

$$x_k \geq (\ell+1)[1 + \sum_{j=3}^{k} 2^{j-3}]$$

or

$$x_k \geq 2^{k-2}(\ell+1) \ . \tag{6.2.12}$$

Furthermore,

$$T_k \geq T_{k-1} + a_k \lambda \omega_{OPT}$$

or

$$T_k \geq 2^{k-3}(\ell+1)\lambda\omega_{OPT} + x_{k-1}\lambda\omega_{OPT} \tag{6.2.13}$$

since from inequality (6.2.4) and induction hypothesis for T_i we have $a_k \geq x_{k-1}$ and $T_{k-1} \geq 2^{k-3}(\ell+1)\lambda\omega_{OPT}$ respectively. Finally, since $x_{k-1} \geq 2^{k-3}(\ell+1)$, from inequality (6.2.13) we obtain

$$T_k \geq 2^{k-2}(\ell+1)\lambda\omega_{OPT} \ . \tag{6.2.14}$$

The inequalities (6.2.12) and (6.2.14) complete the induction step and hence (6.2.9) and (6.2.10) hold for any $2 \leq i \leq r+2$.

Now, in the time period $[\omega_{PD}-(r+2)\lambda\omega_{OPT}, \omega_{PD}]$ the processors are kept busy for T_{r+2} units of time. Thus, from (6.2.10) we can obtain $T_{r+2} \geq 2^r(\ell+1)\lambda\omega_{OPT}$. This implies that at most $(m-2^r(\ell+1))\lambda\omega_{OPT}$ schedule time has been left to be used in the interval d_{r+3}, since the total time cannot exceed the $m\lambda\omega_{OPT}$ units. Since $x_{r+2} \geq 2^r(\ell+1)$ according to the inequality (6.2.9) for i=r+2, there exists at least one task with label $2^r(\ell+1)$ which commences in the interval d_{r+2}. This indicates that at least $2^r(\ell+1)$ processors contain no idle time in the interval d_{r+3} and hence, the length of that interval is at most $(m-2^r(\ell+1))\lambda\omega_{OPT}/2^r(\ell+1)$. However, since each of the other r+2 intervals is of length $\lambda\omega_{OPT}$, we obtain

$$\omega_{PD} \leq \lambda[r + 2 + (m-2^r(\ell+1))/2^r(\ell+1)]\omega_{OPT}.$$

This final inequality contradicts our original assumption and therefore, the worst-case bound for part (i) of the theorem, when $1 \leq \ell \leq (m-1)$, has been partly proven.

In order to complete the proof we shall show that

$$2 + r + \frac{u}{2^r(\ell+1)} \leq 2 + \log_2(\frac{m}{\ell+1}) \ , \qquad (6.2.15)$$

where $m=2^r(\ell+1)+u$ and $0 \leq u \leq 2^r(\ell+1)$. Define a function

$$f(r,u) = 2 + \log_2(\frac{2^r(\ell+1)+u}{(\ell+1)}) - (2 + r + \frac{u}{2^r(\ell+1)})$$

$$= \log_2(\frac{2^r(\ell+1)+u}{(\ell+1)}) - r - \frac{u}{2^r(\ell+1)} \ .$$

Consider a fixed, arbitrary value of r. Then,

$$\frac{\partial f}{\partial u} = \frac{1}{(\ell n2)(2^r(\ell+1)+u)} - \frac{1}{2^r(\ell+1)}$$

and

$$\frac{\partial^2 f}{\partial u^2} = - \frac{1}{(\ell n2)(2^r(\ell+1)+u)^2} < 0.$$

If $\frac{\partial f}{\partial u}=0$, then $m=\frac{2^r(\ell+1)}{\ell n2}$ is a local maxima. The local minima must occur at the end points $u=0$ and $u=2^r(\ell+1)$. At these points $f(r,0)=f(r,2^r(\ell+1))=0$. Thus, $f(r,u) \geq 0$. Finally, the definition of $f(r,u)$ implies the inequality (6.2.15).

(2) Now let $\ell=1$. To prove the first part of the worst-case bound we follow a contradicting process similar to the one used in (1) of this theorem. The reason which causes the slightly different bound is that the initial relation (6.2.2) must be substituted by the relation

$$T_1 \geq \omega_{OPT} \ . \qquad (6.2.2')$$

The second part can be established by following a similar mathematical analysis to the one used in (1) for the corresponding case. Therefore, eventually we obtain the bound given in the theorem for $\ell=1$.

(3) Finally, if $\ell=m$ we use a different approach to prove the guaranteed performance level for this case. We have

$$\omega_{PD} = \frac{1}{m}[U_1 + I_1 + U_2 + I_2 + \ldots + U_m + I_m]$$

or equivalently,

$$\omega_{PD} = \frac{1}{m}\left[\frac{1}{\lambda}\left(\sum_{i=1}^{m} U_i\right)+\left(\frac{\lambda-1}{\lambda}\right)\left(\sum_{i=1}^{m}(U_i+I_i)\right)+\frac{1}{\lambda}\left(\sum_{i=1}^{m} I_i\right)\right] \qquad (6.2.16)$$

Clearly,

$$\left.\begin{aligned} &\omega_{PD} = U_i + I_i \qquad \text{for } i=1(1)m, \\ &\omega_{OPT} \geq \frac{1}{m}\left(\sum_{j=1}^{n}\tau_j\right) \geq \frac{1}{\lambda m}\left(\sum_{j=1}^{n}\sigma_j\right) \geq \frac{1}{\lambda m}\left(\sum_{i=1}^{m} U_i\right), \\ &I_i \leq \lambda\omega_{OPT}^{\dagger} \text{ for } i=1(1)m;\ i\neq x \end{aligned}\right\} \qquad (6.2.17)$$

and $I_x = 0$,

where x is the index of the processor where J_w is allocated.

Using the relationships (6.2.17), equation (6.2.16) becomes:

$$m\omega_{PD} \leq m\omega_{OPT} + m\omega_{PD} - \frac{m}{\lambda}\omega_{PD} + \frac{(m-1)}{\lambda}\lambda\omega_{OPT}$$

and thus,

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \lambda\left(2 - \frac{1}{m}\right),$$

which completes the proof for part (i) of the theorem.

Moreover, there are task systems for which the P.D. algorithm constructs schedules whose completion times deviate from their optimal finishing times by the amount allowed by the proven worst-case bounds so far. Such task systems are given in the Examples 6.2 and 6.3, which are incorporated into the proof of the next theorem.

(ii) In the case of uniform processors, to prove the lower limit of the worst-case performance of the P.D. algorithm under an arbitrary ordering rule a method will be given which achieves the bound for arbitrary m and $1\leq\ell\leq m-1$.

The general example consists of $(m-\ell+2)$ groups of jobs. In the priority list the jobs in the $(k+1)^{st}$ group precede those in group k, $1\leq k\leq(m-\ell+1)$.

---

†*This is because* $\sigma_j \leq \lambda\omega_{OPT}$ *for every job (since no parts of the same job can be executed simultaneously on two or more processors) and* $I \leq \sigma_j$ *(since no processor is idle before* s_w*).*

The number of tasks in the last $(m-\ell+1)$ groups is $(k+\ell-2)$, $2\leq k\leq(m-\ell+2)$, while in the first group there are ℓ jobs. The memory and time requirements of the jobs in the group $k=(m-\ell+2)$ are given by $(J_{j_k}, |P_{j_k}|, \{(j_k-1)\varepsilon b_{j_k}\})$, $1\leq j_k\leq m$ and $\varepsilon>0$ is a very small quantitly; for the latter $(m-\ell)$ groups are given by $(J_k, |P_{k+\ell-1}|, \{b_{j_k} b_{k+\ell-1}/(\sum_{i=1}^{k+\ell-2} b_i)\})$, $1\leq j_k\leq(k+\ell-2)$, $2\leq k\leq(m-\ell+1)$; and for the first group are given by $(J_{j_1}, |P_\ell|, \{b_{j_1}\})$, $1\leq j_1\leq\ell$. The jobs in each group appear in the priority list in an increasing sequence according to their job-index. We should notice that the $(k+\ell-2)$ jobs of each group k, $2\leq k\leq(m-\ell+2)$ and the ℓ jobs of the first group terminate within an ε-time difference as the processors index increases. So, for each group k, $1\leq k\leq(m-\ell+1)$, the i^{th} processor becomes available before the $(i+1)^{th}$ processor for $1\leq i\leq(k+\ell-2)$. However, the contribution of each group k, $2\leq k\leq(m-\ell+1)$, to the schedule length is $(b_{k+\ell-1}/\sum_{i=1}^{k+\ell-2} b_i)$ while the contribution of the jobs in the first group is 1. Therefore, the total schedule length is $[1+\sum_{j=\ell+1}^{m}(b_j/\sum_{i=1}^{j-1} b_i)+(\ell-1)\varepsilon]$. The appearance of the general schedule produced by the P.D. algorithm for the above mentioned task system is shown in Fig.6.1. Now, we shall present that an optimal schedule of length $[1+(m-1)\varepsilon]$ can be formed for the given task system. Clearly, if each processor i, $\ell\leq i\leq m$, executes the jobs of the $(i-\ell+1)^{th}$ group and if each processor i_1, $1\leq i_1\leq\ell$, executes the job J_{j_1}, $1\leq j_1\leq\ell$, of the first group while the jobs in the $(m-\ell+2)^{th}$ group being scheduled as in the general schedule, then the completion time will be $[1+(m-1)\varepsilon]$. Therefore, the bound produced by this example achieves the bound given in the theorem as $\varepsilon\to 0$.

Now, for $\ell=m$ we have

$$\omega_{PD} = \frac{1}{m}\left[\sum_{i=1}^{m}(U_i+I_i)\right]$$

or equivalently,

$$\omega_{PD} = \frac{1}{m}\left[\frac{1}{b_{max}}\left(\sum_{i=1}^{m}(U_i b_i)\right) + \sum_{i=1}^{m}\left(\frac{b_{max}-b_i}{b_{max}}\right)(U_i+I_i) + \frac{1}{b_{max}}\left(\sum_{i=1}^{m}(I_i b_i)\right)\right] \qquad (6.2.18)$$

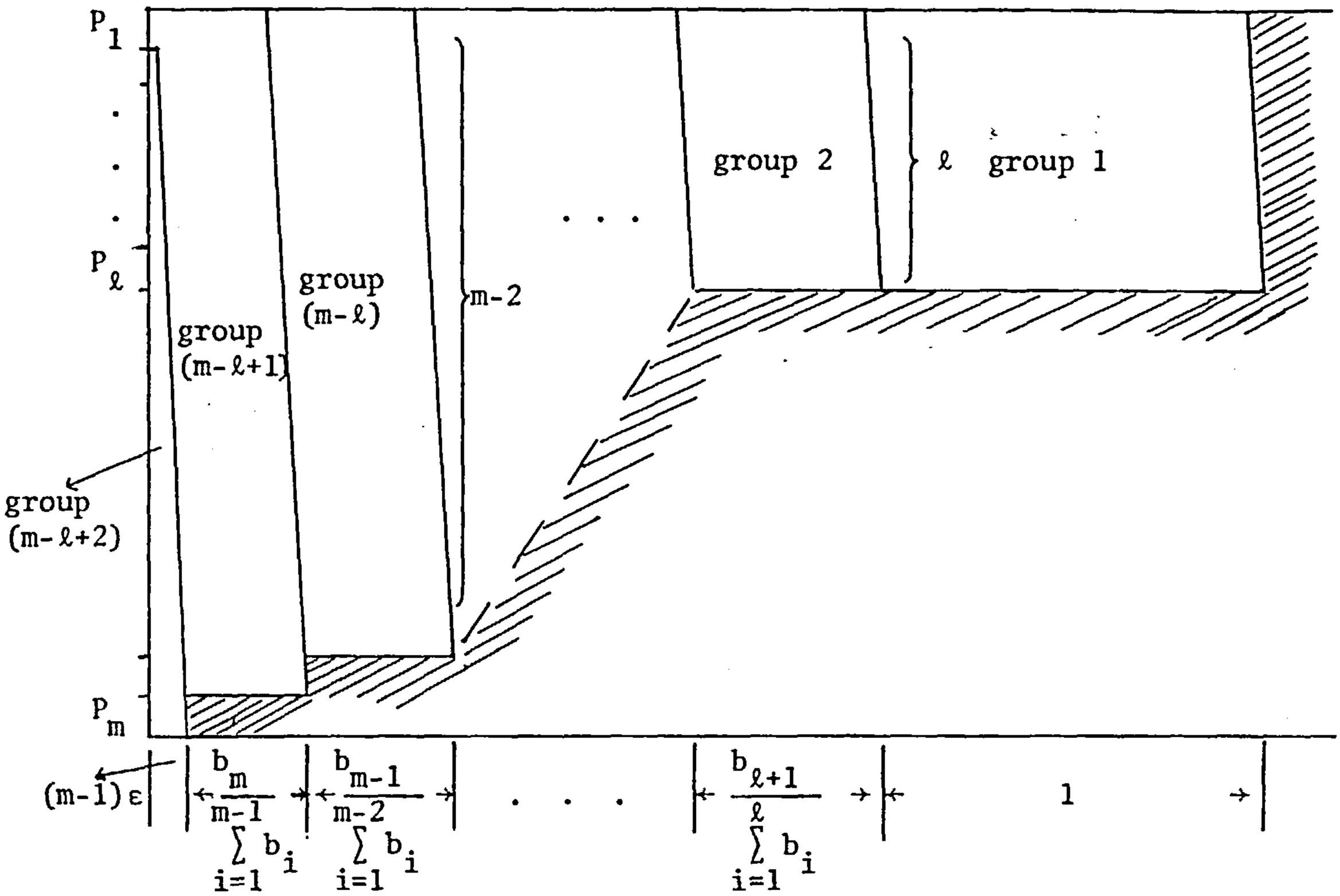


FIGURE 6.1: General schedule produced by the P.D. algorithm

However,

$$\left.\begin{aligned} \omega_{PD} &= U_i + I_i \ , \qquad 1 \leq i \leq m, \\ \omega_{OPT} &\geq \Big(\sum_{i=1}^{m} (U_i b_i)\Big) \Big/ \Big(\sum_{i=1}^{m} b_i\Big) \ , \\ I_i &\leq \frac{b_{max}}{b_{min}} \omega_{OPT} \ , \quad 1 \leq i \leq m, \quad i \neq x \end{aligned}\right\} \qquad (6.2.19)$$

and $$I_x = 0 \ ,$$

where x is the index of the processor where J_w is allocated.

Because of the relations (6.2.19), the equation (6.2.18) becomes:

$$m\omega_{PD} \leq \frac{\omega_{OPT}}{b_{max}} \Big(\sum_{i=1}^{m} b_i\Big) + m\omega_{PD} - \frac{\omega_{PD}}{b_{max}} \Big(\sum_{i=1}^{m} b_i\Big) + \frac{\omega_{OPT}}{b_{min}} \Big(\sum_{\substack{i=1 \\ i \neq x}}^{m} b_i\Big)$$

or

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 1 + \frac{b_{max}}{b_{min}} - \frac{b_{max}}{\sum_{i=1}^{m} b_i} \ , \qquad (6.2.20)$$

which completes the proof of part (ii) and hence the theorem is proven.

Moreover, the bound (6.2.20) is a best possible one and can be realised by the following example.

Example 6.1: For simplicity, let us assume that P_1 is the processor with the minimum processing speed and P_m the processor with the maximum speed. In such a heterogeneous multiprocessor system with uniform processors let the task system $(J,\{m_j\},\{T_j\})$ be defined by:

$$J_1 \quad : (|P_1|, \{\varepsilon b_1\})$$

$$J_j \quad : (|P_j|, \{2\varepsilon b_j\}) \quad \text{for } j=2(1)m$$

$$J_{im+j} : \left(|P_j|, \left\{\frac{b_i}{\sum_{i=1}^{m} b_i} b_j\right\}\right) \quad \text{for } j=1(1)m, \quad i=1(1)m-1$$

and $J_{m^2+1} : (|P_m|, \{b_m\})$.

The schedule resulting from the priority list $L=(J_1,J_2,\ldots,J_{m^2+1})$ using the P.D. algorithm is shown in Fig.6.2, whereas the optimal schedule for the given task system is given in Fig.6.3.

Clearly, the ratio of the completion times of these two schedules is

$$\frac{\omega_{PD}}{\omega_{OPT}} = \left[\left(\sum_{i=1}^{m-1} b_i / \sum_{i=1}^{m} b_i\right) + (b_m/b_1) + \varepsilon\right] /(1+2\varepsilon)$$

or

$$\frac{\omega_{PD}}{\omega_{OPT}} = 1 + \frac{b_m}{b_1} - \frac{b_m}{\sum_{i=1}^{m} b_i} ,$$

as $\varepsilon \to 0$, and since $b_{max}=b_m$ and $b_{min}=b_1$, it becomes

$$\frac{\omega_{PD}}{\omega_{OPT}} = 1 + \frac{b_{max}}{b_{min}} - \frac{b_{max}}{\sum_{i=1}^{m} b_i} ,$$

which is the predicted value of the bound (6.2.22) ¤

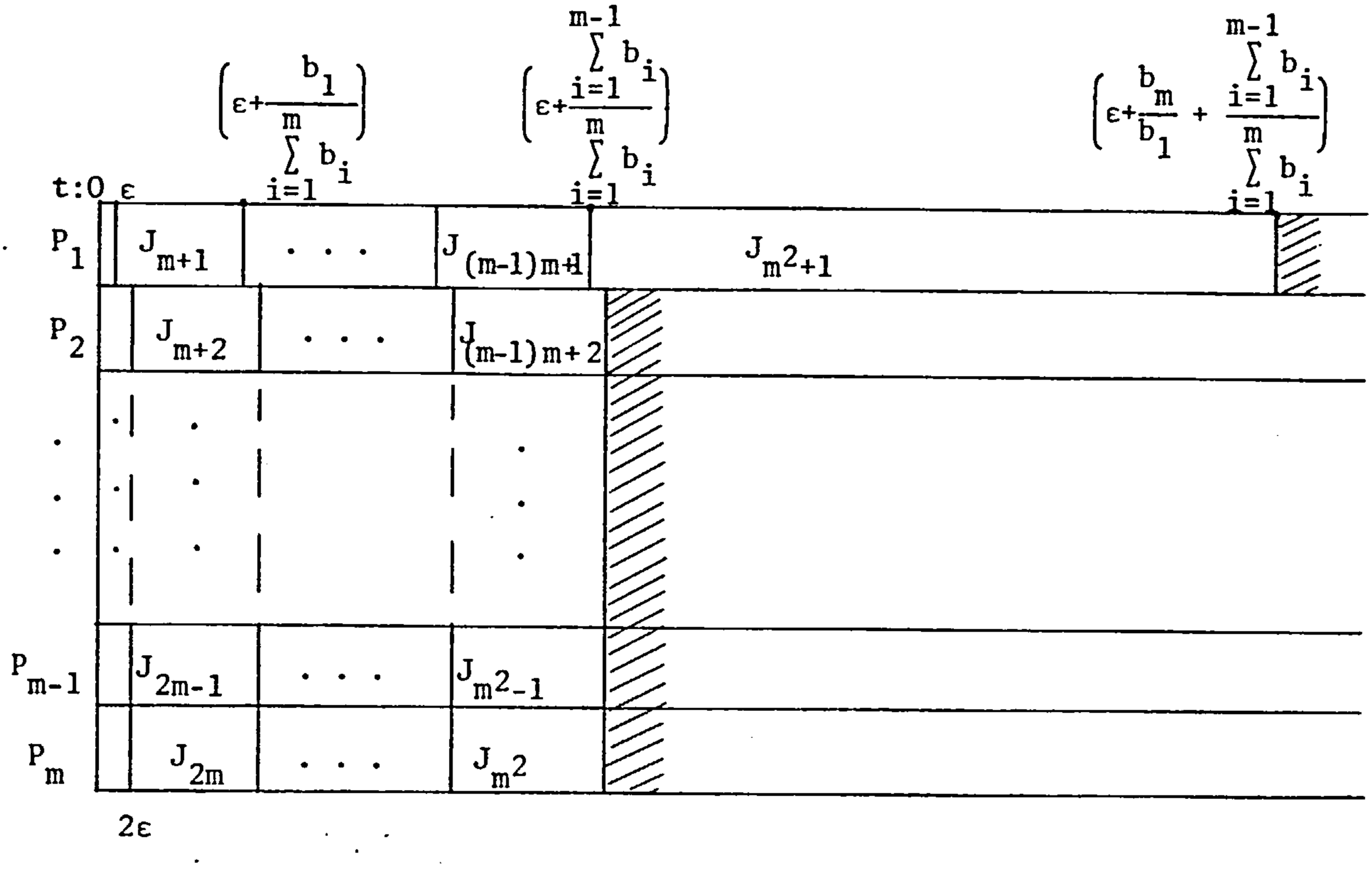


FIGURE 6.2: Schedule constructed by the P.D. algorithm for the given task system in Example 6.1

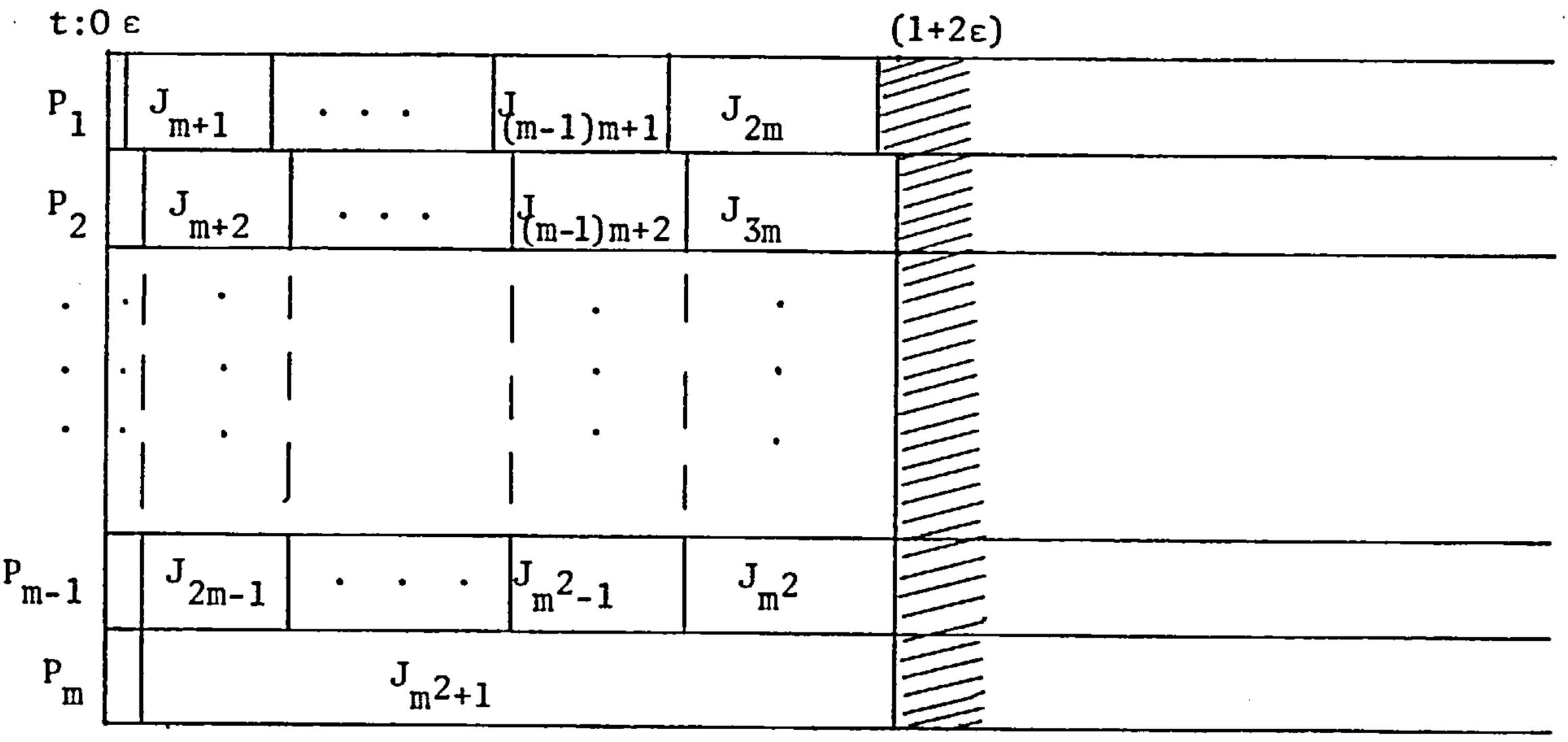


FIGURE 6.3: Optimal schedule for the given task system in Example 6.1

As can be seen, the worst-case bound, when non-identical processors characterise the heterogeneous multiprocessor system, increases logarithmically as m increases and ℓ is constant ($1 \lesssim \ell \lesssim m-1$). The extreme performance depends also from the value of λ. In particular, when the deviation of the jobs time requirements amongst the processors increases, the value of the bound increases as well. Furthermore, although we have not found upper bounds for the worst-case when there are uniform processors, the proven lower limit indicates a similar, logarithmic type behaviour of the algorithm as one of the quantites m or ℓ increases and the other remains constant.

When $\lambda=1$ the bounds found in the (i) part of the theorem agree with the corresponding worst-case bound established for the homogeneous multiprocessor system with independent memories. (See Theorem III.1, Appendix III) Finally, when $\ell=m$ the worst-case bound proved here for uniform processors agrees with the corresponding one found in [LiL1] for the classical heterogeneous multiprocessor system with uniform processors. Actually, we should expect this to be so, because the nature of the algorithm and the fact that $\ell=m$ degenerate the memory constraint.

Theorem 6.2.2: Let the priority list be in a STF_{MIN} or STF_{MAX} ordering. Then,

(i) for non-identical processors:

$$\frac{\omega_{PD}}{\omega_{OPT}} \lesssim \lambda\left[r + 2 + \frac{u}{2^r(\ell+1)}\right] \lesssim \lambda\left[2 + \log_2\left(\frac{m}{\ell+1}\right)\right] \quad \text{for } 2 \lesssim \ell \lesssim m-1,$$

$$\frac{\omega_{PD}}{\omega_{OPT}} \lesssim 1 + \lambda\left(r + 1 + \frac{u}{2^{r+1}}\right) \lesssim 1 + \lambda\log_2(m), \quad \text{for } \ell=1, \text{ and}$$

$$\frac{\omega_{PD}}{\omega_{OPT}} \lesssim \lambda\left(2 - \frac{1}{m}\right) \quad \text{for } \ell=m;$$

(ii) for uniform processors:

$$\max\left\{\frac{\omega_{PD}}{\omega_{OPT}}\right\} \gtrsim 1 + \sum_{j=\ell+1}^{m} \left(\frac{b_j}{\sum_{i=1}^{j-1} b_i}\right) \quad \text{for } 1 \lesssim \ell \lesssim m-1, \text{ and}$$

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 1 + \frac{b_{max}}{b_{min}} - \frac{b_{max}}{\sum_{i=1}^{m} b_i} \quad \text{for} \quad \ell=m,$$

where λ and ℓ,r,u are as defined in Lemma 5.2.6 and Theorem 6.2.1 respectively.

Proof: (i) Since the upper bounds for non-identical processors in Theorem 6.2.1 were proved for an arbitrary priority list, these bounds can be considered as extreme performance levels for any priority list when the P.D. algorithm is used to construct the schedule. In particular, we shall present examples which can cause the performance of the P.D. algorithm under STF_{MIN} or STF_{MAX} ordering rules to deviate from optimal completion time by the value allowed by the worst-case performance bounds stated in the theorem for non-identical processors. First, we present an example which attains the bound for the case where $2 \leq \ell \leq m-1$.

Example 6.2: Let the task system $(J,[t_{ij}])$ be defined by the set of independent jobs $J=\{J_1,J_2,\ldots,J_n\}$ and the following $(m \times n)$ matrix, where $\varepsilon>0$ is a very small quantity, $u=m-2^r(\ell+1)$, w the largest integer which satisfies the conditions $w>0$, $w \leq u$ and $\frac{2^r(\ell+1)}{w}=0(\text{mod}(v))$, $v \in Z^+$, $n=\alpha+2^{r+1}(\ell+1)$ and $\alpha=[(m(m+1)-(2^r(\ell+1)+w-1)(2^r(\ell+1)+w))/2]$.

Clearly, the priority list $L=(J_1,J_2,\ldots,J_n)$ is a STF_{MIN} or STF_{MAX} ordering and also an abritrary ordering. The schedule resulting from the priority list L, when the P.D. algorithm is used, is given in Figure 6.4 whereas the optimal schedule for the same task system is shown in Figure 6.5. The ratio of the completion times of the schedules shown in Figures 6.4 and 6.5 is:

$$\frac{\omega_{PD}}{\omega_{OPT}} = \lambda\left[r + 2 + \frac{w}{2^r(\ell+1)} + \sum_{i=k}^{m} \frac{1}{(i-1)}\right] / (1+m\varepsilon)$$

or

$$\lim_{\varepsilon \to 0} \frac{\omega_{PD}}{\omega_{OPT}} = \lambda\left[r + 2 + \frac{w}{2^r(\ell+1)} + \sum_{i=k}^{m} \frac{1}{i-1}\right], \qquad (6.2.21)$$

$$
\begin{array}{c|cccccccccccccccccccccccccccccccccccc}
 & \overbrace{}^{m} & & & & \overbrace{}^{m-1} & & & \overbrace{}^{m-2} & & & & \overbrace{}^{2^r(\ell+1)+w} & & & & \overbrace{}^{m-(2^r(\ell+1)+w)} & & & \overbrace{}^{2^r(\ell+1)} & & & \overbrace{\phantom{2^{r-1}(\ell+1)}}^{2^{r-1}(\ell+1)} & & & \overbrace{}^{2^0(\ell+1)} & & & \overbrace{}^{\ell} & & \\
1 & \varepsilon & 2\varepsilon & \dots & m\varepsilon & \frac{\lambda}{m-1} & \dots & \frac{\lambda}{m-1} & \frac{\lambda}{m-2} & \dots & \frac{\lambda}{m-2} & \dots & \frac{\lambda}{2^r(\ell+1)+w} & \dots & \frac{\lambda}{2^r(\ell+1)+w} & \frac{w\lambda}{2^r(\ell+1)} & \dots & \frac{w\lambda}{2^r(\ell+1)} & \lambda & \dots & \lambda & \dots & \lambda & \dots & \lambda & \lambda & 1 & \dots & 1 & \lambda \\
2 & \infty & 2\varepsilon & \dots & m\varepsilon & \frac{\lambda}{m-1} & \dots & \frac{\lambda}{m-1} & \frac{\lambda}{m-2} & \dots & \frac{\lambda}{m-2} & \dots & \frac{\lambda}{2^r(\ell+1)+w} & \dots & \frac{\lambda}{2^r(\ell+1)+w} & \frac{w\lambda}{2^r(\ell+1)} & \dots & \frac{w\lambda}{2^r(\ell+1)} & \lambda & \dots & \lambda & \dots & \lambda & \dots & \lambda & \lambda & \lambda & \dots & 1 & \lambda \\
 & \infty & \infty & & m\varepsilon & & & & & & & & & & & & & & & & & & & & & & & & & \\
 & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & \vdots \\
\ell & & & & & & & & & & & & & & & & & & & & & & & & & & & & \\
\ell+1 & \infty & \infty & \dots & m\varepsilon & \frac{\lambda}{m-1} & \dots & \frac{\lambda}{m-1} & \frac{\lambda}{m-2} & \dots & \frac{\lambda}{m-2} & \dots & \frac{\lambda}{2^r(\ell+1)+w} & \dots & \frac{\lambda}{2^r(\ell+1)+w} & \frac{w\lambda}{2^r(\ell+1)} & \dots & \frac{w\lambda}{2^r(\ell+1)} & \lambda & \dots & \lambda & \dots & \lambda & \dots & \lambda & 1 & \lambda & \dots & \lambda & \infty \\
 & & & & & & & & & & & & & & & & & & & & & & & 1 & \dots & 1 & \infty & \infty & \dots & \infty & \infty \\
 & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & \vdots \\
2^{r-1}(\ell+1) & \infty & \infty & \dots & m\varepsilon & \frac{\lambda}{m-1} & \dots & \frac{\lambda}{m-1} & \frac{\lambda}{m-2} & \dots & \frac{\lambda}{m-2} & \dots & \frac{\lambda}{2^r(\ell+1)+w} & \dots & \frac{\lambda}{2^r(\ell+1)+w} & \frac{w\lambda}{2^r(\ell+1)} & \dots & \frac{w\lambda}{2^r(\ell+1)} & \lambda & \dots & \lambda & \dots & \infty & \dots & \infty & & & \dots & & \infty \\
 & & & & & & & & & & & & & & & & & & & & & & & 1 & \dots & 1 & & & & & \\
 & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & \vdots \\
 & & & & & & & & & & & & & & & & & & & & & & & & & & & & \\
2^r(\ell+1) & \infty & \infty & \dots & m\varepsilon & \frac{\lambda}{m-1} & \dots & \frac{\lambda}{m-1} & \frac{\lambda}{m-2} & \dots & \frac{\lambda}{m-2} & \dots & \frac{\lambda}{2^r(\ell+1)+w} & \dots & \frac{\lambda}{2^r(\ell+1)+w} & \frac{w\lambda}{2^r(\ell+1)} & \dots & \frac{w\lambda}{2^r(\ell+1)} & 1 & \dots & 1 & & & & & & & & & \\
 & & & & & & & & & & & & & & & \frac{w}{2^r(\ell+1)} & \dots & \frac{w}{2^r(\ell+1)} & \infty & \dots & \infty & \infty & \cdot & \cdot & \cdot & \infty & & & & \\
 & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & \vdots \\
2^r(\ell+1)+w & & & & & & & & & & & & \frac{\lambda}{2^r(\ell+1)+w} & \dots & \frac{\lambda}{2^r(\ell+1)+w} & \frac{w}{2^r(\ell+1)} & & \frac{w}{2^r(\ell+1)} & & & & & & & & & & & \\
 & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & \vdots \\
2^r(\ell+1)+w+1 & \infty & \infty & \dots & m\varepsilon & \frac{\lambda}{m-1} & \dots & \frac{\lambda}{m-1} & \frac{\lambda}{m-2} & \dots & \frac{\lambda}{m-2} & \dots & \frac{1}{2^r(\ell+1)+w} & \dots & \frac{1}{2^r(\ell+1)+w} & \infty & \dots & \infty & \infty & \dots & \infty & \cdot & \cdot & \cdot & \infty & & & & \\
 & \vdots & \vdots & & \vdots & \vdots & & \vdots & \vdots & & \vdots & & \vdots & & \vdots & \infty & \dots & \infty & \vdots & & \vdots & & \vdots & & \vdots & \vdots & & \vdots & \vdots \\
 & & & & & & & & \frac{\lambda}{m-2} & \dots & \frac{\lambda}{m-2} & & & & & & & & & & & & & & & & & & & \\
m-1 & \infty & \infty & \dots & m\varepsilon & \frac{\lambda}{m-1} & \dots & \frac{\lambda}{m-1} & \frac{1}{m-2} & \dots & \frac{1}{m-2} & & & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & & & & & & & & & & & & \\
m & \infty & \infty & \dots & m\varepsilon & \frac{1}{m-1} & \dots & \frac{1}{m-1} & \infty & \dots & \infty & \dots & \infty & \dots & \infty & \infty & \dots & \infty & & & & & & & & & & & & \infty
\end{array}
$$

$$
(J_1, J_2 \dots, J_m, J_{m+1}, \dots, J_{2m-1}, \quad \dots, J_{3m-3}, \quad \cdot \quad \cdot \quad \cdot, J_\alpha, \quad J_{\alpha+1}, \quad \dots \quad J_{\alpha+2^r(\ell+1)}, \quad \cdot \quad \cdot \quad \cdot, J_{\alpha+2^{r+1}(\ell+1)})
$$

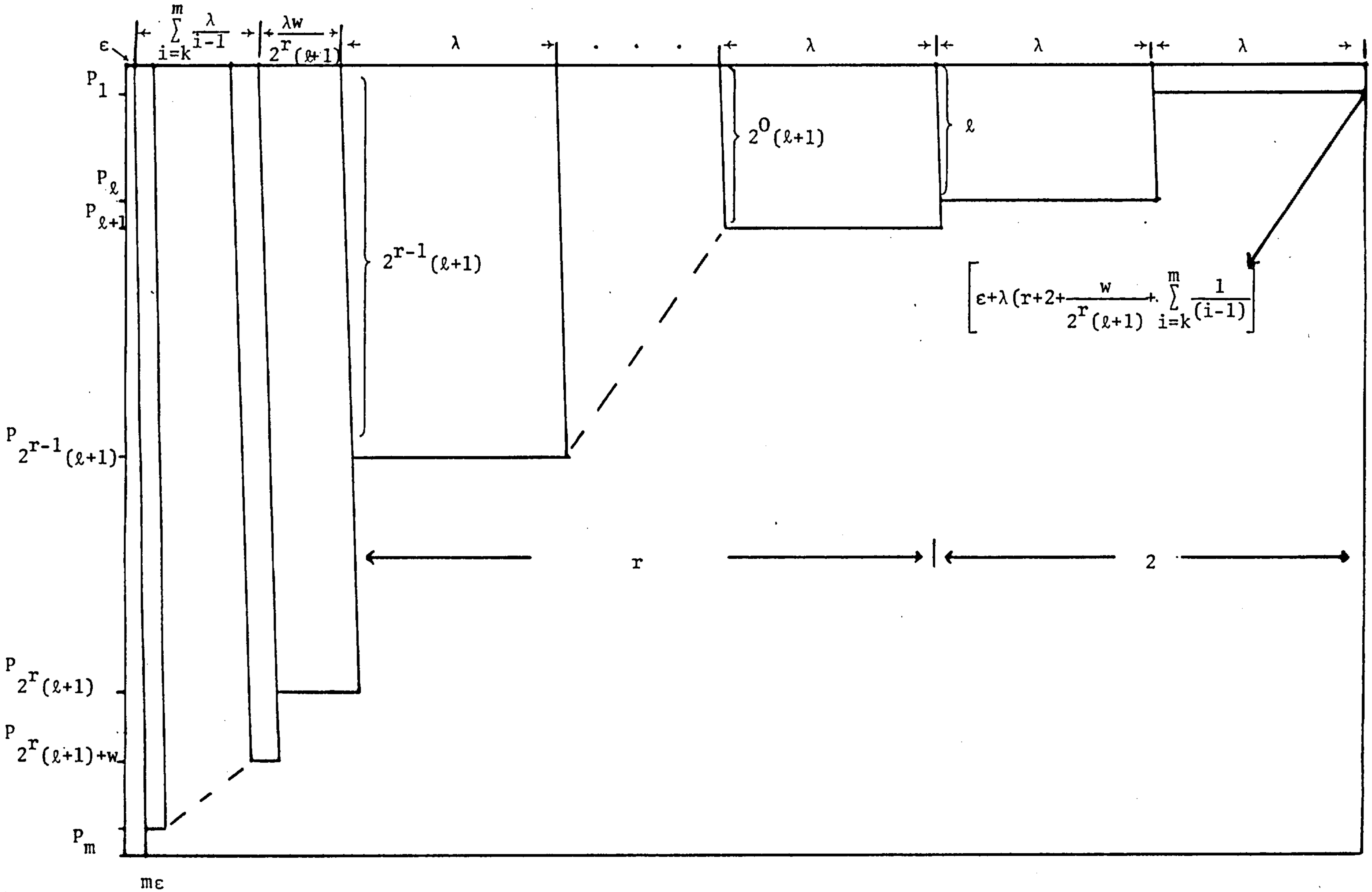


FIGURE 6.4: The schedule resulting from the priority list L given in Example 6.2.

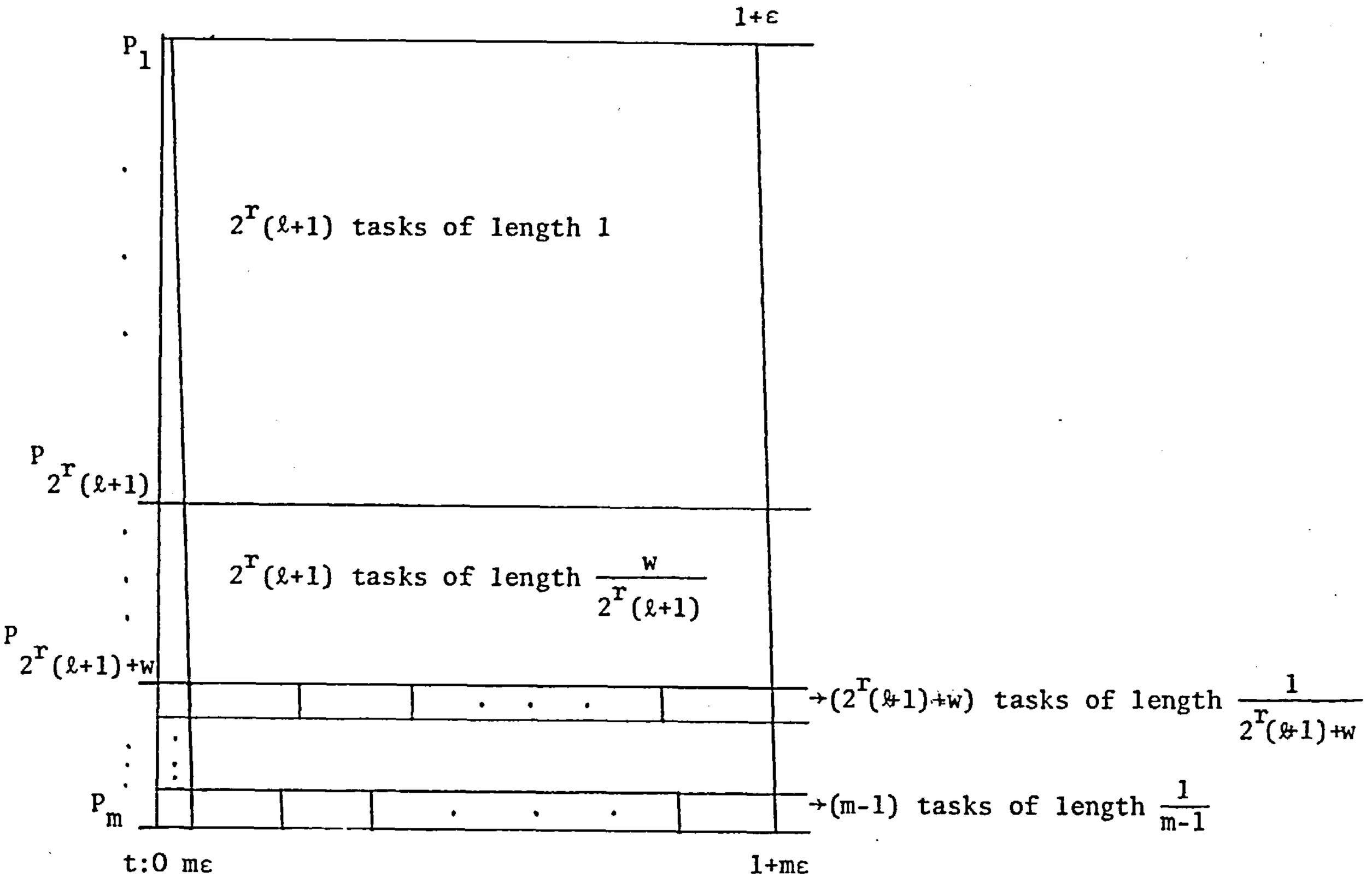


FIGURE 6.5: Optimal schedule for the task system given in Example 6.2.

where $k=2^r(\ell+1)+w+1$. The ratio (6.2.21) approaches the worst-case bound given by the theorem in part (i) for $2\leq\ell\leq m-1$ very closely. (i.e., let $m=7$ and $\ell=3$. Consequently, $r=0$, $u=3$ and $w=2$. Thus, from (6.2.21) we get $(\omega_{PD}/\omega_{OPT})=\lambda(2+\frac{2}{4}+\frac{1}{6})$ whereas from the proven bound $(\omega_{PD}/\omega_{OPT})\leq\lambda(2+\frac{3}{4})$. Also, if $m=9$ and $\ell=4$ and hence $r=0$, $u=5$ and $w=1$, we get from (6.2.21) the result $(\omega_{PD}/\omega_{OPT})=\lambda(2+\frac{1}{5}+\frac{1}{6}+\frac{1}{7}+\frac{1}{8})$ instead of $(\omega_{PD}/\omega_{OPT})\leq(2+\frac{4}{5})$ obtained from the worst-case bound.)

Also, there are two special cases when $u=0$ and $w=u>0$. However, rather than construct different but similar examples, it should be better to consider for each case a relaxed version of the general example as given above.

(1) If u=0 and hence $m=2^r(\ell+1)$, then consider as a new task system the first m and the last $2^r(\ell+1)$ jobs of the general task system given in the Example 6.2. It is easy to see then that for such a task system the ratio of the worst completion time over the optimal one is:

$$\lim_{\varepsilon \to 0} \frac{\omega_{PD}}{\omega_{OPT}} = \lambda(r+2),$$

which is the value of the worst-case performance predicted by the theorem in part (i) for $2 \leq \ell \leq m-1$, when u=0.

(2) If w=u>0 and hence, $w=m-2^r(\ell+1)$ then, consider as task system the one formed from the first m and the last $2^{r+1}(\ell+1)$ jobs of the task system given in the Example 6.2. Thus, with such a task system we can obtain

$$\lim_{\varepsilon \to 0} \frac{\omega_{PD}}{\omega_{OPT}} = \lambda \left[r + 2 + \frac{u}{2^r(\ell+1)} \right] ,$$

which is the extreme value of performance that the P.D. algorithm is allowed to deviate from the optimal performance according to the bound given in part (i) of the theorem for $2 \leq \ell \leq m-1$.

When $\ell=1$, the job which finishes at ω_{PD} should have at most an ω_{OPT} time requirement. Example 6.2, with the last job changed, can be used again to show that the worst-case performance bound proved for this case is a best possible one, when the priority list is in a STF_{MIN} ordering and also in a RAND ordering as well. However, the bound is not a best possible one when the priority list is in a STF_{MAX} ordering.

Furthermore, for the case where $\ell=m$, consider the following example in order to realise that the proven bound for this case is a best possible one.

Example 6.3: Let the task system $(J,[t_{ij}])$ be defined by the set of jobs $J=\{J_1,J_2,\ldots,J_n\}$ and the (m×n) matrix:

$$[t_{ij}] = \begin{bmatrix}
\epsilon & 2\epsilon \ldots m\epsilon & \overbrace{\frac{\lambda}{m} \; \frac{\lambda}{m} \ldots \frac{\lambda}{m} \; \frac{1}{m} \; \frac{1}{m}}^{m} & \overbrace{\ldots \; \frac{1}{m}}^{m-1} & \lambda \\
\infty & 2\epsilon \ldots m\epsilon & \frac{1}{m} \; \frac{\lambda}{m} \ldots \frac{\lambda}{m} \; \frac{\lambda}{m} \; \frac{1}{m} & \ldots \; \frac{1}{m} & \lambda \\
\infty & \infty \ldots m\epsilon & \infty \; \frac{1}{m} \ldots \frac{\lambda}{m} \; \frac{\lambda}{m} \; \frac{1}{m} & \ldots \; \frac{1}{m} & \lambda \\
\infty & \infty \ldots m\epsilon & \infty \; \infty & \ldots & \\
 & & \cdot \; \cdot \; \cdot & & \cdot \\
\cdot & \cdot \quad \cdot & \cdot \; \cdot \; \cdot & \ldots & \cdot \\
\cdot & \cdot \quad \cdot & \cdot \; \cdot \; \cdot & \ldots & \cdot \\
 & & \frac{\lambda}{m} \; \frac{\lambda}{m} \; \frac{1}{m} & \ldots & \lambda \\
 & & \frac{1}{m} \; \frac{\lambda}{m} \; \frac{1}{m} & \ldots & \lambda \\
\infty & \infty \quad m\epsilon \quad \infty \; \infty \ldots & \infty \; \infty \; \frac{\lambda}{m} & \ldots & 1
\end{bmatrix},$$

$$(J_1, J_2 \ldots J_m, J_{m+1}, \ldots, J_{2m}, \ldots, J_{m^2+1})$$

where $n=m^2+1$ and $\epsilon>0$ is a very small quantity.

The priority list $L=(J_1,J_2,\ldots,J_n)$ is in a STF_{MIN} or STF_{MAX} ordering and hence in a RAND ordering as well. The schedule which the P.D. algorithm constructs from such a priority list is shown in Fig.6.6 whereas the corresponding optimal schedule is given in Fig.6.7.

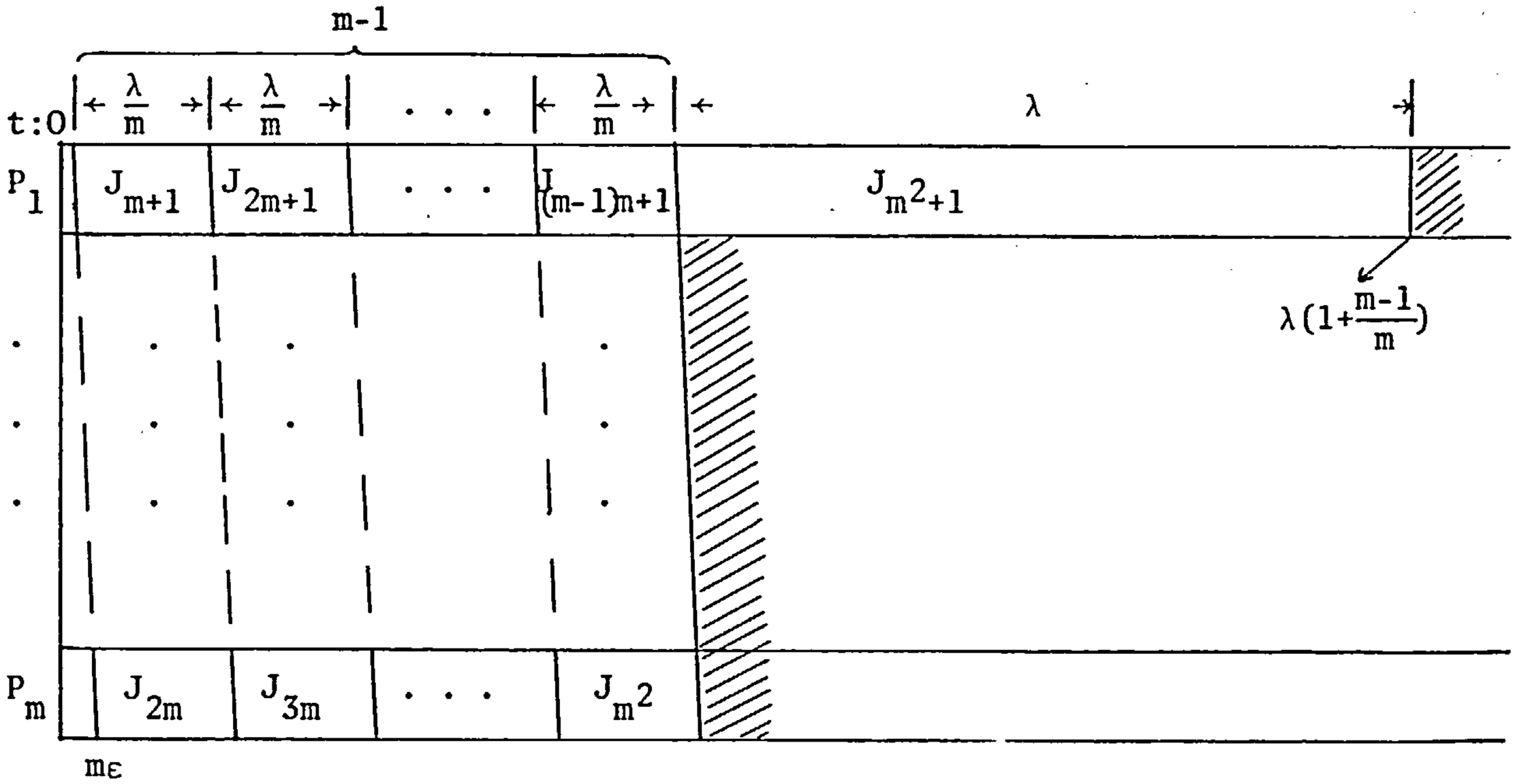


FIGURE 6.6: Schedule produced by the P.D. algorithm for the priority list L given in Example 6.3.

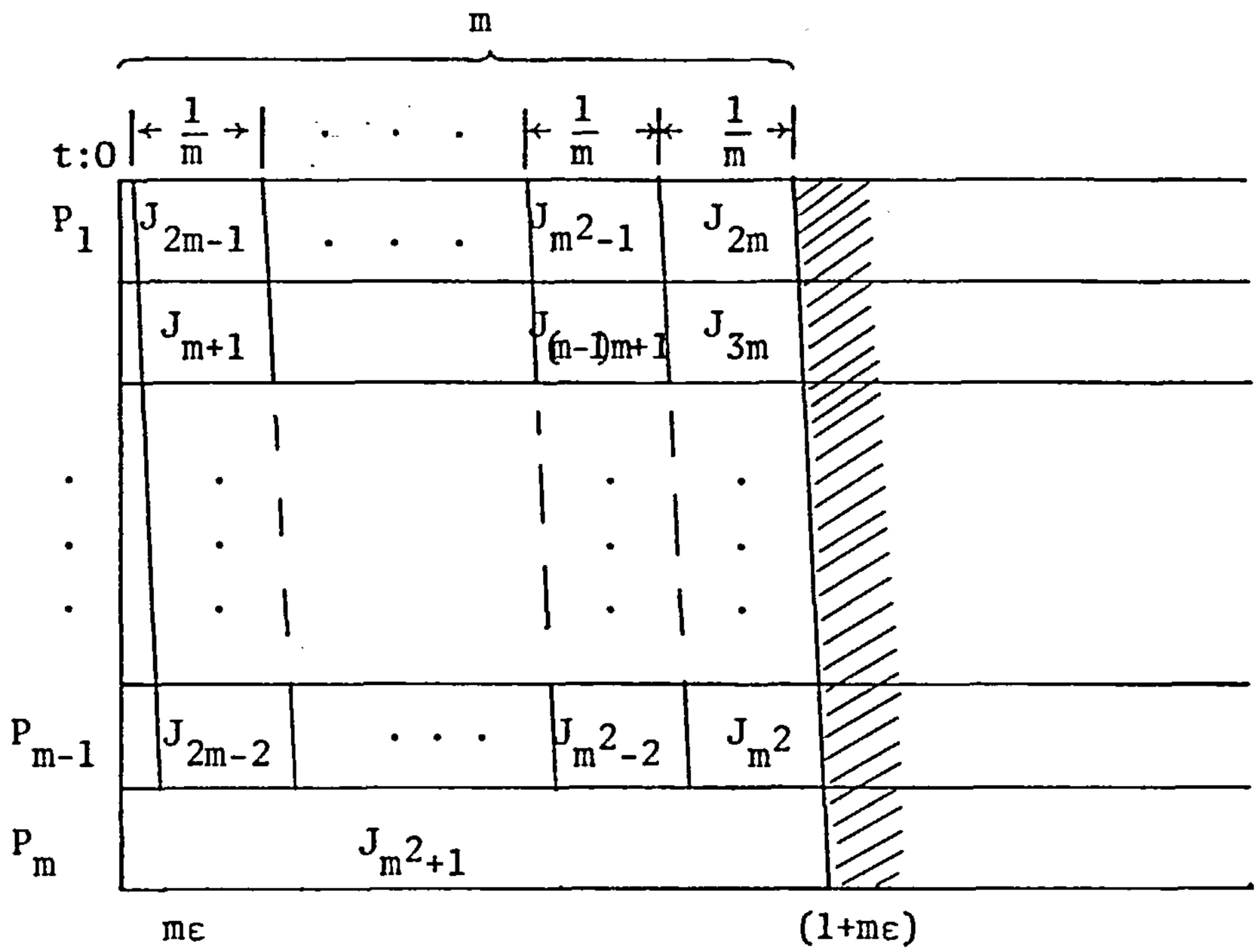


FIGURE 6.7: Optimal schedule for the task system given in Example 6.3.

The ratio of the completion times of these two schedules is:

$$\frac{\omega_{PD}}{\omega_{OPT}} = \frac{\lambda(1+\frac{m-1}{m})+\varepsilon}{1+m\varepsilon}$$

or

$$\lim_{\varepsilon \to 0} \frac{\omega_{PD}}{\omega_{OPT}} = \lambda(2 - \frac{1}{m}) \ ,$$

which is the bound predicted by the theorem in part (i) for $\ell=m$.

(ii) In the case of uniform processors, the general approach given in Theorem 6.2.1, to achieve the lower limit worst-case bound for arbitrary m and $1 \leq \ell \leq m-1$, can also be used here for the same purpose. However, the STF_{MIN} or STF_{MAX} ordering of the jobs in the priority list is guaranteed if $b_m \geq b_{m-1} \geq \ldots \geq b_1$ and $b_j \leq (1+(b_{j-1} / \sum_{i=1}^{j-2} b_i))b_1$, $3 \leq j \leq m$. Thus, eventually we can show

$$\max\left\{\frac{\omega_{PD}}{\omega_{OPT}}\right\} \geq 1 + \sum_{j=\ell+1}^{m} \frac{b_j}{\left(\sum_{i=1}^{j-1} b_i\right)} \ .$$

Now, if $\ell=m$, the bound obtained for the corresponding case in Theorem

6.2.1 can be used as the worst-case performance bound even when the priority list is in a STF_{MIN} or STF_{MAX} ordering. Moreover, the bound is a best possible one. This can be realised by considering the task system of the Example 6.1. For that task system, the priority list $L_1=(J_1,J_2,\ldots,J_{m^2+1})$ is a STF_{MIN} ordering, while the list $L_2=(J_1,J_2,\ldots,J_m,J_{m+1},(J_{m+2},J_{2m+1},J_{2m+2}),(J_{m+3},J_{3m+1},J_{2m+3},J_{3m+2},J_{3m+3}),\ldots,(J_{m+(m-1)},J_{(m-1)m+1},\ldots,J_{(m-2)m+(m-1)},J_{(m-1)m+(m-2)},J_{(m-1)m+(m-1)}),J_{m+m},J_{2m+m},\ldots,J_{(m-1)m+m},J_{m^2+1})$ is a STF_{MAX} ordering provided that $b_m \geq b_{m-1} \geq \ldots \geq b_1$, and $(b_{j+1} \geq \frac{b_j}{b_1} b_j)$, for $1 \leq j \leq m-1$ respectively. The schedule resulting from L_1 or L_2 using the P.D. algorithm is exactly the same as the corresponding one resulting from priority list in the Example 6.1 (i.e., Figure 6.2). Therefore, it can easily be seen that the ratio

$$\frac{\omega_{PD}}{\omega_{OPT}} = 1 + \frac{b_{max}}{b_{min}} - \frac{b_{max}}{\sum_{i=1}^{m} b_i} ,$$

which is the value of the worst-case bound given in part (ii) of the present theorem for $\ell=m$.

Theorem 6.2.2 indicates that the STF_{MIN} or STF_{MAX} ordering procedures can not offer any improvement in a worst-case sense over an arbitrary ordering for the P.D. algorithm, when the completion time is the performance criterion.

Theorem 6.2.3: Let the priority list be in a LMF, $LMST_{MIN}$ or $LMST_{MAX}$ ordering. Then,

for non-identical processors:

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \lambda_\ell^* (2 - \frac{1}{\ell}), \qquad 1 \leq \ell \leq m;$$

for uniform processors:

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 1 + \frac{b_{max}^{\ell}}{b_{min}^{\ell}} - \frac{b_{max}^{\ell}}{\sum_{i=1}^{\ell} b_i} , \quad 1 \leq \ell \leq m,$$

where ℓ is as defined in Theorem 6.2.1, $\lambda_\ell^* = \max_{J_j \in (\bigcup_{i=1}^{\ell} F_i)_j} \left\{\frac{\sigma_j}{\tau_j}\right\}$, $b_{max}^{\ell} = \max_{1 \leq i \leq \ell}\{b_i\}$

and $b_{min}^{\ell} = \min_{1 \leq i \leq \ell}\{b_i\}$.

Proof: (i) Let J_w, $1 \leq w \leq n$, be the largest task which finishes at ω_{PD}. Also, let ω'_{OPT} be the optimal completion time of the jobs in the truncated list $\{J_1, J_2, \ldots, J_w\}$ which can be scheduled on the first ℓ processors. However, if ω'_{PD} is the completion time of the schedule constructed by the P.D. algorithm while the jobs in the truncated list are in a LMF, $LMST_{MIN}$ or $LMST_{MAX}$ ordering, then we will have

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \frac{\omega'_{PD}}{\omega'_{OPT}} \quad , \tag{6.2.22}$$

since $\omega_{PD} = \omega'_{PD}$ and $\omega_{OPT} \geq \omega'_{OPT}$.

Now, ω'_{PD} can be expressed in a similar manner as ω_{PD} has been expressed in (6.2.16). In particular,

$$\omega'_{PD} = \frac{1}{\ell}\left[\frac{1}{\lambda_\ell^*}\left(\sum_{i=1}^{\ell} U_i\right) + \left(\frac{\lambda_\ell^* - 1}{\lambda_\ell^*}\right)\left(\sum_{i=1}^{\ell}(U_i + I_i)\right) + \frac{1}{\lambda_\ell^*}\left(\sum_{i=1}^{\ell} I_i\right)\right] \tag{6.2.23}$$

However,

$$\left.\begin{aligned} &\omega'_{PD} = U_i + I_i, \qquad 1 \leq i \leq \ell, \\ &\omega'_{OPT} \geq \frac{1}{\ell}\left(\sum_{j=1}^{W} \tau_j\right) \geq \frac{1}{\lambda_\ell^*}\left(\sum_{j=1}^{W} \sigma_j\right) \geq \frac{1}{\lambda_\ell^* \ell}\left(\sum_{i=1}^{\ell} U_i\right), \\ &I_i \leq \lambda_\ell^* \omega'_{OPT} \quad , \qquad 1 \leq i \leq \ell,\ i \neq x \end{aligned}\right\} \tag{6.2.24}$$

and $I_x = 0$,

where x is the processor on which the J_w is scheduled.

Because of the relationships (6.2.24), equation (6.2.23) becomes

$$\ell\omega'_{PD} \leq \frac{\ell\lambda_\ell^*\omega'_{OPT}}{\lambda_\ell^*} + \ell\omega'_{PD} - \frac{\ell}{\lambda_\ell^*}\omega'_{PD} + \frac{(\ell-1)}{\lambda_\ell^*}\lambda_\ell^*\,\omega'_{OPT}$$

or equivalently,

$$\frac{\omega'_{PD}}{\omega'_{OPT}} \leq \lambda_\ell^*\left(2 - \frac{1}{\ell}\right) \quad . \tag{6.2.25}$$

Finally from the inequalities (6.2.22) and (6.2.25) we obtain

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \lambda_{\ell}^{*}(2 - \frac{1}{\ell}) \ ,$$

which proves part (i) of the theorem. Note, that equality holds only if $\ell=1$. In such a case, we obtain the optimal schedule since $\lambda_1^*=1$.

Moreover, this bound is a best possible one and can be realised from the following example.

Example 6.4: Let the jobs $\{J_1,J_2,\ldots,J_w\}$ of the task system $(J,[t_{ij}])$, with at most $\ell_j=\ell$, $1\leq j\leq w$, be defined by the $(m\times w)$ matrix:

$$[t_{ij}] = \begin{bmatrix}
\varepsilon & \lambda_{\ell}^{*}(\frac{\ell-1}{\ell}) & 2\varepsilon & \lambda_{\ell}^{*}(\frac{\ell-1}{\ell}) & \cdots & (\ell-2)\varepsilon & \lambda_{\ell}^{*}(\frac{\ell-1}{\ell}) & (\frac{\ell-1}{\ell}) & \overbrace{\frac{1}{\ell}\cdots\frac{1}{\ell}}^{\ell-1} & \lambda_{\ell}^{*} \\
\varepsilon & (\frac{\ell-1}{\ell}) & 2\varepsilon & \lambda_{\ell}^{*}(\frac{\ell-1}{\ell}) & \cdots & (\ell-2)\varepsilon & \lambda_{\ell}^{*}(\frac{\ell-1}{\ell}) & (\frac{\ell-1}{\ell}) & \frac{1}{\ell}\cdots\frac{1}{\ell} & \lambda_{\ell}^{*} \\
\infty & \infty & 2\varepsilon & (\frac{\ell-1}{\ell}) & \cdots & (\ell-2)\varepsilon & \lambda_{\ell}^{*}(\frac{\ell-1}{\ell}) & (\frac{\ell-1}{\ell}) & \frac{1}{\ell}\cdots\frac{1}{\ell} & \lambda_{\ell}^{*} \\
\infty & \infty & \infty & \infty & \cdots & (\ell-2)\varepsilon & \lambda_{\ell}^{*}(\frac{\ell-1}{\ell}) & (\frac{\ell-1}{\ell}) & \frac{1}{\ell}\cdots\frac{1}{\ell} & \lambda_{\ell}^{*} \\
\vdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\infty & \infty & \infty & \infty & \cdots & (\ell-2)\varepsilon & \lambda_{\ell}^{*}(\frac{\ell-1}{\ell}) & (\frac{\ell-1}{\ell}) & \frac{1}{\ell}\cdots\frac{1}{\ell} & \lambda_{\ell}^{*} \\
\infty & \infty & \infty & \infty & \cdots & (\ell-2)\varepsilon & (\frac{\ell-1}{\ell}) & \lambda^{*}(\frac{\ell-1}{\ell}) & \frac{1}{\ell}\cdots\frac{1}{\ell} & \lambda_{\ell}^{*} \\
\infty & \infty & \infty & \infty & \cdots & \infty & \infty & \infty & \frac{\lambda_{\ell}^{*}}{\ell}\cdots\frac{\lambda_{\ell}^{*}}{\ell}+\varepsilon & 1
\end{bmatrix} ,$$

$$(J_1, \ J_2, \ J_3, \ J_4, \ \ldots, J_{2\ell-5}, \ \ldots, J_{2\ell-2}, \ldots, J_{3\ell-3})$$

where $w=3\ell-3$ and $\varepsilon>0$ is a very small quantity.

The priority list $L=(J_1,J_2,\ldots,J_w)$ is in a LMF, $LMST_{MIN}$ or $LMST_{MAX}$ ordering. Now, the schedule constructed by the P.D. algorithm and the corresponding optimal one are given in Fig.6.8 and Fig.6.9 respectively.

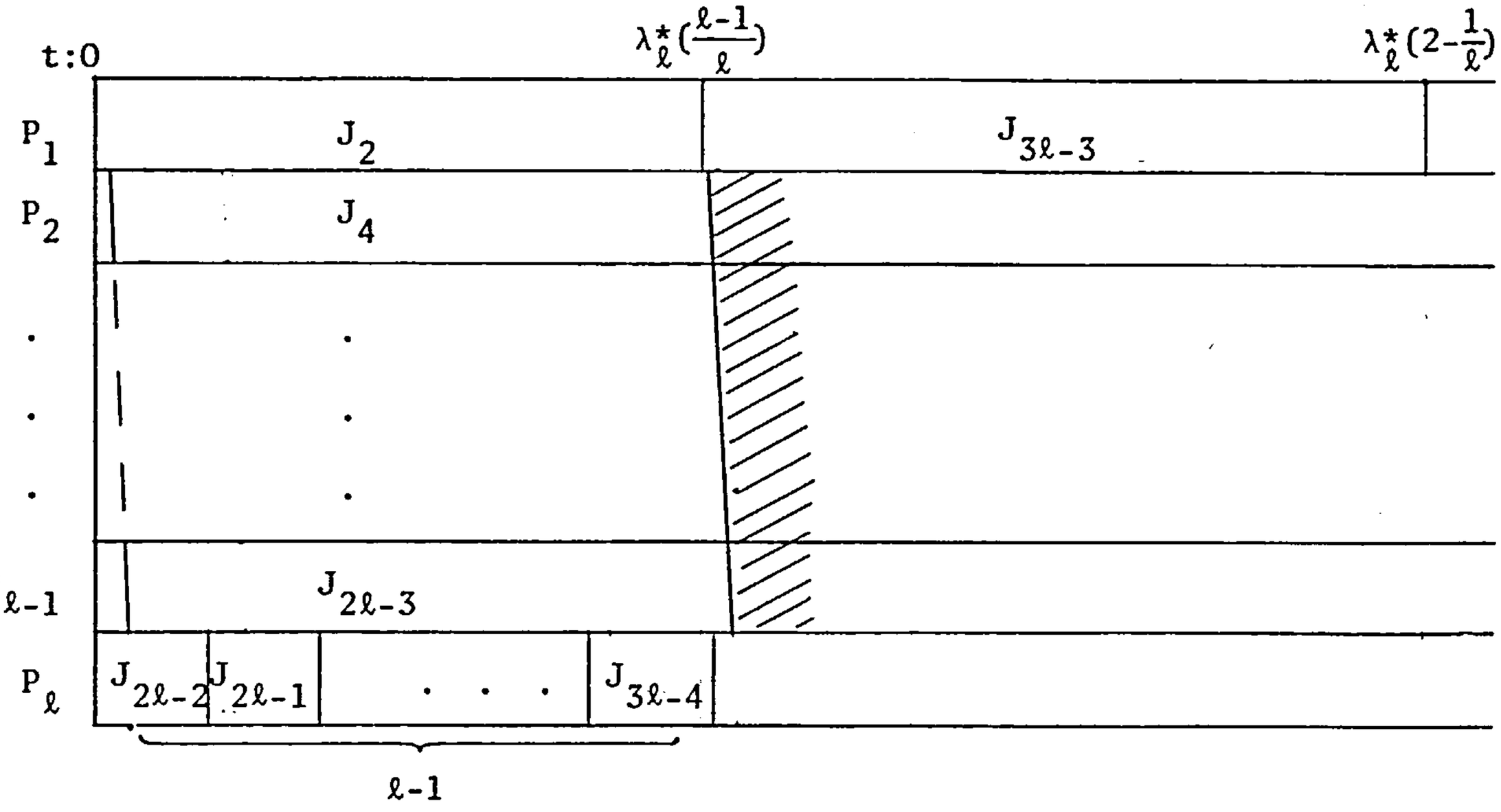


FIGURE 6.8: Schedule produced by the P.D. algorithm for the priority list given in Example 6.4.

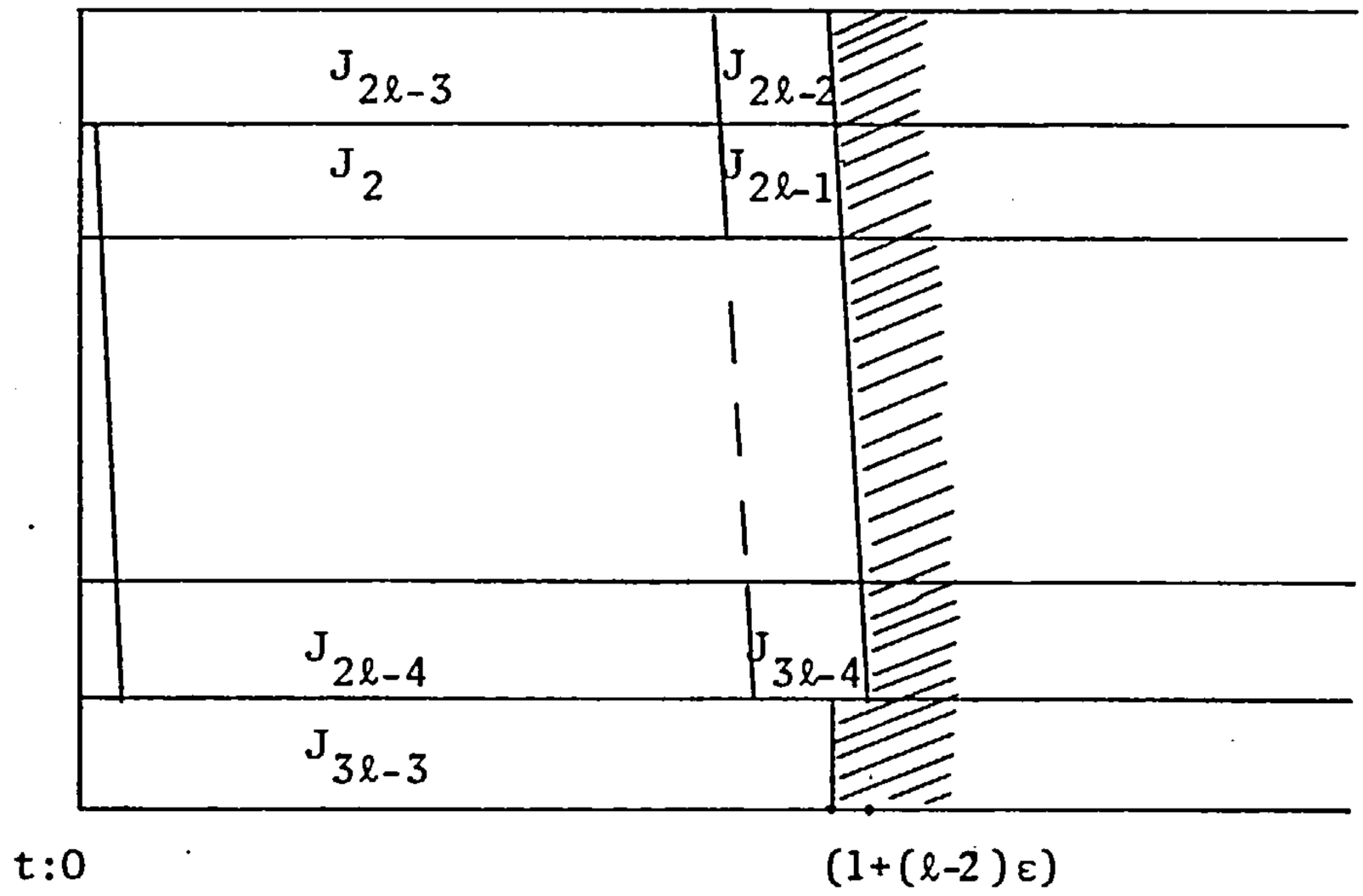


FIGURE 6.9: Optimal schedule for the truncated task system given in Example 6.4.

The ratio of the completion times of these two schedules, as $\varepsilon \to 0$, is:

$$\frac{\omega_{PD}}{\omega_{OPT}} = \lambda_{\ell}^{*}\left(2 - \frac{1}{\ell}\right) ,$$

which is exactly the value of the bound given in part (i) of the theorem.

(ii) Following a similar analysis as given in part (i) of the theorem and having in mind that

$$\omega'_{PD} = \frac{1}{\ell}\left[\frac{1}{b_{max}^{\ell}}\left(\sum_{i=1}^{\ell}(U_i b_i)\right) + \sum_{i=1}^{\ell}\left[\left(\frac{b_{max}^{\ell} - b_i}{b_{max}^{\ell}}\right)\left(U_i + I_i\right)\right] + \frac{1}{b_{max}^{\ell}}\left(\sum_{i=1}^{\ell}(I_i b_i)\right)\right]$$

as well as

$$\omega'_{PD} = U_i + I_i, \qquad 1 \leq i \leq \ell,$$

$$\omega'_{OPT} \geq \left(\sum_{i=1}^{\ell}(U_i b_i)\right) \Big/ \left(\sum_{i=1}^{\ell} b_i\right) ,$$

$$I_i \leq \frac{b_{max}^{\ell}}{b_{min}^{\ell}}\,\omega'_{OPT} , \quad i=1(1)\ell, \quad i \neq x$$

and $$I_x = 0,$$

where x is the processor on which J_w is scheduled, we obtain:

$$\ell\omega'_{PD} \leq \frac{\omega'_{OPT}}{b_{max}^{\ell}}\left(\sum_{i=1}^{\ell} b_i\right) + \ell\omega'_{PD} - \frac{\omega'_{PD}}{b_{max}^{\ell}}\left(\sum_{i=1}^{\ell} b_i\right) + \sum_{\substack{i=1 \\ i \neq x}}^{\ell}\left(\frac{b_i}{b_{max}^{\ell}} \cdot \frac{b_{max}^{\ell}}{b_{min}^{\ell}}\omega'_{OPT}\right)$$

or

$$\frac{\omega'_{PD}}{\omega'_{OPT}} \leq 1 + \frac{b_{max}^{\ell}}{b_{min}^{\ell}} - \frac{b_{max}^{\ell}}{\sum_{i=1}^{\ell} b_i} , \qquad 1 \leq \ell \leq m$$

and finally, because of the inequality (6.2.22),

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 1 + \frac{b_{max}^{\ell}}{b_{min}^{\ell}} - \frac{b_{max}^{\ell}}{\sum_{i=1}^{\ell} b_i} \qquad 1 \leq \ell \leq m,$$

which completes the proof of the theorem.

Furthermore, the following example shows that the value of the bound can be attained and hence the bound is a best possible one.

Example 6.5: Let the jobs $\{J_1,J_2,\ldots,J_w\}$ of the task system $(\mathcal{J},\{m_j\},\{T_j\})$, with at most $\ell_j=\ell$, $1\leq j\leq w$, be defined by:

$$J_1 : \left(|P_1|, \left\{\frac{\sum_{i=1}^{\ell-1} b_i}{\sum_{i=1}^{\ell} b_i}\ b_1 - \varepsilon\right\}\right)$$

$$J_j : \left(|P_j|, \left\{\frac{\sum_{i=1}^{\ell-1} b_i}{\sum_{i=1}^{\ell} b_i}\ b_j\right\}\right), \qquad j=2(1)(\ell-1)$$

$$J_j : \left(|P_\ell|, \left\{\frac{b_{max}^{\ell}}{\sum_{i=1}^{\ell} b_i}\ b_i\right\}\right), \qquad j=\ell(1)(2\ell-2),\ i=1(1)(\ell-1)$$

and $$J_{2\ell-1}: \left(|P_\ell|, \{b_{max}^{\ell}\}\right),$$

where $w=2\ell-1$ and $b_1\leq b_2\leq\ldots\leq b_\ell$.

Clearly, the priority list $L=(J_1,J_2,\ldots,J_w)$ is a LMF, $LMST_{MIN}$ or $LMST_{MAX}$. The schedule resulting from this priority list using the P.D. algorithm is shown in Fig.6.10, whereas the corresponding optimal schedule is given in Fig.6.11.

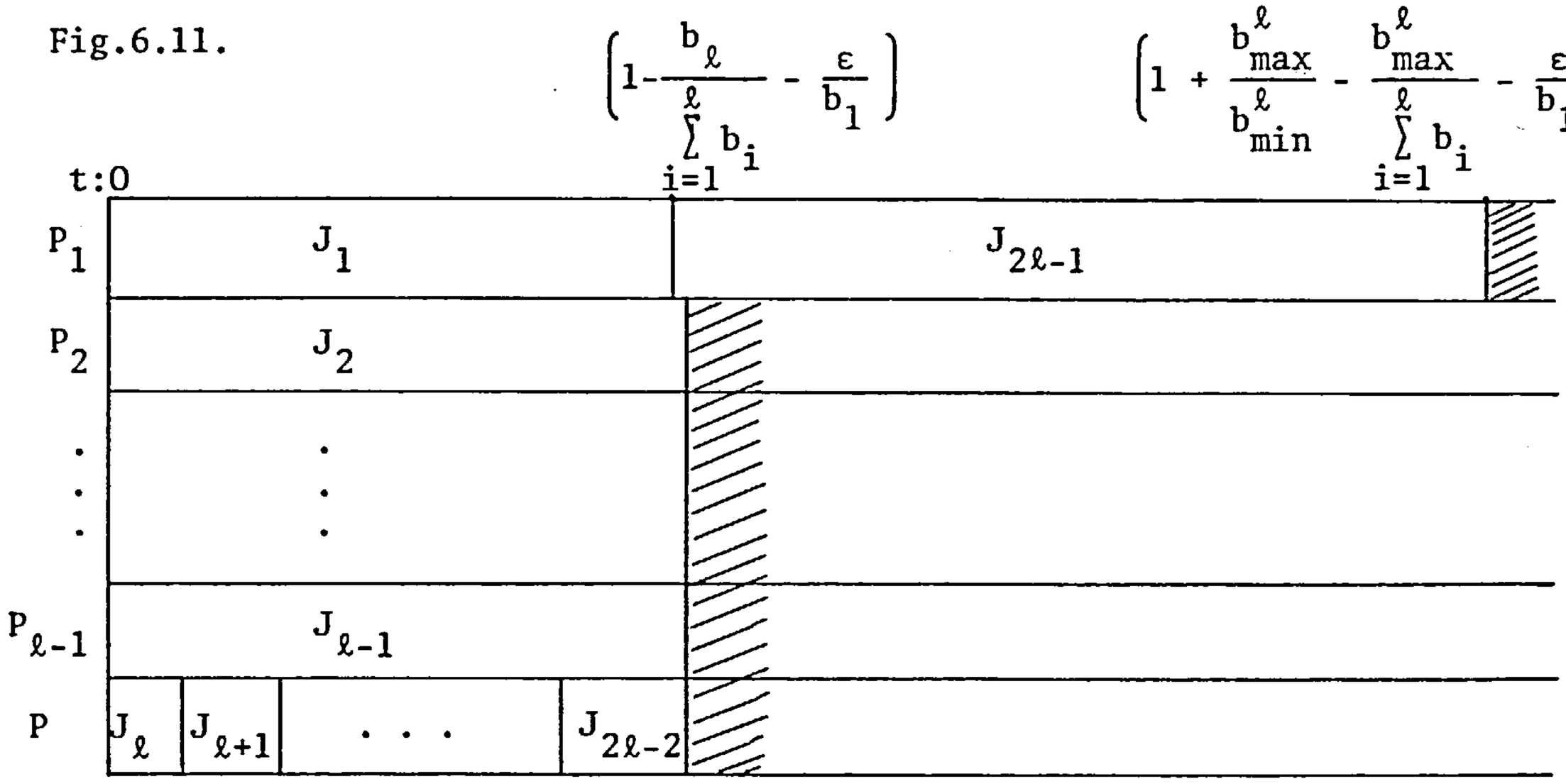


FIGURE 6.10: Schedule produced by the P.D. algorithm for the priority list given in Example 6.5.

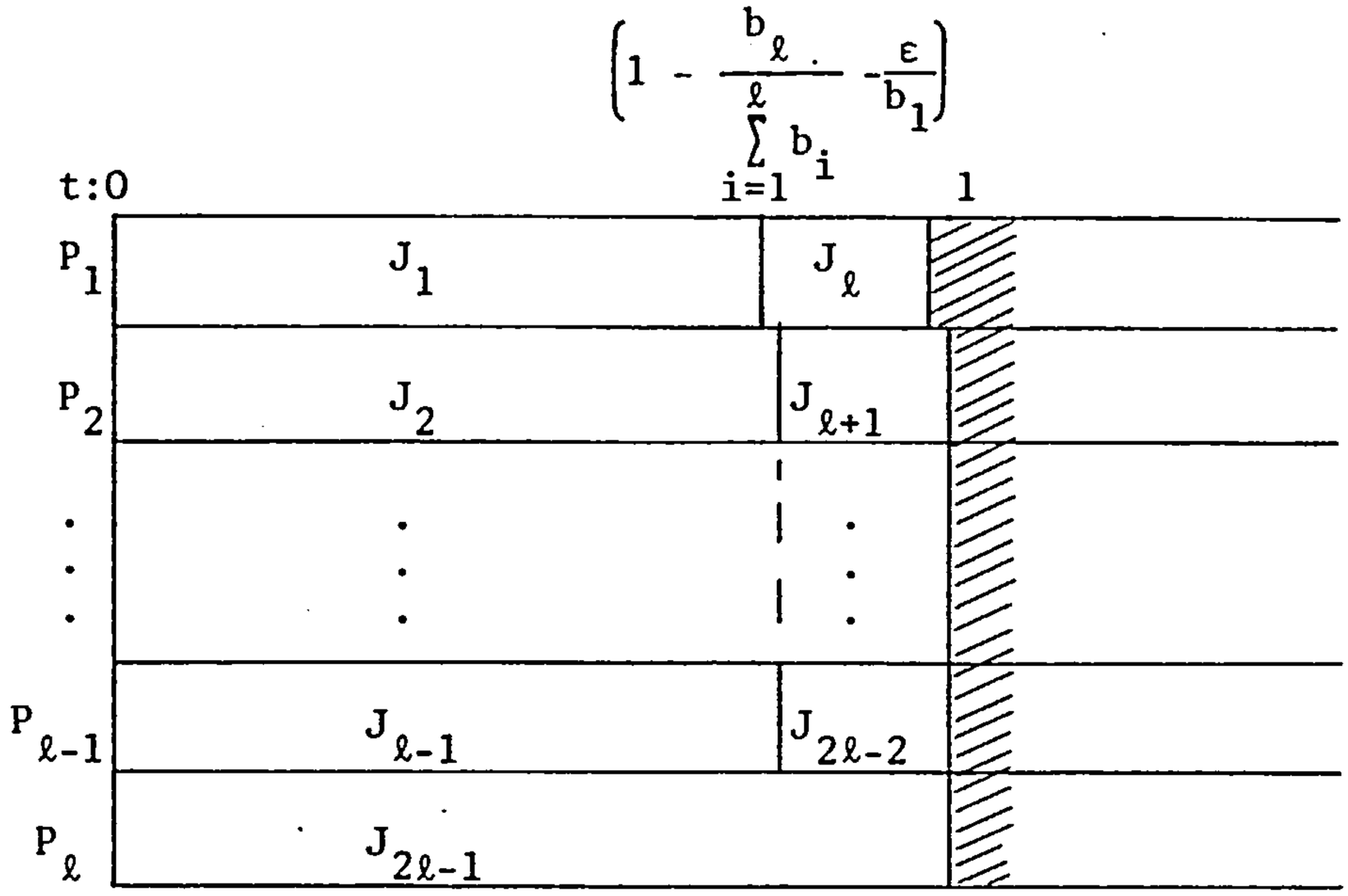


FIGURE 6.11: Optimal schedule for the truncated task system given in Example 6.5.

The ratio of the completion times of the above schedules as $\epsilon \to 0$, is:

$$\frac{\omega'_{PD}}{\omega'_{OPT}} = 1 + \frac{b^{\ell}_{max}}{b^{\ell}_{min}} - \frac{b^{\ell}_{max}}{\sum_{i=1}^{\ell} b_i} ,$$

which is the value of the proven bound in part (ii) of the theorem□

From the given bounds in Theorem 6.2.3 one can easily realise that the extreme performance of the P.D. algorithm when the priority list is in a $LMST_{MIN}$, $LMST_{MAX}$ and hence a LMF ordering deviates from the optimal performance as the value of ℓ increases. In addition, when ℓ increases the values of λ^*_ℓ or $(b^{\ell}_{max}/b^{\ell}_{min})$ may increase as well. However, although the worst-case bound is always much better than the corresponding one of an arbitrary ordering, for $\ell=m$ it gets exactly the same value. This also means that, apart from the case where $\ell=m$, the utilisation of memory in contributing to the STF ordering rule does offer an improvement in the worst-case performance of the P.D. algorithm. Finally, if λ relaxes to 1

or b_i to b_{i+1}, $1 \leq i \leq m-1$, then the bounds of the last theorem agree with the corresponding one found for the homogeneous multiprocessor model with independent memories. (See Theorem III.2, Appendix III.)

Theorem 6.2.4: Let the priority list be in a $LMLT_{MIN}$ or $LMLT_{MAX}$ ordering. Then,

(i) for non-identical processors:

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \lambda_\ell^* \left(1 + \frac{1}{k'+1} - \frac{1}{(k'+1)}\right) \quad \text{, for } \ell=1,2 \text{ and}$$

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \lambda_\ell^* \times \max \left\{\left(1 + \frac{1}{k'} - \frac{1}{(\ell-1)k'}\right) , \left(1 + \frac{1}{k'+1} - \frac{1}{\ell(k'+1)}\right)\right\}, \text{ for } 3 \leq \ell \leq m$$

(ii) for uniform processors:

$$\frac{\omega_{PD}}{\omega_{OPT}} = 1 \text{ for } \ell=1 \text{ and}$$

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 1 + \max \left\{\left[\frac{1}{k'} \left(\frac{b_{max}^\ell}{b_{min}^\ell} - \frac{b_{max}^\ell}{\sum_{i=1}^{\ell} b_i}\right)\right] , \left[\frac{1}{k'+1} \left(\frac{b_{max}^\ell}{b_{min}^\ell} - \frac{b_{max}^\ell}{\sum_{i=1}^{\ell} b_i}\right)\right]\right\} \text{ for } 2 \leq \ell \leq m$$

$i \neq x$

where ℓ and λ_ℓ^* b_{max}^ℓ, b_{min}^ℓ are as defined in Theorems 6.2.1 and 6.2.3, x is the index of the processor with the lowest speed and k' is the maximum number of jobs with $\ell_j=\ell$ scheduled on a particular processor before J_w has begun its execution.

Proof: (i) As in the previous theorem

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \frac{\omega'_{PD}}{\omega'_{OPT}} \quad ,$$

where ω'_{PD} and ω'_{OPT} are the completion times of the schedule constructed by the P.D. algorithm and the corresponding optimal process respectively, for the truncated list of jobs $\{J_1, J_2, \ldots, J_w\}$. Therefore, the analysis needs only to be concerned with the truncated list and the first ℓ processors.

Let x and z be the indices of the processors where the schedule

finishes and k' occurs, respectively. Then, according to the values of x and z, we can distinguish two cases when $x<z$ and $x\geq z$.

Case 1: If $x<z$ then, $\omega'_{PD}, \omega'_{OPT}$ and I_i are bounded by:

$$\left.\begin{aligned} \omega'_{PD} &= U_i + I_i \quad , && 1\leq i\leq \ell \ , \\ \omega'_{OPT} &\leq \left(\sum_{i=1}^{\ell} U_i\right)\Big/\left(\ell\lambda^*_\ell\right) \quad , \\ I_i &\leq \lambda^*_\ell \omega'_{OPT}/k' \ ^{\dagger} \quad , && 1\leq i\leq \ell, \quad i\neq x\neq z, \\ I_z &\leq \omega'_{OPT} - \lambda^*_\ell \omega'_{OPT} \ ^{\dagger\dagger} \end{aligned}\right\} \qquad (6.2.26)$$

and $I_x = 0.$

ω'_{PD} is expressed in the equation (6.2.23) as

$$\omega'_{PD} = \frac{1}{\ell}\left[\frac{1}{\lambda^*_\ell}\left(\sum_{i=1}^{\ell} U_i\right) + \frac{\lambda^*_\ell - 1}{\lambda^*_\ell}\left(\sum_{i=1}^{\ell}(U_i+I_i)\right) + \frac{1}{\lambda^*_\ell}\left(\sum_{i=1}^{\ell} I_i\right)\right] .$$

Then, following a similar analysis as in Theorem 6.2.3 and having in mind the relationships (6.2.26) and (6.2.22) we can obtain

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \lambda^*_\ell \left(1 + \frac{1}{k'} - \frac{1}{k'(\ell-1)}\right) .$$

Case 2: If $x\geq z$ then, only the idle times of the processors are bounded differently and hence should be replaced in the set of relations (6.2.26).

$^{\dagger} I_i \leq \lambda^*_\ell \omega'_{OPT}/k'$, *because* $\sigma_w \leq \lambda^*_\ell \omega'_{OPT}/k'$ *(since no parts of a job can be executed on two or more processors simultaneously and if* $\sigma_w > \lambda^*_\ell \omega'_{OPT}/k'$ *and hence* $\tau_w > \omega'_{OPT}/k'$ *then, because no schedule could have completion time less than* $k'\tau_w$ *subject to the conditions that:* $\tau_j \geq \tau_w$ *or* $\sigma_j \geq \sigma_w$ *for every job with* $\ell_j = \ell$ *and* $s_w > (k'-1)\tau_w$, $\omega'_{OPT} \geq k'\tau_w \geq k'\sigma_w/\lambda^*_\ell > \omega'_{OPT}$ *which is a contradiction) and* $I_i \leq \sigma_w$ *(since* $U_i \geq s_w$ *for* $1\leq i\leq \ell$ *and* $i\neq x\neq z$).

$^{\dagger\dagger} I_z \leq \omega'_{PD} - \lambda^* \omega'_{OPT}$, *because* $I_z = \omega'_{PD} - U_z$ *and* $U_z \geq \lambda^*_\ell \omega'_{OPT}$.

In particular, †

$$I_i \lesssim \lambda^*_\ell \frac{\omega'_{OPT}}{k'+1} \quad , \; 1 \lesssim i \lesssim \ell \quad \text{and} \quad i \neq x.$$

Therefore, from equation (6.2.23) using the new set of relations and (6.2.22) we get

$$\frac{\omega_{PD}}{\omega_{OPT}} \lesssim \lambda^*_\ell \left(1 + \frac{1}{k'+1} - \frac{1}{(k'+1)\ell}\right) .$$

Now, combining the proven worst-case bounds for Case 1 and 2 we obtain

$$\frac{\omega_{PD}}{\omega_{OPT}} \lesssim \lambda^*_\ell \times \max\left\{\left(1 + \frac{1}{k'} - \frac{1}{k'(\ell-1)}\right) , \left(1 + \frac{1}{k'+1} - \frac{1}{(k'+1)\ell}\right)\right\} .$$

This bound characterises the extreme performance of the algorithm when $3 \lesssim \ell \lesssim m$, since for $\ell=1$ Case 2 always occurs and for $\ell=2$ the second factor of the above bound is worst than the first for any value of k'. So,

$$\frac{\omega_{PD}}{\omega_{OPT}} \lesssim \lambda^*_\ell \left(1 + \frac{1}{k'+1} - \frac{1}{(k'+1)\ell}\right) , \quad \text{for} \quad \ell=1,2.$$

(ii) For the case of uniform processors ω'_{PD} can be expressed as in equation (6.2.19), i.e.,

$$\omega'_{PD} = \frac{1}{\ell}\left[\frac{1}{b^\ell_{max}}\left(\sum_{i=1}^{\ell}(U_i b_i)\right) + \sum_{i=1}^{\ell}\left[\left(\frac{b^\ell_{max}-b_i}{b^\ell_{max}}\right)(U_i+I_i)\right] + \frac{1}{b^\ell_{max}}\left(\sum_{i=1}^{\ell}(I_i b_i)\right)\right]$$

Then, for Case 1 we have

---

†$I_i \lesssim \lambda^*_\ell \omega'_{OPT}/(k'+1)$, *because* $\sigma_w \lesssim \lambda^*_\ell \omega'_{OPT}/(k'+1)$ *(since no parts of the same job can be run simultaneously on two or more processors and if* $\sigma_w \gtrsim \lambda^*_\ell \omega'_{OPT}/k'$ *then, because* $\sigma_j \gtrsim \sigma_w$ *or* $\tau_j \gtrsim \tau_w$ *for every job with* $\ell_j=\ell$, $\omega'_{OPT} \gtrsim s_w + (1/\ell)\tau_w \gtrsim s_w + (1/\lambda^*_\ell \times \ell)\sigma_w$ *and* $s_w \gtrsim k'\tau_w \gtrsim k'\sigma_w/\lambda^*_\ell$, $\omega'_{OPT} \gtrsim \omega_{OPT} + \omega'_{OPT}/(\ell k')$ *which is a contradiction) and* $I_i \lesssim \sigma_w$ *(since* $U_i \gtrsim s_w$ *for* $1 \lesssim i \lesssim \ell$ *and* $i \neq x$).

$$\left.\begin{aligned}
&\omega'_{PD} = U_i + I_i, \qquad 1\leq i\leq \ell \ , \\
&\omega'_{OPT} \geq \left(\sum_{i=1}^{\ell} (U_i b_i)\right) \Big/ \left(\sum_{i=1}^{\ell} b_i\right) \ , \\
&I_i \leq (\omega'_{OPT}/k')(b^{\ell}_{max}/b^{\ell}_{min}), \quad i=1(1)\ell \ , \quad i\neq x,z, \\
&I_z \leq \omega'_{PD} - \omega'_{OPT} \\
\text{and} \qquad &I_x = 0,
\end{aligned}\right\} \qquad (6.2.27)$$

On the other hand, when Case 2 occurs in the above set of relationships only the idle times of the processors are bounded differently. In particular,

$$I_i \leq (\omega'_{OPT}/(k'+1))(b^{\ell}_{max}/b^{\ell}_{min}), \qquad i=1(1)\ell, \ i\neq x.$$

Now, bearing in mind the equation (6.2.18) and the relations (6.2.27) for Case 1 we obtain

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 1 + \frac{1}{k'}\left(\frac{b^{\ell}_{max}}{b^{\ell}_{min}} - \frac{b^{\ell}_{max}}{\sum_{\substack{i=1 \\ i\neq x}}^{\ell} b_i}\right) \ ,$$

whereas for Case 2

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 1 + \frac{1}{k'+1}\left(\frac{b^{\ell}_{max}}{b^{\ell}_{min}} - \frac{b^{\ell}_{max}}{\sum_{\substack{i=1 \\ i\neq x}}^{\ell} b_i}\right) \ .$$

Then, combining these two bounds we have

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 1 + \max\left\{\left[\frac{1}{k'}\left(\frac{b^{\ell}_{max}}{b^{\ell}_{min}} - \frac{b^{\ell}_{max}}{\sum_{\substack{i=1 \\ i\neq x}}^{\ell} b_i}\right)\right]\right\}, \left[\frac{1}{k'+1}\left(\frac{b^{\ell}_{max}}{b^{\ell}_{min}} - \frac{b^{\ell}_{max}}{\sum_{\substack{i=1 \\ i\neq x}}^{\ell} b_i}\right)\right]\right\} \ , \text{for } 2\leq\ell\leq m.$$

Finally, since for $\ell=1$ Case 2 always takes place,

$$\frac{\omega_{PD}}{\omega_{OPT}} = 1 \qquad \text{for } \ell=1.$$

Moreover, for Case 1 the following example shows that the corresponding bound is a best possible one.

Example 6.6: Let the jobs $\{J_1,J_2,\ldots,J_w\}$ of the task system $(J,\{m_j\},\{T_j\})$, with at most $\ell_j=\ell$, $1\leq j\leq w$, be defined by:

$$J_1 \quad : \left(|P_1|,\ \left\{\left(1-\frac{b_\ell}{k'\left(\sum_{i=1}^{\ell-1} b_i\right)}\right)\ b_1-\varepsilon\right\}\right)$$

$$J_j \quad : \left(|P_j|,\ \left\{\left(1-\frac{b_\ell}{k'\left(\sum_{i=1}^{\ell-1} b_i\right)}\right)\ b_j\right\}\right) \quad , \quad j=2(1)\ell-2$$

$$J_{\ell-1} : \left(|P_{\ell-1}|,\ \left\{\left(1-\frac{1}{k'}\right)b_{\ell-1}\right\}\right)$$

$$J_j \quad : \left(|P_{\ell-1}|,\ \left\{\frac{b_i}{k'\left(\sum_{i=1}^{\ell-1} b_i\right)}\ b_{\ell-1}\right\}\right) \quad , \quad j=\ell(1)2\ell-3,\ i=\ell-2(-1)1.$$

$$J_j \quad : \left(|P_\ell|\ ,\ \left\{\frac{1}{k'}\times b_\ell\right\}\right) \quad ; \quad j=(2\ell-2)(1)(2\ell-2+k')$$

and $J_{2\ell-1+k'}$: $\left(|P_\ell|\ ,\ \left\{\frac{1}{k'}\times b_{\ell-1}\right\}\right)$,

where $w=2\ell-1+k'$ and $b_1\leq b_2\leq\ldots\leq b_{\ell-1}=b_\ell$.

It is easy to realise that the priority list $L=(J_1,J_2,\ldots,J_w)$ is a $LMLT_{MIN}$ and therefore, a $LMLT_{MAX}$ ordering as well. However, the schedule resulting from the P.D. algorithm is shown in Fig.6.12, whereas the corresponding optimal schedule is given in Fig. 6.13.

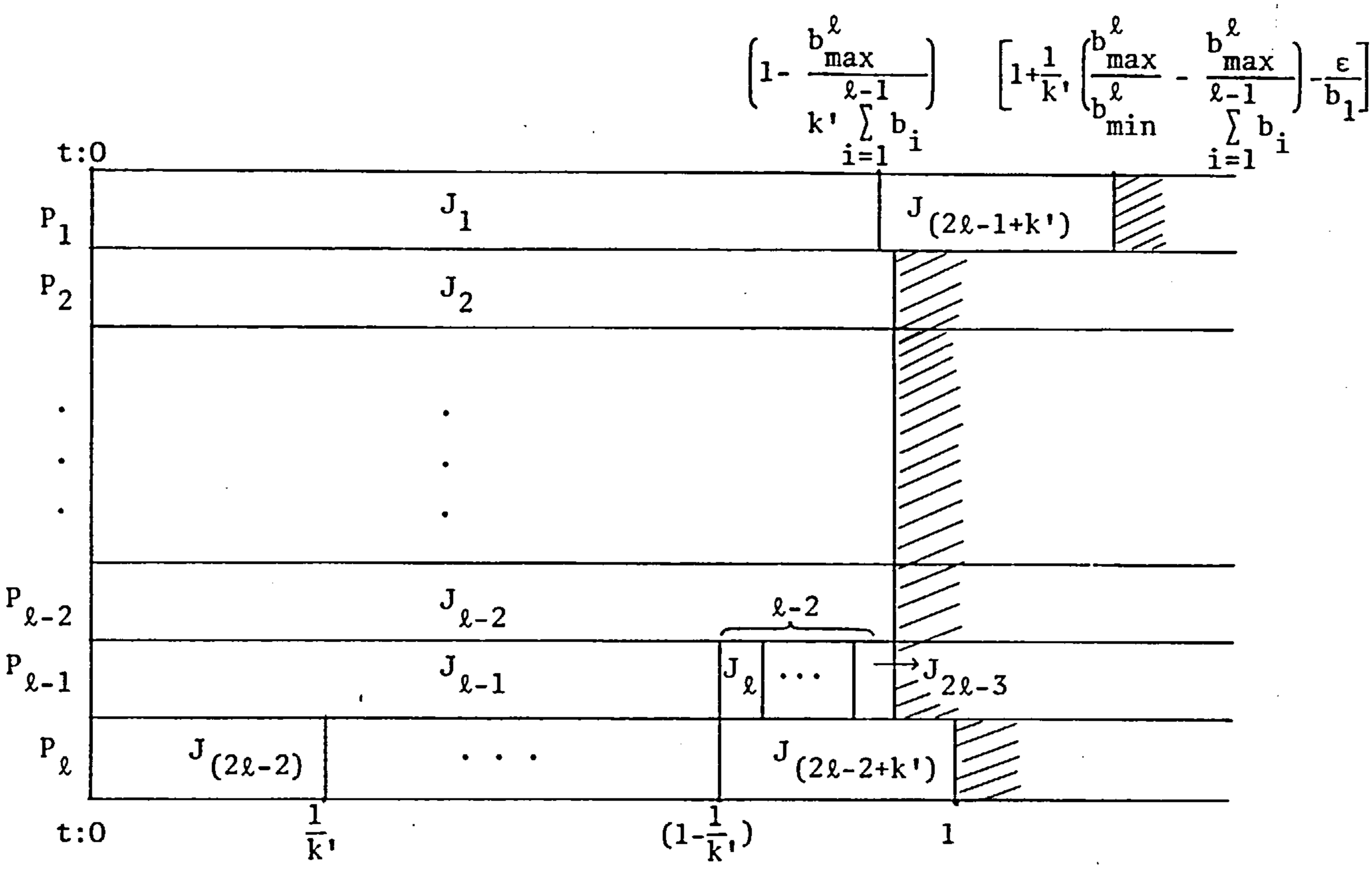


FIGURE 6.12: Schedule produced by the P.D. algorithm for the priority list given in Example 6.6

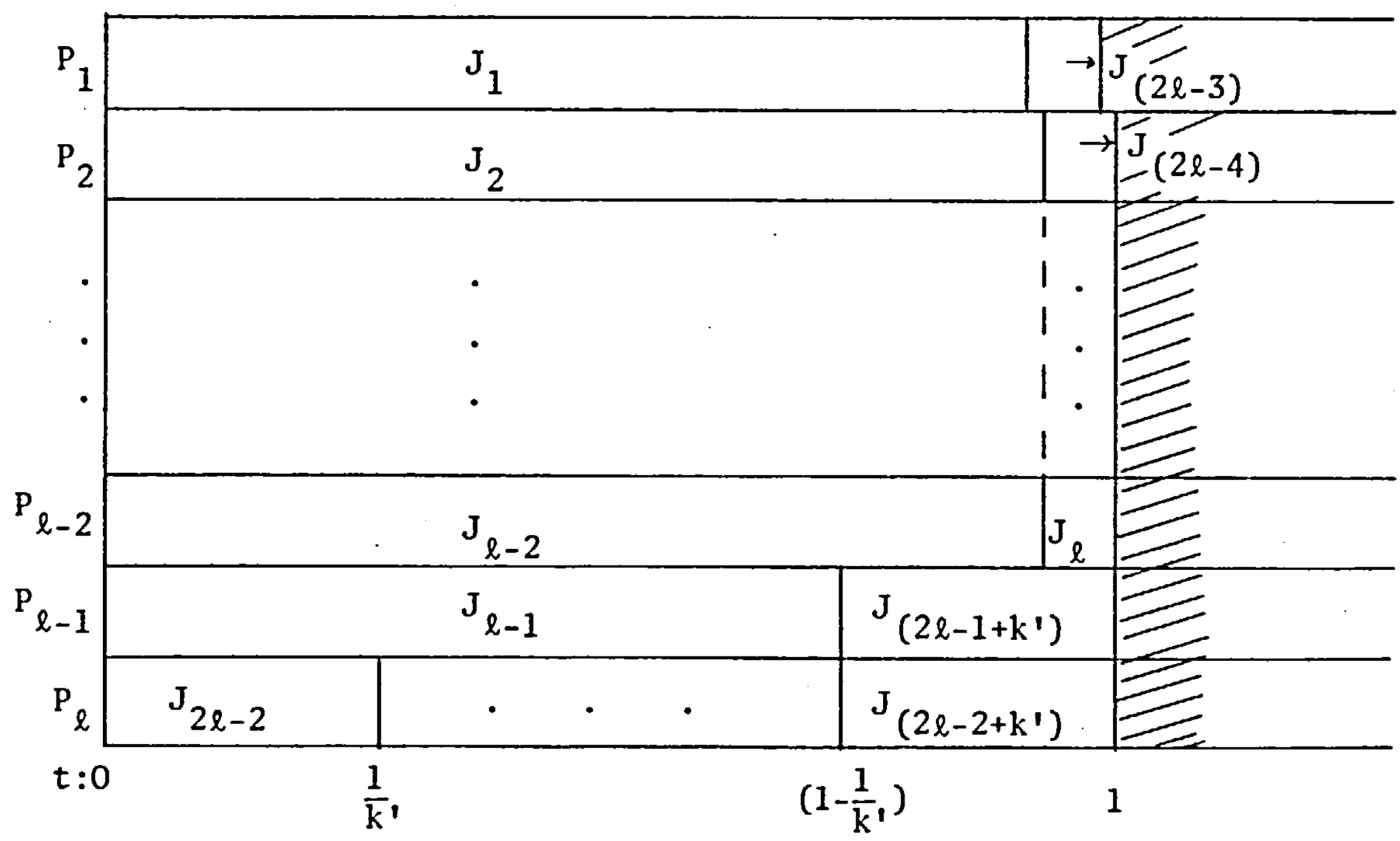


FIGURE 6.13: The optimal schedule for the task system given in Example 6.6.

The ratio of the completion times of the above schedules, as $\varepsilon \to 0$ is:

$$\frac{\omega_{PD}}{\omega_{OPT}} = 1 + \frac{1}{k'}\left(\frac{b^{\ell}_{max}}{b^{\ell}_{min}} - \frac{b^{\ell}_{max}}{\sum_{i=1}^{\ell-1} b_i}\right) \quad ,$$

which is the value of the worst-case bound proven for the corresponding case in Theorem 6.2.5¤

We can see that the established bounds for the P.D. algorithm, when the priority list is in a $LMLT_{MIN}$ or $LMLT_{MAX}$ ordering, depends mainly on the values of k' and ℓ. In effect, as ℓ increases and k' is constant the values of the worst-case performance bounds increase since, λ^*_{ℓ} or $(b^{\ell}_{max}/b^{\ell}_{min})$ may increase as well. On the other hand, when k' increases and ℓ is constant the extreme performance of the algorithm becomes more close to the optimal one. However, the value of k' is expected to increase as the number of the jobs in the task system increases. Thus, the greater the number of jobs in the task system, the better the worst-case bound becomes.

Further, in the pathological situation where k'=1 the bounds of the last theorem become close to the corresponding ones found in Theorem 6.2.3, when the priority list is in a LMF ordering. (Actually, $(\omega_{PD}/\omega_{OPT}) \leq \lambda^*_{\ell}(2-(1/(\ell-1)))$ and $(\omega_{PD}/\omega_{OPT}) \leq 1+(b^{\ell}_{max}/b^{\ell}_{min})-(b^{\ell}_{max}/\sum_{\substack{i=1 \\ i\neq x}}^{\ell} b_i)$ for non-identical and uniform processors respectively.) Finally, we could say that when the priority list is in a LMF ordering, the extra effort to arrange the jobs with $\ell_j=i$ in a LTF ordering, i=1(1)m, can offer an improvement in the extreme performance of the algorithm. The degree of improvement depends mainly on the values of k' and ℓ; in fact, it becomes higher when the values of k' or ℓ are increased. For $\lambda^*_{\ell}=1$ or $b_i=b_{i+1}$, $1 \leq i \leq m-1$, the bounds of the Theorem 6.2.4 agree with the corresponding ones found for the homogeneous multiprocessor system with independent memories. (See Theorem III.3, Appendix III.)

Theorem 6.2.5: Let the priority list be in a LTF_{MAX} or LTF_{MIN} ordering. Then,

(i) for non-identical processors:

$$\max\left\{\frac{\omega_{PD}}{\omega_{OPT}}\right\} \gtrsim \lambda(1 + H_{\lceil\lambda\rceil+m-1} - H_{\lceil\lambda\rceil+\ell-1}) \approx \lambda\left[1 + \ell n \left(\frac{\lceil\lambda\rceil+m-1}{\lceil\lambda\rceil+\ell-1}\right)\right] \quad \text{for } 2\lesssim\ell\lesssim m-1,$$

$$\max\left\{\frac{\omega_{PD}}{\omega_{OPT}}\right\} \gtrsim 1 + \lambda(H_{\lceil\lambda\rceil+m-1} - H_{\lceil\lambda\rceil+\ell-1}) \approx 1 + \lambda\left[\ell n \left(\frac{\lceil\lambda\rceil+m-1}{\lceil\lambda\rceil+\ell-1}\right)\right] \quad \text{for } \ell=1, \text{ and}$$

$$\frac{\omega_{PD}}{\omega_{OPT}} \lesssim \lambda, \qquad k^*=1,2,$$

$$\frac{\omega_{PD}}{\omega_{OPT}} \lesssim \lambda\left(1 + \frac{1}{k^*} - \frac{1}{k^*m}\right), \qquad k^*\gtrsim 3 \quad \text{for } \ell=m;$$

(ii) for uniform processors:

$$\max\left\{\frac{\omega_{PD}}{\omega_{OPT}}\right\} \gtrsim 1 + \frac{1}{b_{max}} \left(\sum_{j=\ell+1}^{m} \frac{b_j}{j}\right) \quad \text{for } 1\lesssim\ell\lesssim m-1, \text{ and}$$

$$\frac{\omega_{PD}}{\omega_{OPT}} \lesssim \frac{b_{max}}{b_{min}}, \qquad k^*=1,2,$$

$$\frac{\omega_{PD}}{\omega_{OPT}} \lesssim 1 + \frac{1}{k^*}\left(\frac{b_{max}}{b_{min}} - \frac{b_{max}}{\sum_{i=1}^{m} b_i}\right) \quad \text{for } \ell=m,$$

where λ and ℓ are as defined in Lemma 5.2.6 and Theorem 6.2.1 respectively, H_n is the harmonic number (i.e., $H_n = \sum_{i=1}^{n} \frac{1}{i}$, $n\gtrsim 0$) and k^* is the maximum number of jobs, amongst the first w, scheduled on any particular processor.

Proof: (i) To prove the lower limit of the worst-case performance of the P.D. algorithm under the LTF_{MAX} or LTF_{MIN} rule, a method will be given which achieves the bound for arbitrary m, while ℓ can vary from 1 to (m-1).

The general example consists of $(m-\ell+1)$ groups of jobs. In the priority list the jobs in the $(k+1)^{st}$ group precede those in group k, $1\lesssim k\lesssim(m-\ell+1)$. The number of tasks in each of the last $(m-\ell)$ groups is $[2^{(m-\ell+1-k)}\times(k+\ell-1)]$, $2\lesssim k\lesssim(m-\ell+1)$ and k is the group index. The tasks in

the k^{th} group have memory requirement $|P_{(k+\ell-1)}|$. However, the time requirements of the jobs belonging to the $(m-\ell+1)^{st}$ group as well as to any group k, $2 \leq k \leq (m-\ell)$, are as given in Fig.6.14(a) and (b) respectively. On the contrary, the first group has $[(2^{(m-\ell+r_1)}+2^{(m-\ell)} \times \ell)\ell]$ jobs, where r_1 is the smallest integer which satisfies the condition $2^{r_1} \geq \lambda$, with memory requirements $|P_\ell|$ and time requirement as given in Fig.6.14(c). The jobs of each group appear in the priority list in an increasing sequence according to their job index and so the LTF_{MAX} or LTF_{MIN} ordering rule is preserved. We should notice, that the jobs in each particular group have the same maximum time requirement (i.e., for the k^{th} group, $\sigma_k = \lambda/(2^{(m-\ell+1-k)} \times (\lambda+k+\ell-2))$, $2 \leq k \leq m-\ell+1$, whereas for the first group, $\sigma_1 = \lambda/(2^{(m-\ell+r_1)}+2^{(m-\ell)} \times \ell)$.

Now, if we apply the P.D. algorithm to schedule the jobs then, the m tasks of the $(m-\ell+1)^{st}$ group, which appear first in the priority list, will terminate within an ε-time difference as the processors' index increases and will contribute to the schedule length a time interval of length $[(\lambda/(\lambda+m-1))-h(\varepsilon)]$, $0 \leq h(\varepsilon) \leq (m-1)\varepsilon$. Consequently, since the number of jobs

(a)

	m			
1	$\frac{\lambda}{\gamma_{m-\ell+1}}$	$\frac{\lambda}{\gamma_{m-\ell+1}}$	. . .	$\left(\frac{\lambda}{\gamma_{m-\ell+1}} - (m-1)\varepsilon\right)$
2	$\frac{\lambda}{\gamma_{m-\ell+1}}$	$\frac{\lambda}{\gamma_{m-\ell+1}}$	. . .	$\frac{\lambda}{\gamma_{m-\ell+1}}$
	.	.		.
	.	.		.
	.	.		.
m-2	$\frac{\lambda}{\gamma_{m-\ell+1}}$	$\frac{\lambda}{\gamma_{m-\ell+1}}$	. . .	$\frac{\lambda}{\gamma_{m-\ell+1}}$
m-1	$\frac{\lambda}{\gamma_{m-\ell+1}}$	$\left(\frac{\lambda}{\gamma_{m-\ell+1}} - \varepsilon\right)$	. . .	$\frac{\lambda}{\gamma_{m-\ell+1}}$
m	$\frac{\lambda}{\gamma_{m-\ell+1}}$	$\frac{1}{\gamma_{m-\ell+1}}$	. . .	$\frac{1}{\gamma_{m-\ell+1}}$

$\} \ . \ . \ .$

$$\left(J_1^{m-\ell+1}, J_2^{m-\ell+1}, \ . \ . \ . \ , J_m^{m-\ell+1} \right)$$

(b)

	$2^{(m-\ell+1-k)}$	
	$k+\ell-1$	$k+\ell-1$
1	$\frac{\lambda}{\gamma_k}\ \frac{\lambda}{\gamma_k}\ \ldots\ \frac{1}{\gamma_k}$	. . .
.	. . .	
.	. . .	
.	. . .	
	$\frac{\lambda}{\gamma_k}\ \frac{\lambda}{\gamma_k}\ \ldots\ \frac{1}{\gamma_k}$	
$k+\ell-1$	$\frac{1}{\gamma_k}\ \frac{1}{\gamma_k}\ \ldots\ \frac{\lambda}{\gamma_k}$	. . . ...
.	$\infty\quad \infty\ \ldots\ \infty$	
.	. . .	
.	. . .	
.	. . .	
m	$\infty\quad \infty\ \ldots\ \infty$	. . .

$$(J^k_1,\ J^k_2,\ \ldots,\ J^k_{k+\ell-1},\ \ldots,\ J^k_{2^{(m-\ell+1-k)}\times(k+\ell-1)})$$

(c)

	$\overbrace{\qquad\qquad\qquad}^{\ell}$	$\overbrace{\qquad\qquad\qquad\qquad}^{(2^{(m-\ell+r_1)}+2^{(m-\ell)}\times\ell)-\ell}$
1	$\frac{\lambda}{\gamma_1}\ \frac{1}{\gamma_1}\ \ldots\ \frac{1}{\gamma_1}$	. . .
.	$\frac{1}{\gamma_1}\ \frac{\lambda}{\gamma_1}\ \ldots\ \frac{1}{\gamma_1}$	
.	$\frac{1}{\gamma_1}\ \frac{1}{\gamma_1}\ \ldots\ \frac{1}{\gamma_1}$	
.	. . .	
.	. . .	...
.	. . .	
	$\frac{1}{\gamma_1}\ \frac{1}{\gamma_1}\ \ldots\ \frac{1}{\gamma_1}$	
ℓ	$\frac{1}{\gamma_1}\ \frac{1}{\gamma_1}\ \ldots\ \frac{\lambda}{\gamma_1}$	
.	$\infty\quad \infty\ \ldots\ \infty$	
.	. . .	
.	. . .	
.	. . .	
m	$\infty\quad \infty\ \ldots\ \infty$	

$$(J^1_1,\ J^1_2,\ \ldots\ J^1_\ell,\ \ldots,\ J^1_{(2^{(m-\ell+r_1)}+2^{(m-\ell)}\times\ell)})$$

NOTATION: $\gamma_k=(2^{(m-\ell+1-k)}\times(\lambda+k+\ell-2))$ for $k=2(1)(m-\ell+1)$ and $\gamma_1=(2^{m-\ell+r_1}+2^{(m-\ell)}\times\ell)$

FIGURE 6.14: Time requirements and the sequence of jobs (a) in the $(m-\ell+1)^{st}$ group, (b) in the k^{th} group, $2\leqslant k\leqslant m-\ell$ and (c) in the first group.

in the k^{th} group, $1 \leq k \leq m-\ell$, is a multiple of the number of processors capable of executing those tasks, and since the P.D. algorithm allocates the maximum time requirement of the jobs to each of these processors, the last $(k+\ell-1)$ jobs of the k^{th} group will also terminate within an ε-time difference. In addition, one can realise that the contribution of the jobs in the k^{th} group, $2 \leq k \leq (m-\ell)$ will be a solid block of length $(\lambda/(\lambda+k+\ell-2))$, while the jobs of the first group will contribute a time interval of length λ. Therefore, the total schedule length is $[\lambda + \sum_{k=2}^{(m-\ell+1)} (\lambda/(\lambda+k+\ell-2)) - h(\varepsilon)]$. However, the appearance of the general schedule for the above mentioned task system, using the P.D. algorithm, is shown in Fig.6.15. We shall present now that an optimal schedule of length 1 can be formed. Clearly, if the jobs of the 1^{st} group appear at the beginning in the priority list, followed by the jobs of the k^{th} group, for $k=2(1)(m-\ell+1)$, then, the P.D. algorithm can construct a schedule of length 1.

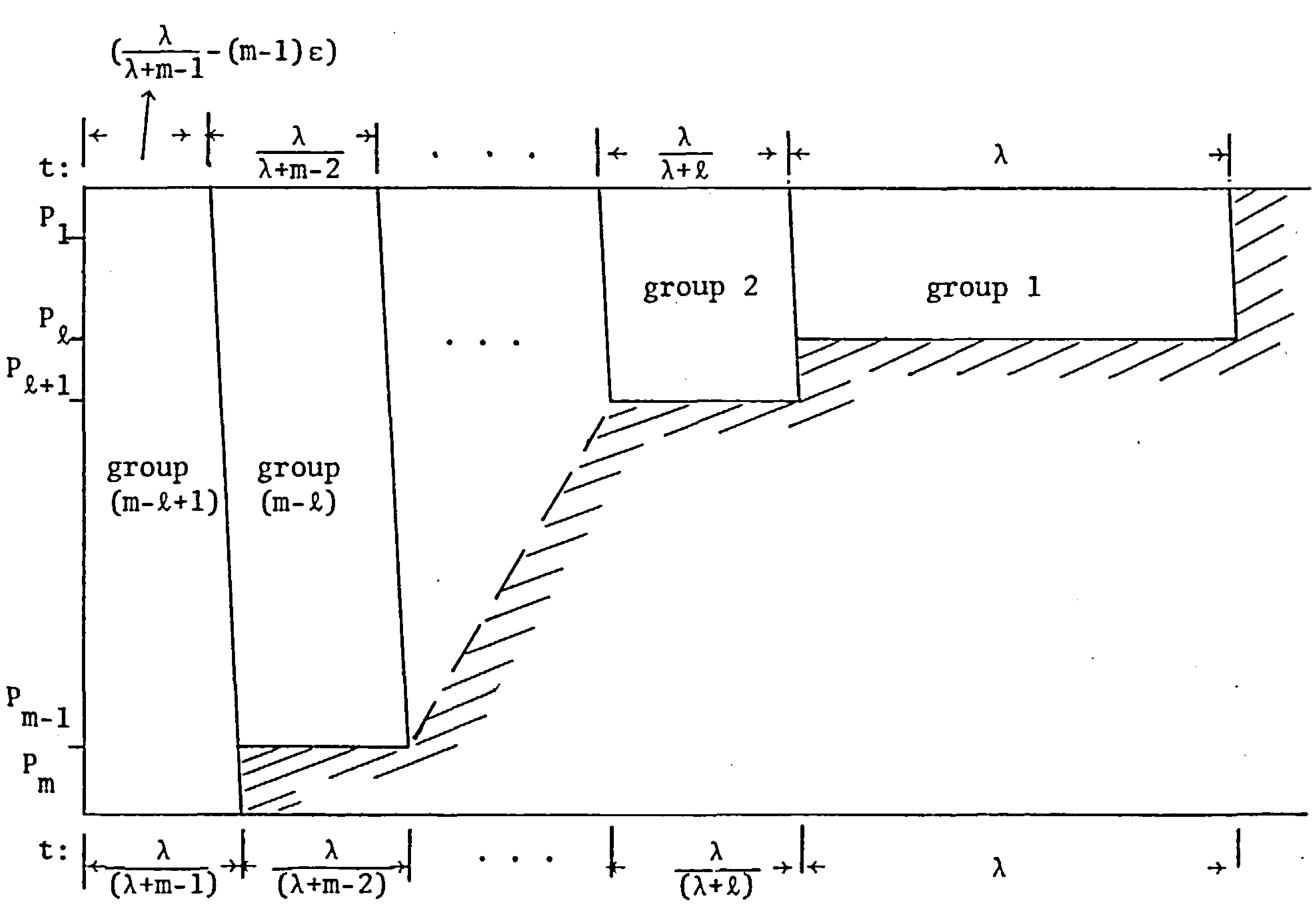


FIGURE 6.15: The general schedule produced by the P.D. algorithm

Therefore, as $\varepsilon \to 0$

$$\frac{\omega_{PD}}{\omega_{OPT}} = \lambda + \sum_{k=2}^{(m-\ell+1)} (\lambda/(\lambda+k+\ell-2))$$

or, since $\left(\frac{\lambda}{\lambda+k+\ell-2}\right) \geq \frac{\lambda}{\lceil\lambda\rceil+k+\ell-2}$,

$$\frac{\omega_{PD}}{\omega_{OPT}} = \lambda\left(1 + \sum_{k=2}^{(m-\ell+1)} \left(\frac{1}{\lceil\lambda\rceil+k+\ell-2}\right)\right.$$

$$= \lambda\left(1 + \sum_{i=1}^{(\lceil\lambda\rceil+m-1)} \left(\frac{1}{i}\right) - \sum_{i=1}^{(\lceil\lambda\rceil+\ell-1)} \left(\frac{1}{i}\right)\right)$$

$$= (1 + H_{\lceil\lambda\rceil+m-1} - H_{\lceil\lambda\rceil+\ell-1}) \ .$$

However, since $H_n = \ell n(n) + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} - \varepsilon, \quad 0<\varepsilon<\frac{1}{252n^6}$, where

γ=0.577215664 ... is Euler's constant, we can approximate H_n by $\ell n(n)$. Thus,

$$\frac{\omega_{PD}}{\omega_{OPT}} = \lambda(1+H_{\lceil\lambda\rceil+m-1} - H_{\lceil\lambda\rceil+\ell-1}) \simeq \lambda\left[1+\ell n\left(\frac{\lceil\lambda\rceil+m-1}{\lceil\lambda\rceil+\ell-1}\right)\right] \ .$$

So, the bound produced by the example achieves the lower bound given by the theorem in part (i) for $2 \leq \ell \leq m-1$.

If ℓ=1, we can see that the contribution of group 1 will be a time interval of length 1 and hence in this case,

$$\max\left\{\frac{\omega_{PD}}{\omega_{OPT}}\right\} \geq 1 + \lambda\, H_{\lceil\lambda\rceil+m-1} - H_{\lceil\lambda\rceil+\ell-1} \simeq 1 + \lambda\left[\ell n\left(\frac{\lceil\lambda\rceil+m-1}{\lceil\lambda\rceil+\ell-1}\right)\right].$$

Further, for ℓ=m we can distinguish the following cases which depend on the value of k*. When k*=1, then it is obvious that the completion time of any valid schedule can not be worse than $\lambda\omega_{OPT}$. For k*=2, since ℓ_w=m, the J_w task will be scheduled on the processor, amongst those with one job already being executed, which becomes available first, according to the P.D. algorithm. Therefore, as for the case of k*=1, $\omega_{PD} \leq \lambda\omega_{OPT}$. This results in

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \lambda \qquad \text{for } k^*=1,2.$$

For $k^* \geq 3$, since,

$$\omega'_{PD} = U_i + I_i \quad ,$$

$$\omega'_{OPT} \geq \left(\sum_{i=1}^{m} U_i \right) / (\lambda m),$$

$$I_i \leq \lambda \frac{\omega_{OPT}}{k^*} \quad , \qquad 1 \leq i \leq m, \qquad i \neq x$$

and $I_x = 0$,

following a similar analysis as the one used in part (i) of Theorem 6.2.4, we can obtain

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \lambda \left(1 + \frac{1}{k^*} - \frac{1}{k^* m_n} \right) \quad , \qquad k^* \geq 3.$$

(ii) A similar approach to the one given in part (i) of this theorem can achieve the lower limit bound of part (ii), when $1 \leq \ell \leq m-1$. In particular, the general example consists of $(m-\ell+1)$ groups of jobs. The processing speeds of the processors are considered to be in the $b_1 \geq b_2 \geq \ldots \geq b_m$ order. Further, although the number of tasks of the $(m-\ell+1)^{st}$ group is $n_m = m$, the number of jobs in the k^{th} group (i.e., $n_{k+\ell-1}$), when the number of tasks in the $(k+1)^{st}$ group (i.e. $n_{k+\ell}$) is known, is the smallest integer such that $n_{k+\ell-1} \geq c$ and $(n_{k+\ell-1}/(k+\ell-1)) = 0(\mathrm{mod}(v))$, $2 \leq k \leq m-\ell$ where $v \in Z^+$ and $c = n_{k+\ell}(b_{k+\ell-1}/b_{k+\ell})$. However, the memory and time requirements of the jobs in the k^{th} group, $2 \leq k \leq m-\ell+1$ are $|P_{k+\ell-1}|$ and $\{b_{k+\ell-1}/n_{k+\ell-1}\}$ respectively. In addition, the number of jobs in the first group is $n_\ell \ell$, where n_ℓ can be defined in a similar way as n_i, $\ell+1 \leq i \leq m$ but now $c = n_{\ell+1}(b_1/b_{\ell+1})$. The memory and time requirements of the n_ℓ jobs in the i^{th} subgroup, $1 \leq i \leq \ell$ are $|P_i|$ and $\{b_i/n_\ell\}$ respectively. Thus, in order to preserve a LTF ordering in the priority list the jobs of the $(k+1)^{st}$ group must precede the tasks of the k^{th} group. Then, the P.D. algorithm will result in a schedule of length at least $[1 + \sum_{j=\ell+1}^{m} b_j/(jb_{max})]$. The general appearance of the schedule is shown in Fig. 6.16. On the other hand, if we schedule the jobs of the k^{th} group,

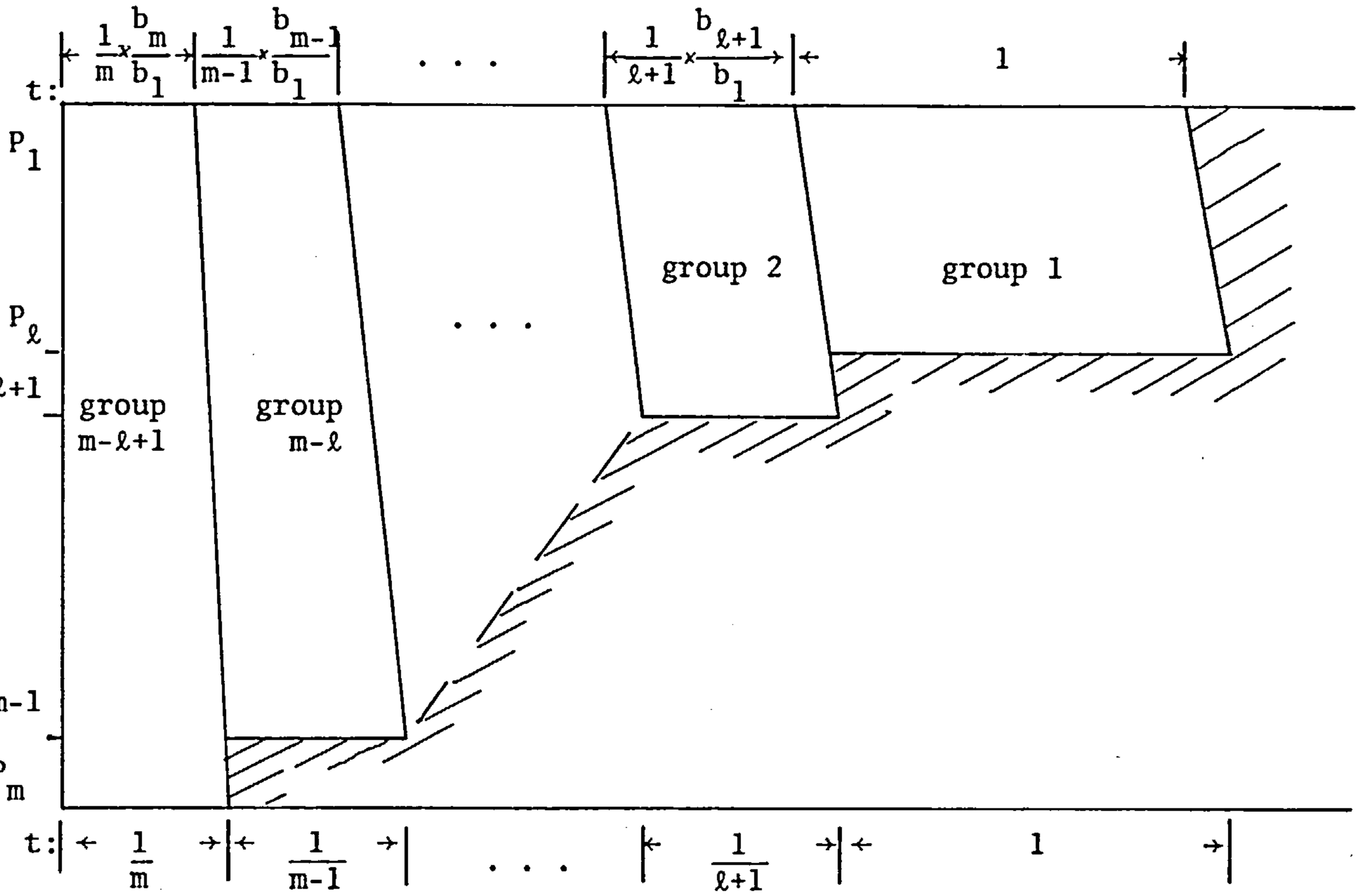


FIGURE 6.16: The general schedule for the task system given in part (ii) of the Theorem 6.2.5 when $1 \leq \ell \leq m-1$.

$2 \leq k \leq m-\ell+1$ on the processor $P_{k+\ell-1}$, then a schedule of length 1 will be constructed. Therefore, since $b_1 = b_{max}$ we have that

$$\frac{\omega_{PD}}{\omega_{OPT}} = 1 + \frac{1}{b_{max}} \left(\sum_{j=\ell+1}^{m} \frac{b_j}{j} \right) ,$$

which is the lower bound given in the present theorem for part (ii) when $1 \leq \ell \leq m-1$.

Finally, when $\ell = m$ and for $k^* = 1,2$ we can easily obtain

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \frac{b_{max}}{b_{min}} .$$

Furthermore, for $k^* \geq 3$, since

$$\omega'_{PD} = U_i + I_i, \qquad 1 \leq i \leq m ,$$

$$\omega'_{OPT} \geq \left(\sum_{i=1}^{m} (U_i b_i) \right) / \left(\sum_{i=1}^{m} b_i \right) ,$$

$$I_i \leq (\omega'_{OPT}/k^*)(b_{max}/b_{min}) \qquad \text{for } i=1(1)m, \quad i \neq x$$

and $I_x = 0$,

following a similar analysis as the one used in part (ii) of Theorem 6.2.4, we can prove that

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 1 + \frac{1}{k^*} \left(\frac{b_{max}}{b_{min}} - \frac{b_{max}}{\sum_{i=1}^{m} b_i} \right) ,$$

which completes the proof of the theorem ¤

For $1 \leq \ell \leq (m-1)$, although an upper limit on the worst-case bound has not been found, the lower limits given in Theorem 6.2.5 show the behaviour of the algorithm when the priority list is in the LTF_{MIN} or LTF_{MAX} ordering. So, as in the case of an arbitrary ordering the bound should increase logarithmically without any limit as m increases and ℓ is constant, whereas if ℓ increases and m is constant, the bound should decrease logarithmically. In addition, one can realise that as ℓ becomes close to m, the extreme performance of the algorithm improves and for $\ell=m$ the values of $[\lambda(1+(1/k^*)-(1/(k^*m)))]$ and $[1+(1/k^*)((b_{max}/b_{min})-(b_{max}/(\sum_{i=1}^{m} b_i)))]$, $k^* \geq 3$ are obtained for non-identical and uniform processors respectively. Moreover, if $k^* > k'$, these values of the worst-case bound might be better than their corresponding ones when the priority list is in a $LMLT_{MIN}$ or $LMLT_{MAX}$ ordering. Also, it should be noticed that when $\lambda=1$ or $b_i = b_{i+1}$, $1 \leq i \leq m-1$, the worst-case bounds of theorem 6.2.5 agree with the corresponding ones found for the homogeneous multiprocessor system with independent memories. (See Theorem III.4, Appendix III.)

Finally, from the established worst-case bounds for the P.D. algorithm, when the completion time is the performance criterion, one can realise that:

- they vary widely from one ordering rule to another;
- they are informative;
- most of them are best possible bounds ;
- the extreme performance may improve when the jobs in the priority list are arranged according to a heuristic ordering procedure; and
- the improvement depends on the values of the system parameters, which are involved in the expression of the bounds.

6.3 QUICK AND DIRTY (Q.A.D.) SCHEDULING ALGORITHMS

In order to complete the aims of this chapter we shall establish here, guaranteed levels of performance for the Q.A.D. algorithms, when the completion time is chosen as the performance criterion.

Let ω_{QAD} be the completion time of the schedule constructed by the Q.A.D. algorithm, when the priority list is formed by a heuristic ordering rule, and ω_{OPT} be the length of the optimal schedule for a given task system $(J,[t_{ij}])$, $1\leq i\leq m$, $1\leq j\leq n$.

Theorem 6.3.1: Let the priority list be in an arbitrary ordering (RAND). Then,

(i) for non-identical processors:

$$\frac{\omega_{QAD}}{\omega_{OPT}} \leq 1 + \lambda(\nu_x - 1)$$

(ii) for uniform processors:

$$\frac{\omega_{QAD}}{\omega_{OPT}} \leq \frac{\sum_{i=1}^{x} b_i}{b_{min}} ,$$

where λ and ν_x are as defined in Lemma 5.2.6 and Theorem 5.3.1, respectively and x is the index of the processor where the schedule finishes.

Proof: (i) The value of ω_{QAD} cannot exceed a time interval of length $[(\nu_x-1)\lambda\omega_{OPT}+\omega_{OPT}]$, because no more than ν_x jobs with $\tau_j=\omega_{OPT}$ and $\sigma_j=\lambda\omega_{OPT}$ can be scheduled on a processor P_r, $1\leq r\leq\nu_x$. This is true since these jobs can only be scheduled on the first ν_x processors and if $n'_x\geq\nu_x+1$ then $\omega_{OPT}\geq\frac{\sum_{i=1}^{n'_x}\tau_j}{\nu_x}\geq\frac{\nu_x+1}{\nu_x}\omega_{OPT}$, which is a contradiction. Moreover, at least one of them should require its minimum time on that processor, since otherwise $\omega_{OPT}=\omega_{OPT}+\frac{\lambda-1}{\nu_x}\omega_{OPT}$, which is also a contradiction. Therefore,

$$\omega_{QAD} \leq (\nu_x-1)\lambda\omega_{OPT} + \omega_{OPT}$$

or $$\frac{\omega_{QAD}}{\omega_{OPT}} \leq 1 + \lambda(\nu_x - 1).$$

(ii) Following a similar argument as in part (i) and considering P_x to be the processor with the lowest speed we can prove that

$$\frac{\omega_{QAD}}{\omega_{OPT}} \leq \frac{\sum_{i=1}^{\nu_x} b_i}{b_{min}} \quad . \quad \square$$

Moreover, both of the bounds are best possible ones. This can be realised from Examples 6.7 and 6.8, which are incorporated into the proof of the next theorem.

Theorem 6.3.2: Let the priority list be in a STF_{MIN}, STF_{MAX}, LMF, $LMST_{MIN}$, $LMST_{MAX}$, $LMLT_{MIN}$ or $LMLT_{MAX}$ ordering. Then,

(i) for non-identical processors:

$$\frac{\omega_{QAD}}{\omega_{OPT}} \leq 1 + \lambda(\nu_x - 1) \ ;$$

(ii) for uniform processors

$$\frac{\omega_{QAD}}{\omega_{OPT}} \leq \frac{\sum_{i=1}^{\nu_x} b_i}{b_{min}} \ ,$$

where λ, ν_x and x are as defined in Lemma 5.2.6, and Theorems 5.3.1 and 6.3.1, respectively.

Proof: Since the upper bounds for non-identical and uniform processors in Theorem 6.3.1 proved for an arbitrary priority list, these bounds can be considered as extreme performance levels for any priority list when the Q.A.D. algorithm is used to construct the schedule. In particular, we shall present examples which can cause the performance of the Q.A.D. algorithm under the LMF, $LMST_{MIN}$ or $LMST_{MAX}$ ordering rules to deviate from the optimal completion time by the value allowed by the worst-case bounds stated in the theorem.

Example 6.7: Let the task system $(J,[t_{ij}])$ be defined by the set of independent jobs $J=\{J_1,J_2,\ldots,J_n\}$ and the following (m×n) matrix, where $\varepsilon>0$ is a very small quantity and $n=\nu_x(\nu_x-1)\lceil\lambda\rceil+x$.

The priority list $L=(J_1,J_2,\ldots,J_n)$ is in $LMST_{MIN}$ or $LMST_{MAX}$ and hence in LMF as well. The schedule resulting from the priority list L, when the Q.A.D. algorithm is used, is shown in Fig.6.17. On the other hand, if the ν_x jobs scheduled on the $|P_x|$ processor are allocated each one on a different processor amongst the first ν_x, then a schedule of length $(\omega_{OPT}+\nu_x\lceil\lambda\rceil\varepsilon)$ can be obtained.

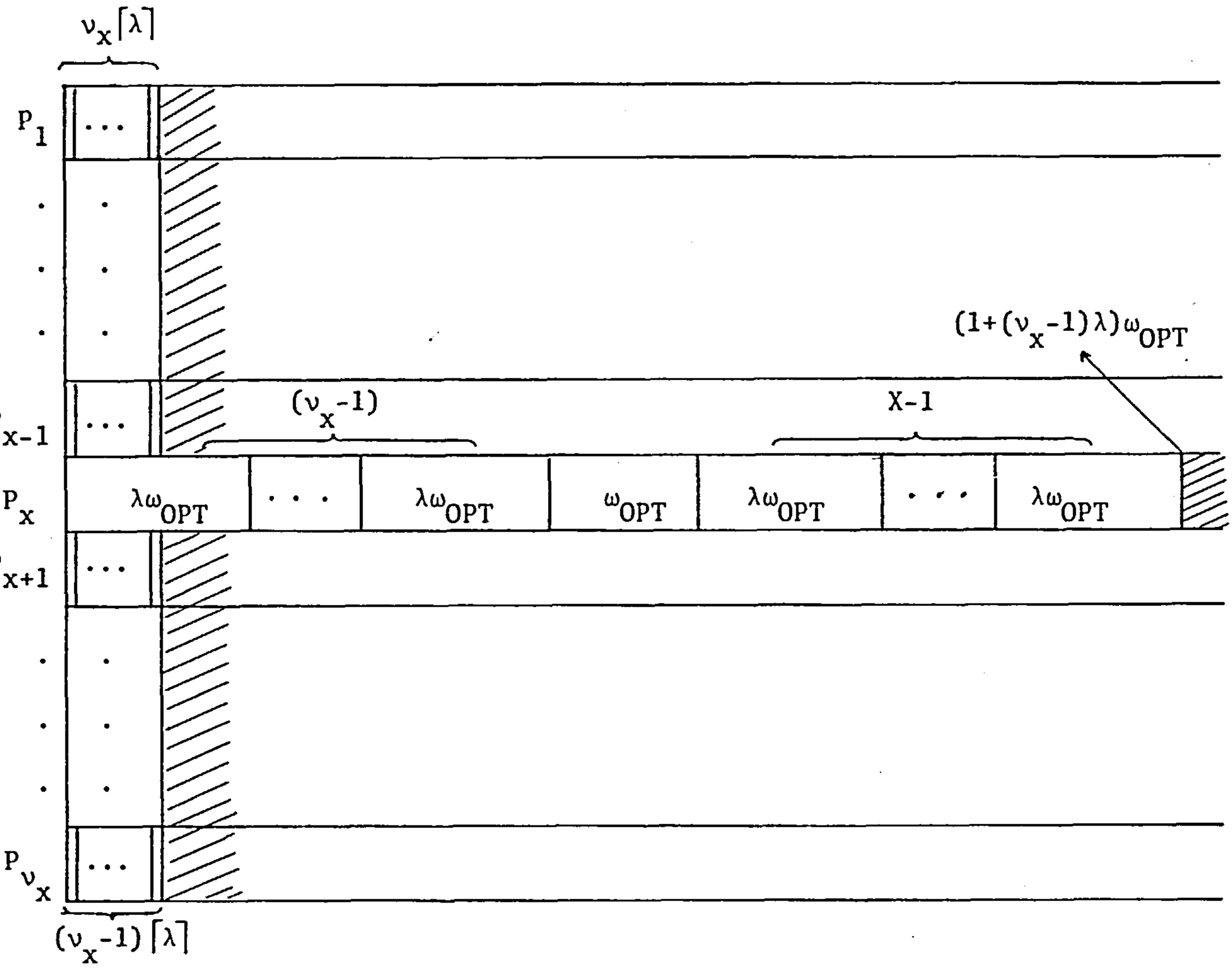


FIGURE 6.17: Schedule produced by the Q.A.D. algorithm for the priority list given in Example 6.7.

$$
[t_{ij}] = \left[\begin{array}{cccccccccccccccccc|l}
\overbrace{\varepsilon \cdots \varepsilon}^{\nu_x\lceil\lambda\rceil} & \cdots & \overbrace{\varepsilon \cdots \varepsilon}^{\nu_x\lceil\lambda\rceil} & \overbrace{\omega_{OPT} \cdots \omega_{OPT}}^{x-1} & \lambda\omega_{OPT} & \overbrace{\varepsilon \cdots \varepsilon}^{(i+1)\lceil\lambda\rceil-1} & \omega_{OPT} & \overbrace{\varepsilon \cdots \varepsilon}^{(i+3)\lceil\lambda\rceil-1} & \omega_{OPT} & \cdots & \overbrace{\varepsilon \cdots \varepsilon}^{[2\nu_x-(x+1)]\lceil\lambda\rceil-1} & \omega_{OPT} & |P_1| \\
\infty \cdots \infty & \cdots & \varepsilon \cdots \varepsilon & \omega_{OPT} \cdots \omega_{OPT} & \lambda\omega_{OPT} & \varepsilon \cdots \varepsilon & \omega_{OPT} & \varepsilon \cdots \varepsilon & \omega_{OPT} & \cdots & \varepsilon \cdots \varepsilon & \omega_{OPT} & \\
\vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \\
\vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \\
 & & \varepsilon \cdots \varepsilon & \omega_{OPT} \cdots \omega_{OPT} & \lambda\omega_{OPT} & \varepsilon \cdots \varepsilon & \omega_{OPT} & \varepsilon \cdots \varepsilon & \omega_{OPT} & \cdots & \varepsilon \cdots \varepsilon & \omega_{OPT} & |P_{x-1}| \\
\infty \cdots \infty & \cdots & \infty \cdots \infty & \lambda\omega_{OPT} \cdots \lambda\omega_{OPT} & \omega_{OPT} & \lambda\varepsilon \cdots \lambda\varepsilon & \lambda\omega_{OPT} & \lambda\varepsilon \cdots \lambda\varepsilon & \lambda\omega_{OPT} & \cdots & \lambda\varepsilon \cdots \lambda\varepsilon & \lambda\omega_{OPT} & |P_x| \\
 & & & \infty \cdots \infty & \infty & \varepsilon \cdots \varepsilon & \omega_{OPT} & \varepsilon \cdots \varepsilon & \omega_{OPT} & \cdots & \varepsilon \cdots \varepsilon & \omega_{OPT} & |P_{x+1}| \\
 & & & & & \infty \cdots \infty & \infty & \varepsilon \cdots \varepsilon & \omega_{OPT} & & \vdots & \vdots & \\
 & & & & & & & \infty \cdots \infty & \infty & & \vdots & \vdots & \\
\vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \varepsilon \cdots \varepsilon & \omega_{OPT} & |P_{\nu_x}| \\
\vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \infty \cdots \infty & \infty & \\
\vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \\
\vdots & & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \\
\infty & \cdots & \infty & \infty & \cdots & \infty & \cdots & \infty & \cdots & \cdots & \infty & \infty & |P_m|
\end{array}\right]
$$

$$
(J_1, \quad \cdots \quad, J_{\nu_x(x-1)\lceil\lambda\rceil}, \quad \cdots \quad, J_n)
$$

Therefore, the ratio of the completion times of these two schedules becomes,

$$\frac{\omega_{QAD}}{\omega_{OPT}} = \frac{(1+(\nu_x-1)\lambda)\omega_{OPT}}{\omega_{OPT}+\nu_x\lceil\lambda\rceil\varepsilon}$$

or $$\lim_{\varepsilon\to 0} \frac{\omega_{QAD}}{\omega_{OPT}} = 1+(\nu_x-1)\lambda \quad ,$$

which is the worst value of the bound given in part (i) of the theorem.

Example 6.8: Let the jobs $\{J_1,J_2,\ldots,J_n\}$ of the task system $(J,\{m_j\},\{T_j\})$ be defined by:

$J_{(i-1)\nu_x r+j}$: $(|P_i|, \varepsilon b_i)$, $r=\left\lceil\frac{b_{max}}{b_{min}}\right\rceil$, $j=1(1)(\nu_x r)$, $1\leq i\leq x-1$

J_j : $(|P_x|, b_j\omega_{OPT})$, $j=(\alpha+1)(1)(\alpha+x)$, $\alpha=(x-1)\ \nu_x r$

$J_{\alpha+x+j}$: $(|P_{x+1}|,\varepsilon b_{x+1})$, $j=1(1)((x+1)r-1)$

J_β : $(|P_{x+1}|, b_{x+1}\omega_{OPT})$, $\beta=\alpha+x+(x+1)r-1$

$J_{\beta+j}$: $(|P_{x+2}|, \varepsilon b_{x+2})$, $j=1(1)(x+2)r$

J_γ : $(|P_{x+2}|, b_{x+2}\omega_{OPT})$, $\gamma=\beta+(x+2)r$

⋮

J_n : $(|P_{\nu_x}|, b_{\nu_x}\omega_{OPT})$, $n=\nu_x(\nu_x-1)r+x$,

where $\varepsilon>0$ is a very small quantity $b_{x+1}=b_{x+2}=\ldots=b_{\nu_x}=b_{max}$.

The priority list $L=(J_1,J_2,\ldots,J_n)$ is again in LMST and hence in LMF ordering as well. Then, using the Q.A.D. algorithm a similar schedule to the one given in Fig.6.17 can be drawn, with completion time $(\sum_{i=1}^{\nu_x} b_i/b_{min})\omega_{OPT}$. However, it can be realised that a schedule of length $(\omega_{OPT}+\left\lceil\frac{b_{max}}{b_{min}}\right\rceil \nu_x\varepsilon)$ can be formed. So, the ratio of these schedules as $\varepsilon\to 0$ is

$$\lim_{\varepsilon\to 0} \frac{\omega_{QAD}}{\omega_{OPT}} = \frac{\sum_{i=1}^{\nu_x} b_i}{b_{min}} \quad ,$$

which is the worst-case bound given in part (ii) of the theorem.

Finally, when the priority list is in the STF_{MIN}, STF_{MAX}, $LMLT_{MIN}$ or $LMLT_{MAX}$ ordering then, the proven worst-case bounds are best possible only if $\nu_x=x$. This is realised by considering relaxed versions of the examples mentioned above.

Theorem 6.3.2 indicates that the use of the heuristic ordering rules STF_{MIN}, STF_{MAX}, LMF, $LMST_{MIN}$, $LMST_{MAX}$, $LMLT_{MIN}$ or $LMLT_{MAX}$ to form the priority list does not offer any improvement in a worst-case sense over an arbitrary ordering for the Q.A.D. algorithm, when the completion time is the performance criterion. Moreover, comparing the Q.A.D. and the corresponding P.D. algorithms, when the priority list is constructed by one of the above mentioned ordering rules, we can easily realise that the worst-case performance bounds of the Q.A.D. algorithms are always worse. Generally, they worsen dramatically as the number of processors in the system increases. In effect, the worst-case performance deviates from the optimal one as m increases (since ν_x may increase as well).

Theorem 6.3.3: Let the priority list be in a LTF_{MIN} or LTF_{MAX} ordering. Then,

(i) for non-identical processors:

$$\max\left\{\frac{\omega_{QAD}}{\omega_{OPT}}\right\} \geq \lambda(1+H_m-H_{\ell^*}) \simeq \lambda[1+\ell n(\frac{m}{\ell^*})], \text{ for } 1\leq\ell^*\leq m-1 \text{ and}$$

$$\frac{\omega_{QAD}}{\omega_{OPT}} \leq \lambda(1+\frac{m-1}{\bar{k}+1}), \text{ if } \bar{k}\geq m-1 \text{ and}$$

$$\frac{\omega_{QAD}}{\omega_{OPT}} < \lambda(1+\frac{\bar{k}}{\bar{k}+1}), \text{ otherwise, for } \ell^*=m;$$

(ii) for uniform processors:

$$\max\left\{\frac{\omega_{QAD}}{\omega_{OPT}}\right\} \geq 1+\frac{1}{b_{max}}\left(\sum_{j=\ell+1}^{m}\frac{b_j}{j}\right), \text{ for } 1\leq\ell^*\leq m-1 \text{ and}$$

$$\frac{\omega_{QAD}}{\omega_{OPT}} \leq 1 + \left(\frac{1}{\bar{k}+1}\right)\left(\frac{\sum_{i=1}^{m-1} b_i}{b_{min}}\right), \quad \text{if } \bar{k} \geq m-1 \text{ and}$$

$$\frac{\omega_{QAD}}{\omega_{OPT}} < 1 + \left(\frac{1}{\bar{k}+1}\right)\left(\frac{\sum_{i=1}^{\bar{k}} b_i}{b_{min}}\right) \quad \text{otherwise, for } \ell^*=m,$$

where λ, and H_n are as defined in Lemma 5.2.6 and Theorem 6.2.5 respectively, $\ell^*=\ell_\alpha$ and J_α is the longest job amongst those with $\ell_j \leq \ell_\alpha$, which commences execution at t=0 and $\bar{k}$ is the number of jobs, amongst the first w, allocated prior to s_w on the processor which finishes the schedule.

Proof: (i) To prove the lower limit of the worst-case performance of the Q.A.D. algorithm under the LTF_{MIN} or LTF_{MAX} rule when ℓ^* varies from 1 to (m-1), a general example similar to the one used in part (i) of Theorem 6.2.5 must be considered.

Although the general task system will not be given, this can be realised from the appearance of the schedule in Fig.6.18 (since it is similar to the corresponding one in part (i) of Theorem 6.2.5). In addition an optimal schedule of length 1 can also be found here, using the same way as in the above mentioned theorem.

Therefore,

$$\frac{\omega_{QAD}}{\omega_{OPT}} = \lambda\left(1 + \sum_{i=\ell+1}^{m} \frac{1}{i}\right) = \lambda(1 + H_m - H_{\ell^*}) \simeq \lambda(1+\ell n(\frac{m}{\ell^*})),$$

which is the lower value of the bound given in part (i) of the theorem when $1 \leq \ell^* \leq m-1$.

When $\ell^*=m$, following a similar analysis, as the one used for the corresponding case in Theorem 6.2.5, and bearing in mind that

$$\omega'_{QAD} = U_i + I_i \ ,$$

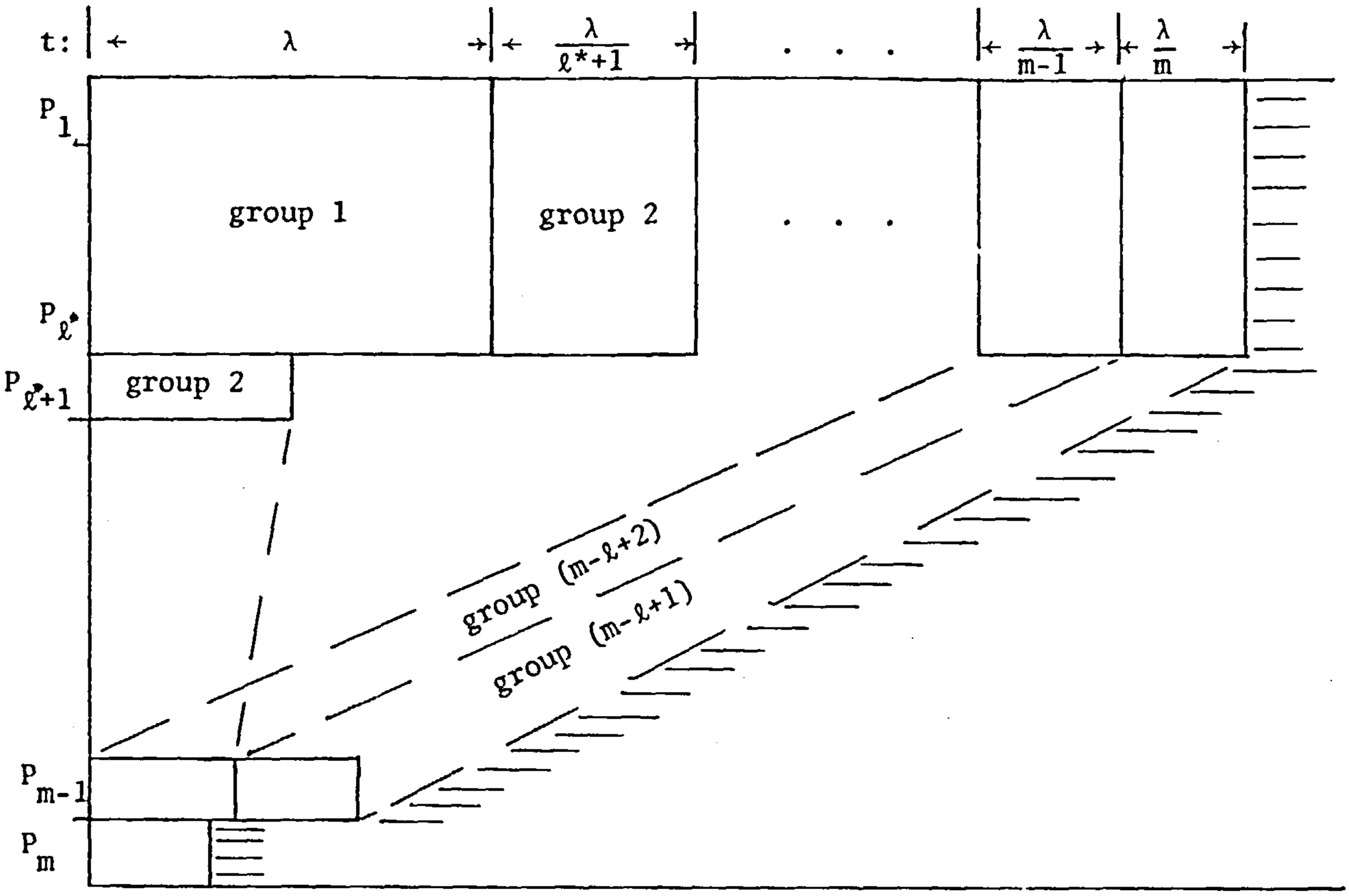


FIGURE 6.18: The general schedule produced by the Q.A.D. algorithm

$$\omega'_{OPT} \geq \frac{\sum_{i=1}^{m} U_i}{\lambda m}$$

$$I_i \leq m\frac{\lambda\omega'_{OPT}}{\bar{k}+1}\ ^{\dagger}, \qquad 1\leq i\leq m, \qquad i\neq x$$

and $$I_x = 0$$

---

†*This is so, because if* $\bar{k}$ *is the number of jobs allocated on* P_x *then, there should be at least* $\bar{k}$ *jobs on any processor,* $\sigma_w \leq \lambda\omega'_{OPT}/(\bar{k}+1)$ *(since no parts of a job can be run simultaneously on two or more processors and if* $\sigma_w \geq \lambda\omega'_{OPT}/\bar{k}$ *and hence* $\tau_w \geq \omega'_{OPT}/\bar{k}$ *then, because* $\sigma_j \geq \sigma_w$, or $\tau_j \geq \tau_w$ *for every job amongst the first w,* $\omega'_{OPT} \geq s_w + \tau_w/m \geq s_w + \sigma_w(\lambda m)$ *and* $s_w \geq \bar{k}\tau_w \geq \bar{k}\sigma_w/\lambda$, $\omega'_{OPT} \geq \omega'_{OPT} + \omega'_{OPT}/(\bar{k}m)$ *which is a contradiction) and* $I_i \leq m\sigma_w$ *(since if* $I_i > m\sigma_w$ *then,* $\omega'_{OPT} \geq s_w + I_i/(\lambda m)$ *or* $\omega'_{OPT} > k\tau_w + (m\lambda\omega'_{OPT})/[(\bar{k}+1)\lambda m] = \omega'_{OPT}$ *which is also a contradiction).*

we can obtain

$$\frac{\omega_{QAD}}{\omega_{OPT}} \lesssim \lambda\left(1 + \frac{m-1}{\bar{k}+1}\right) .$$

However, the above bound is true only if $k \gtrsim m-1$. In the case where $\bar{k} < m-1$, because,

$$I_i \lesssim \left(\frac{\bar{k}}{\bar{k}+1}\right)\lambda\omega'_{OPT} \quad , \; 1 \lesssim i \lesssim m, \quad i \neq x ,$$

following a similar analysis we eventually get

$$\frac{\omega_{QAD}}{\omega_{OPT}} < \lambda\left(1 + \frac{\bar{k}}{\bar{k}+1}\right) .$$

(ii) The same example, which was given in part (ii) of Theorem 6.2.5, can be used here as well to attain the lower limit of the bound when $1 \lesssim \ell^* \lesssim m-1$, provided that $(b_{max}/b_{min}) \lesssim (m-\ell^*)/(m-(\ell^*+1))$.

When $\ell^* = m$, we have

$$\omega'_{QAD} = U_i + I_i ,$$

$$\omega'_{OPT} \gtrsim \left(\sum_{i=1}^{m} U_i b_i\right) \Big/ \left(\sum_{i=1}^{m} b_i\right) ,$$

$$I_i \lesssim \left(\frac{\sum_{r=1}^{m} b_r}{b_{min}}\right) \times \left(\frac{\omega'_{OPT}}{\bar{k}+1}\right) \quad , \; 1 \lesssim i \lesssim m, \quad i \neq x$$

and $$I_x = 0.$$

Furthermore, since

$$\omega'_{QAD} = \frac{1}{m}\left[\frac{1}{b_{min}}\left(\sum_{i=1}^{m} U_i b_i\right) + \sum_{i=1}^{m}\left(\frac{b_{min}-b_i}{b_{min}}\right)(U_i+I_i) + \frac{1}{b_{min}}\left(\sum_{i=1}^{m} I_i b_i\right)\right],$$

using the above set of relationships we can verify that

$$\frac{\omega_{QAD}}{\omega_{OPT}} \lesssim 1 + \left(\frac{1}{\bar{k}+1}\right)\left(\frac{\sum_{i=1}^{m-1} b_i}{b_{min}}\right) \quad , \text{ for } \bar{k} \gtrsim m-1.$$

When $\bar{k} < m-1$, because

$$I_i \lesssim \left(\frac{\sum_{r=1}^{\bar{k}} b_r}{b_{min}}\right) \times \left(\frac{\omega_{OPT}}{\bar{k}+1}\right) \quad , \; 1 \lesssim i \lesssim m, \; i \neq x$$

following the same argument we obtain,

$$\frac{\omega_{QAD}}{\omega_{OPT}} < 1 + \frac{1}{(\bar{k}+1)} \left(\frac{\sum_{i=1}^{\bar{k}} b_i}{b_{min}} \right)$$

which completes the theorem.

Therefore, when the priority list is in the LTF_{MIN} or LTF_{MAX} ordering, the Q.A.D. algorithm has guaranteed performance levels slightly worse than the corresponding ones of the P.D. algorithm. Nevertheless, their behaviour appears identical. Further, although the bound increases logarithmically without any limit as m increases and ℓ^* is constant, it is better than the one given in the previous theorems. Finally, we should notice that when $\lambda=1$ or $b_i=b_{i+1}$, $1 \leq i \leq m-1$, the bounds given in all the theorems of this section agree with the corresponding ones found for the homogeneous multiprocessor system with independent memories (see Appendix IV).

CHAPTER 7

PROBABILISTIC ANALYSIS OF HEURISTIC SCHEDULING ALGORITHMS - A SIMULATION STUDY

7.1 INTRODUCTION

In the last two chapters a variety of heuristic scheduling algorithms were evaluated by their worst-case performance when the mean flow or the completion time is chosen as the performance criterion. In Chapter 1, it was indicated that this method of analysis is applicable to systems where a guaranteed level of performance must be provided. Such systems arise when we are dealing with critical real-time events. A contrasting kind of analysis evaluates the performance of an algorithm by its expected (average) behaviour. In systems where there are no critical deadlines, the average performance of an algorithm is more meaningful than the worst-case bounds. Actually, this is the main purpose of the present chapter (i.e. to evaluate the average performance of the algorithms which have been examined previously, under each one of the performance criteria). The average performance of an algorithm will be approximated by simulating the model of computation and evaluating the algorithm for different task systems using statistical analysis.

Therefore, having in mind the worst-case as well as the average behaviour of the considered algorithms we will be able to find out the degree of correlation between these two performance measures for the model under investigation. Such information will be a promising or not indication that the deterministic scheduling analysis can be a supplement tool, in the way described in Chapter 1, for systems where the average performance is more meaningful than the worst-case bounds.

7.2 SIMULATION PRELIMINARIES

In this study, we shall simulate a more realistic heterogeneous multiprocessor system with independent memories than the one described in Section 4.2. So, rather than assume that the jobs' time requirements are completely arbitrary and hence no processing speed relations exist between the processors, it is more down to earth to consider the existence of classes of jobs for which different processing speed relations may exist between the processors. In fact, this is a realistic approach since, we can roughly determine in advance factors of speed difference among the processors, as a measure of their relative power, when a particular job class is considered. However, if we impose some jobs to run on different processors we could obtain sufficient additional information so that our initial estimations on processors speed difference can be refined and improved.

More exactly, we have written a computer program[†] which simulates the heterogeneous multiprocessor model with independent memories where:

(i) the ranking of the processors, according to their processing speeds, varies from one job class to another;

(ii) the processing speeds differ uniformly in each job class, i.e., there is a constant percentage p of difference between them; and

(iii) p may vary for different job classes.

Further, a number of parameters, which define the simulation experiments, have to be chosen. These parameters describe:

(i) the processing system (i.e., number of processors, distribution function governing the selection of processors' ranking, size of the processors speed and size of private memories);

[†]*The computer program was written in FORTRAN and run on the CDC 7600 machine based at the Manchester Regional Computer Centre.*

(ii) the characteristics of a randomly generated task system (i.e., number of jobs and number of classes in the task system, distribution functions governing the selection of time and memory requirements of a job as well as the selection of its class).

Once these parameters are determined a number of trials are made for each experiment. In each trial a random task system is generated according to the parameters of the experiment. The jobs of the task system, being ordered by each of the considered sequencing rules (i.e. RAND, LMF, STF_{MIN}, STF_{MAX}, LTF_{MIN}, LTF_{MAX}, $LMLT_{MIN}$, $LMLT_{MAX}$, $LMST_{MIN}$ and $LMST_{MAX}$), are scheduled on the computation model illustrated by the defined processing system in the experiment, according to the demand scheduling algorithms examined in this thesis (i.e., P.D., P.D.*, Q.A.D., Q.A.D.*). Then, the mean flow as well as the final completion time of the schedules produced by each algorithm under each ordering rule are calculated. The optimal mean flow time of the task system is evaluated using Bruno's algorithm [Br1]. On the other hand, since the problem of scheduling on the computation model under investigation is a NP-complete one, it is impractical to determine the optimal completion time for an arbitrary task system. So, a lower optimal completion time is estimated for each task system according to the method given in Appendix V. At the end of each trial, statistics are collected on the ratio of the mean flow or completion time of each algorithm under every ordering rule as compared to their corresponding optimal or lower optimal calculated values respectively. As successive trials are performed the mean of each statistic should converge to the expected mean flow or completion time ratio of the corresponding algorithm for the given probability distributions governing the memory and time requirements of the jobs. In order to determine the number of trials which must be made to obtain meaningful results, the confidence interval technique of mathematical

statistics is used. So, in each trial a 95 percent (95%) confidence interval is calculated for the mean of each statistic. In statistical terms, this means that the probability of the true mean of the statistic (i.e. expected performance of the algorithm) lying within the calculated confidence interval is 0.95. The trials, which must not be less than 40 in order to satisfy the statistical terms for meaningful results, proceed until the length L of the confidence interval of the mean of the statistics becomes $L \leq 0.05$. (For details, see [Kre] pp.168-193). However, due to computation time constraints, we do not allow more than 400 trials in each experiment. This means that in some cases L might have greater value than 0.05. Two algorithms will be ranked according to their computed average performance only when their respective 95 percent (95%) confidence intervals are non-overlapping. In case the confidence intervals do overlap, the algorithms will be considered indistinguishable.

7.3 SIMULATION RESULTS

We recall that the purpose of the experiments in this study is to assemble a reasonable compelling background so that, we could examine the degree of correlation between the worst-case and the average performance.

The results of the experiments of the first test are presented in graphical form in Fig.7.1-7.12. In these and succeeding graphics the confidence interval is not explicitly indicated. However, the average performance of the various algorithms, as far as the mean flow time and the completion time performance criteria are concerned, relative to the corresponding optimal performance is shown in these figures as the number of processors m is increasing from 2 to 8. In each trial, the number of jobs in the task system was n=50 while the number of job classes was nc=3. The exponential distribution was used for the selection of processors ranking according to their processing power. More exactly, a (m×nc) matrix was formed using random numbers generated from an exponential distribution. The numbers in each column of this matrix were sorted according to the STF rule and then a new matrix D_{ij}, $1 \leq i \leq m$, $1 \leq j \leq nc$ was created by assigning an index which the i^{th} number in the sorted list of column j had in the original matrix. Notice, that in order to rank the processors according to their processing power, when the computation model is expanded from r to k processors, $2 \leq r \leq k \leq m$, the random numbers generated for the model of r processors were used again, together with [(k-r)nc] new ones selected from the governing distribution. This is a realistic approach, since only the new introduced processor(s) is (are) ranked among the existed processors, when the computation model is expanded. The processing speeds of the processors for each class of jobs were obtained from the form:

$$b_{[D_{ij}]j} = 1 + c_j \times \frac{m-i+1}{m} \times p_j \ , \ 1 \leq i \leq m, \ 1 \leq j \leq nc, \qquad (7.1)$$

where p_j is the probability of speed difference between the processors,

which was obtained from a uniform distribution in the range [0,1], $(1+c_j)$ is a limit on the speeds of the processors for the j^{th} class of jobs and c_j was selected from a uniform distribution in the range [1,7]. Further, the size of the private memories was decided from the form:

$$|P_i| = B + \lfloor A \times \frac{m-i+1}{m} \times p^* \rfloor \ , \ 1 \leq i \leq m \ , \qquad (7.2)$$

where B=4 is the base value of the memory sizes, A=60, (A+B) is the extreme value of memory sizes, and p^*=0.5 is the percentage of memory differences between the processors. Finally, the jobs in the task systems were generated by selecting their time and memory requirements from uniform distributions in the range [0,100] and [4,(4+60×0.5)] respectively. Moreover, their class index was decided by $\lfloor a_j \rfloor$, where a_j is a random number selected from the uniform distribution in the range [1,(nc+1)].

Many conclusions can be drawn from the above set of experiments when we observe the results of the average performance of the previously discussed algorithms. First of all, as was expected, the average performance of any algorithm under any priority list is much better than the corresponding worst-case performance. However, the remaining observations and conclusions are given in two parts according to the chosen performance criterion.

(a) When the mean flow time is the performance criterion from the results given in Fig.7.1-Fig.7.7 we can observe:

(1) the average performance of the P.D. and Q.A.D. algorithms, under the RAND,LMF,LTF or LMLT and RAND,LMF,STF or LMST orderings respectively, is decreasing as the number of processors in the model increases (see Figs. 7.1 and 7.2).

(2) further, the performance of the P.D. and Q.A.D. algorithms, under the STF or LMST and LTF or LMLT orderings respectively, is increasing as the number of processors in the system increases (see Figs.7.1 and 7.2).

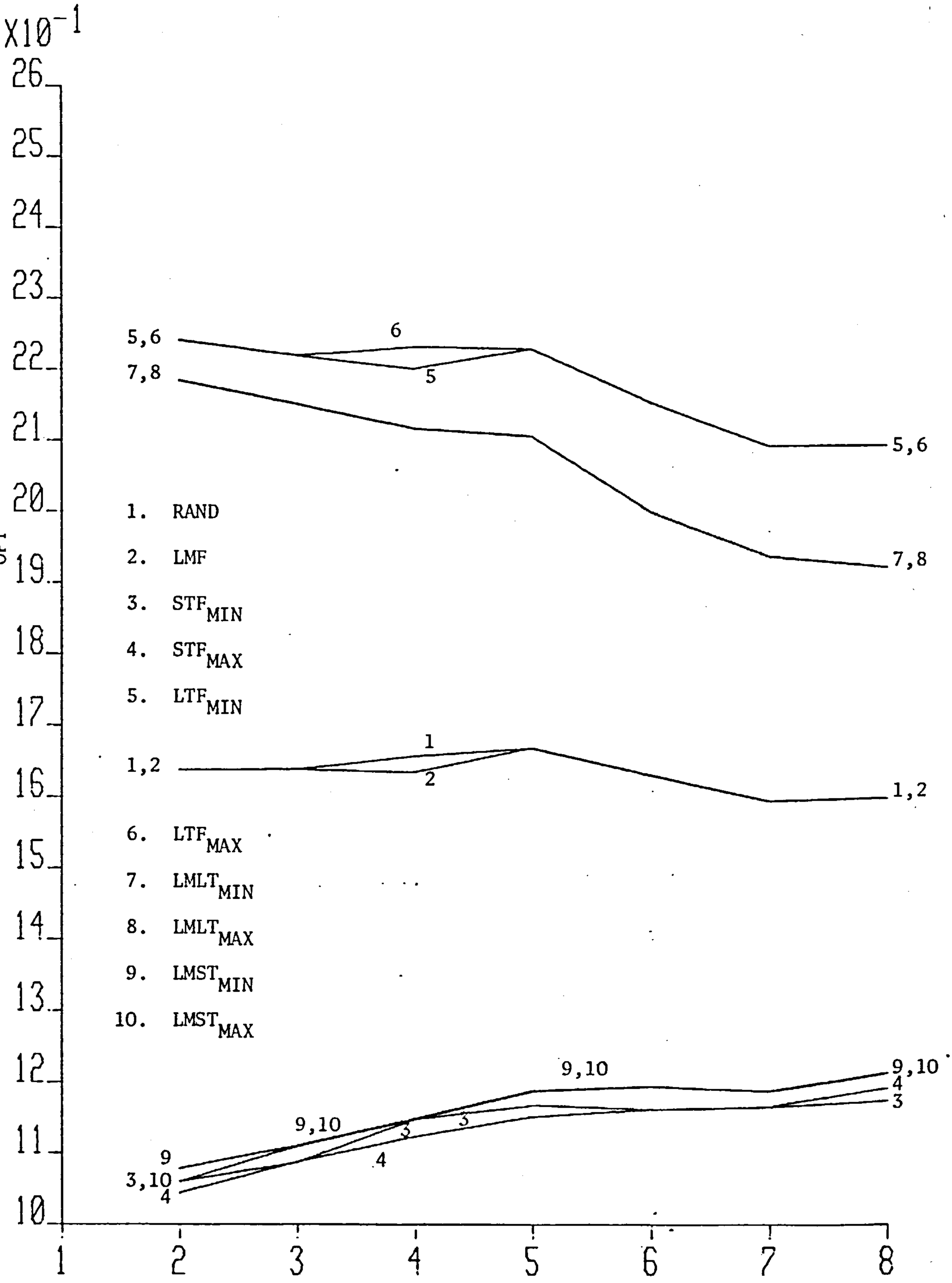


FIGURE 7.1: Test 1 - Mean flow time - P.D. algorithms

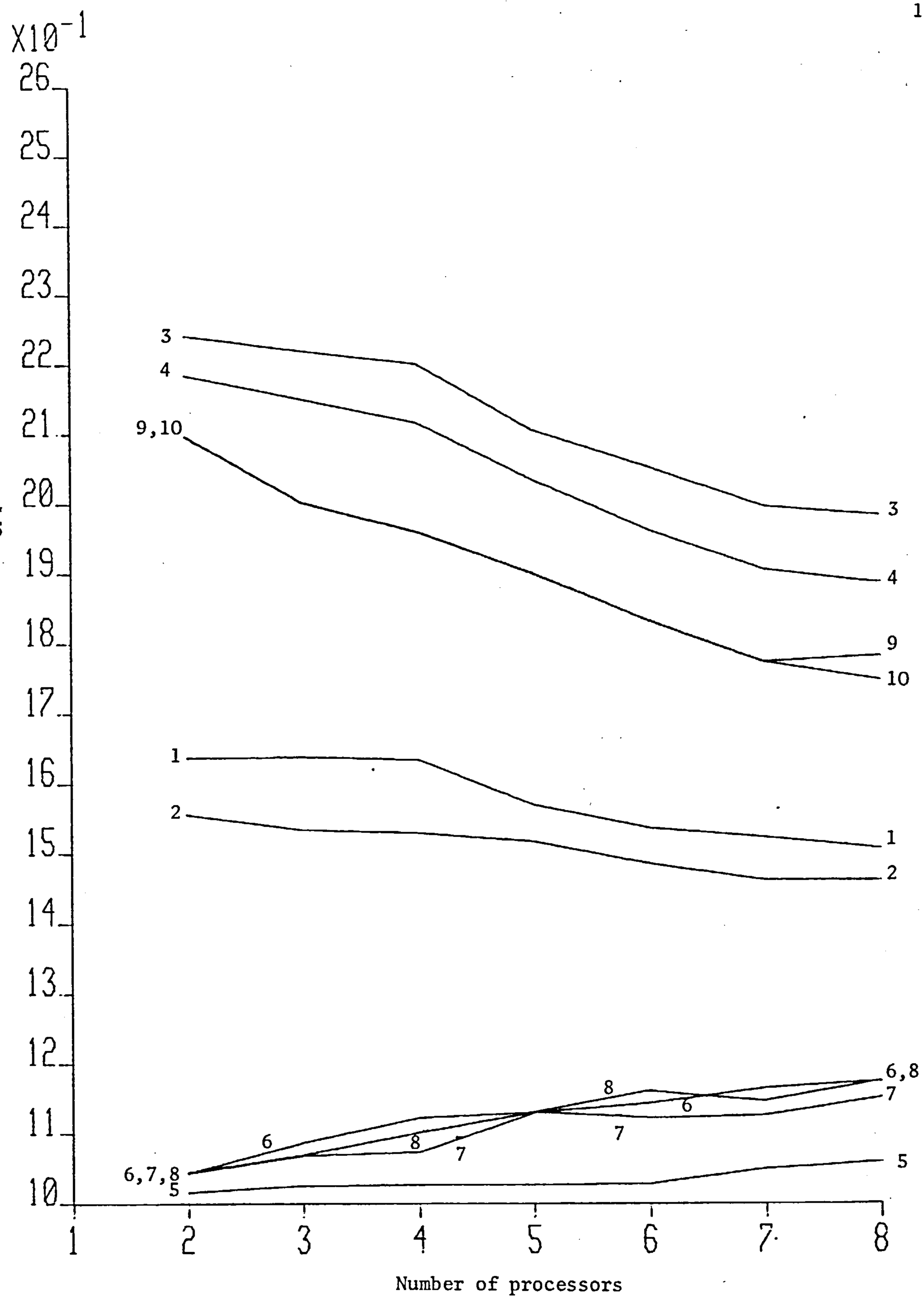


FIGURE 7.2: Test 1 - Mean flow time - Q.A.D. algorithms

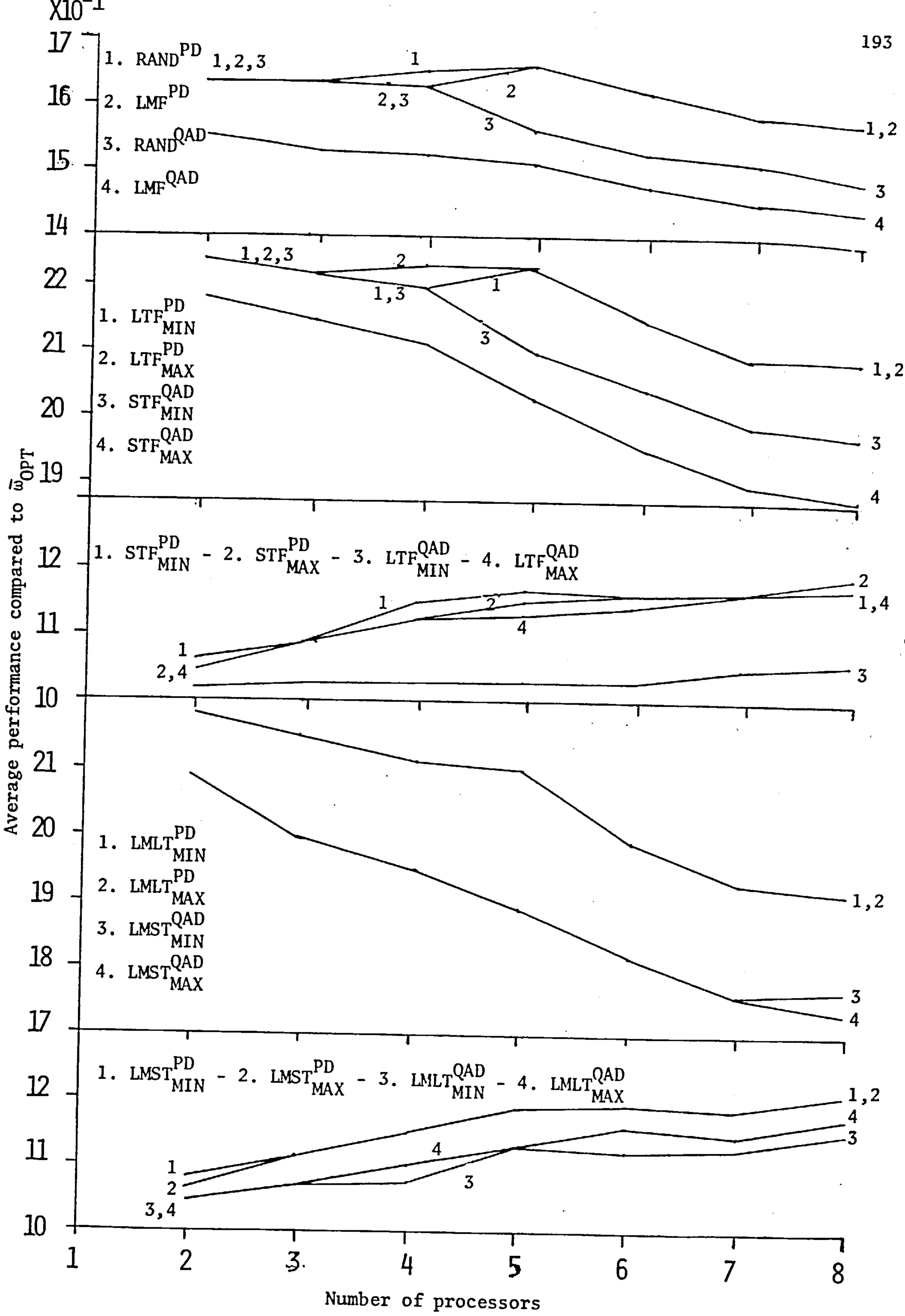


FIGURE 7.3: Test 1 - Mean flow time - Comparing P.D. and Q.A.D. algorithms

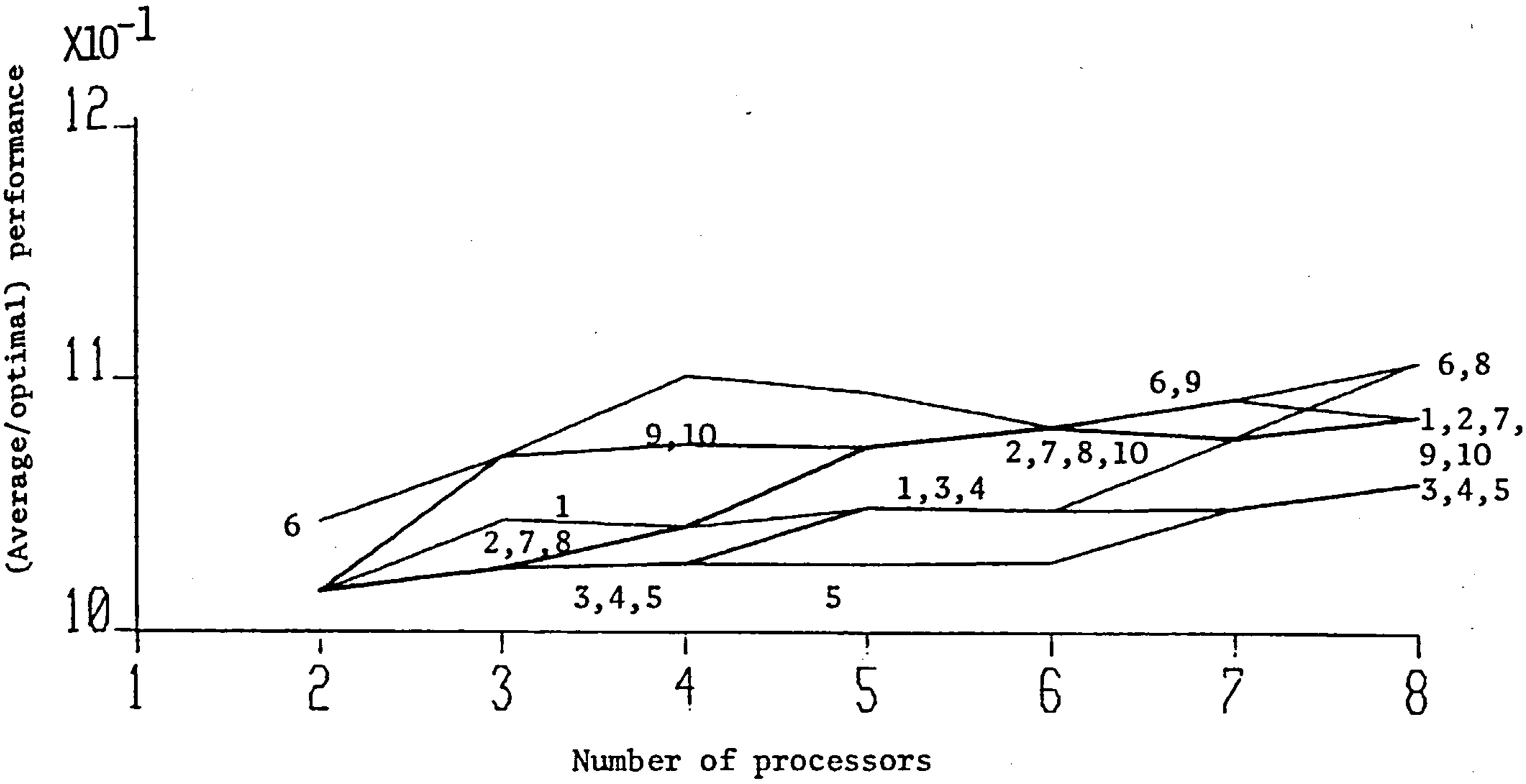


FIGURE 7.4: Test 1 - Mean flow time - Q.A.D.* algorithms

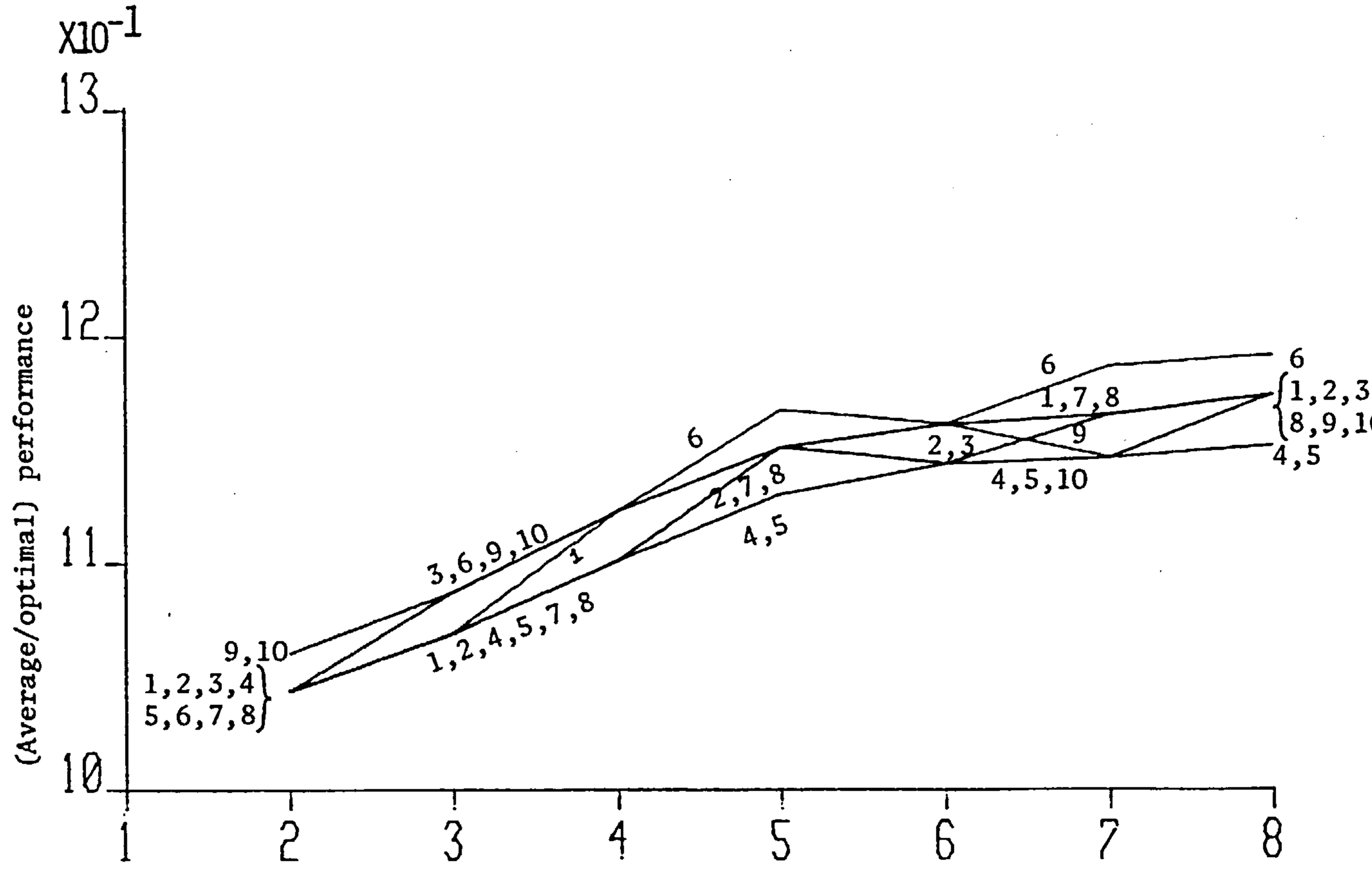


FIGURE 7.5: Test 1 - Mean flow time - P.D.* algorithms

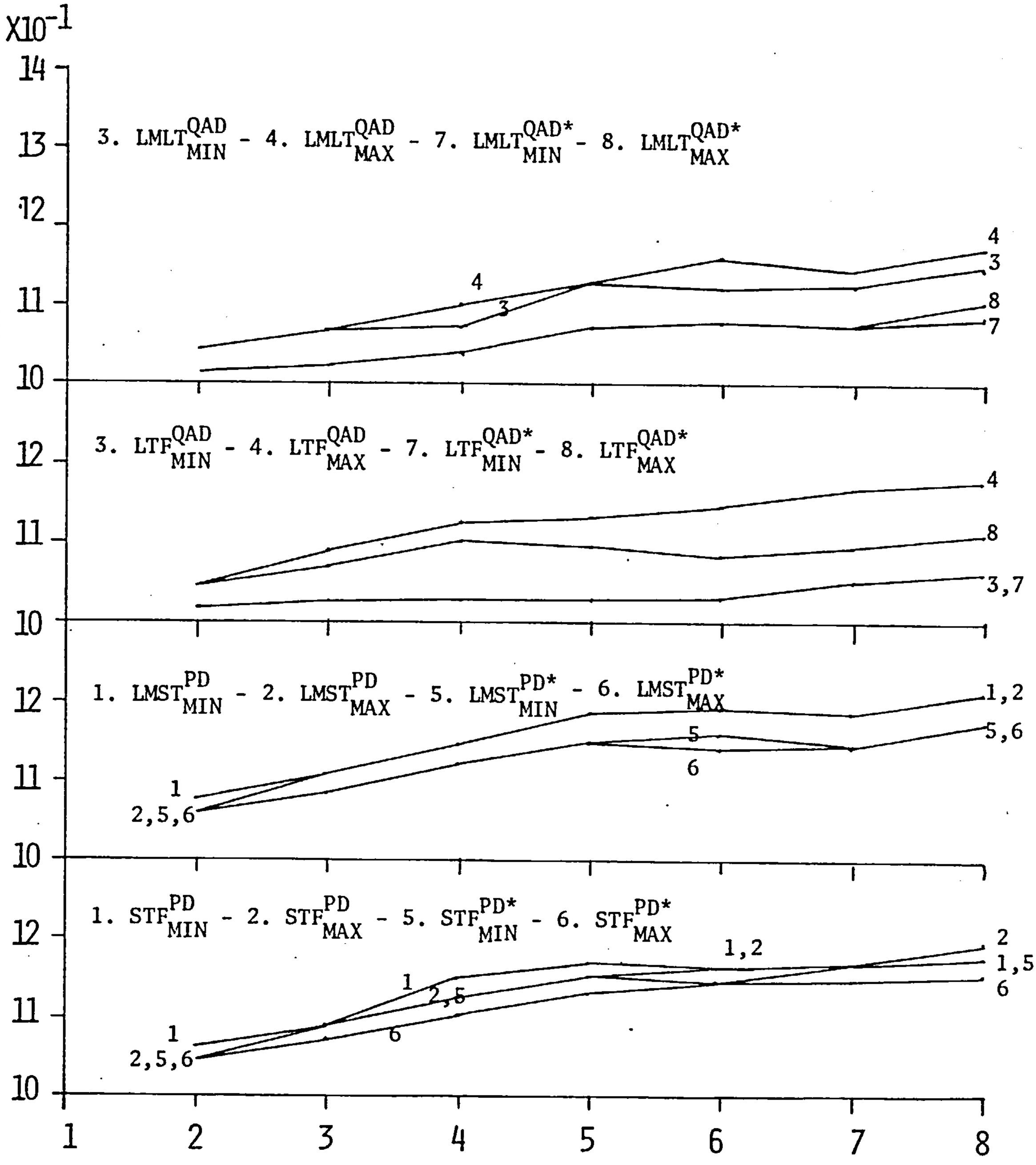


FIGURE 7.6: Test 1 - Mean flow time - Comparing some of the P.D. and Q.A.D. algorithms to their corresponding P.D.* and Q.A.D.* ones

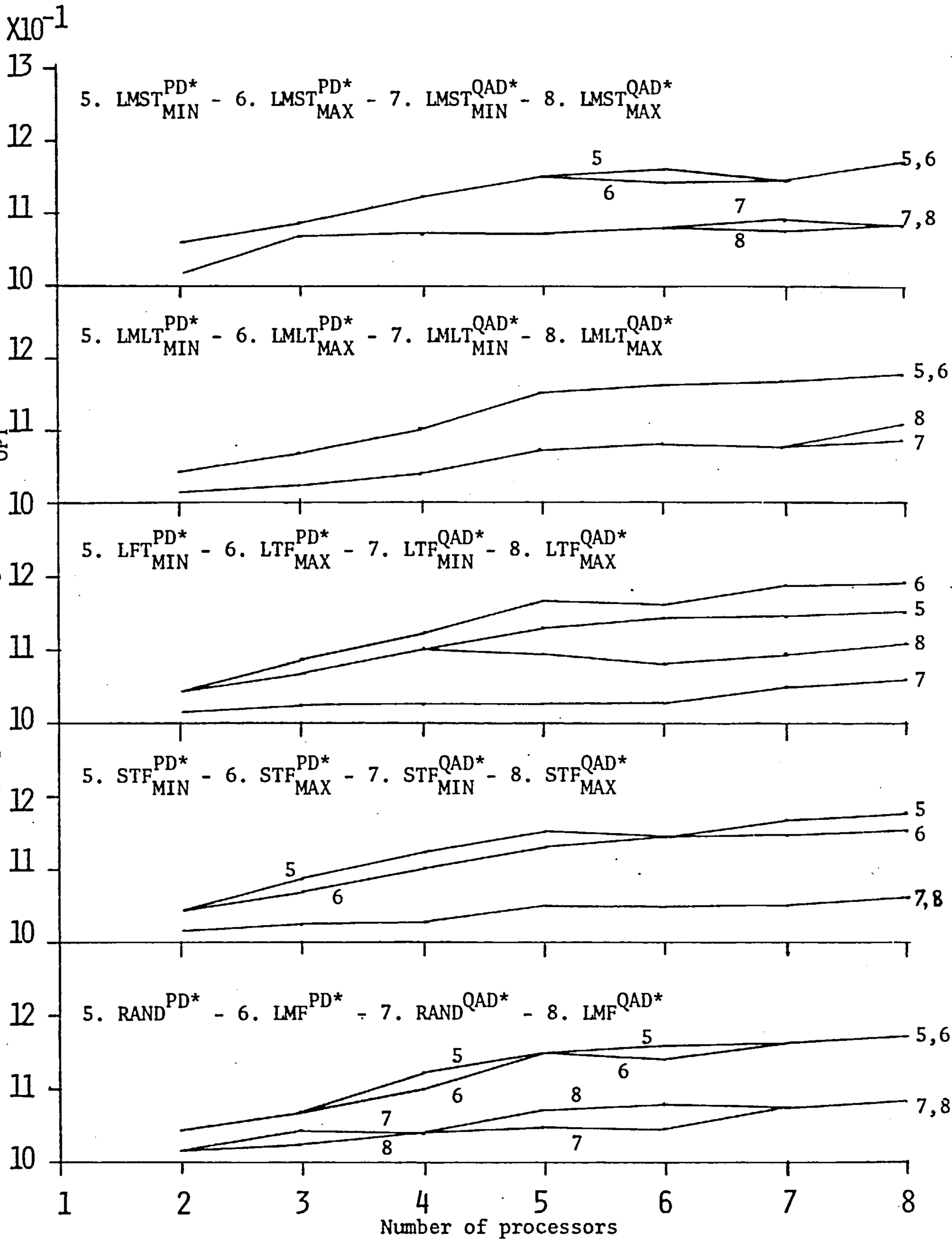


FIGURE 7.7: Test 1 - Mean flow time - Comparing P.D.* and Q.A.D.* algorithms

(3) the distances between the performance curves of the ordering rules mentioned in (1) and (2) are becoming less as the number of processors increases (see Fig.7.1 and 7.2);

(4) the performance of the P.D. algorithm under the STF_{MIN} and STF_{MAX} rule is very close to each other; moreover, the utilisation of memory in contributing to the STF ordering does not offer any improvement in the average performance (see Fig.7.1);

(5) the performance of the Q.A.D. algorithm under the LTF_{MIN} ordering is better than when the LTF_{MAX} ordering rule is used and the utilisation of memory worsens its performance; on the other hand, the utilisation of memory in contributing to the LMF_{MAX} rule offers a slight improvement for some instances (see Fig.7.2);

(6) the performance of the Q.A.D. algorithms is better for some instances or indistinguishable to the corresponding P.D. algorithms (see Fig.7.3);

(7) the P.D.* and Q.A.D.* algorithms do offer a great deal of improvement in average performance over the P.D. and Q.A.D. algorithms only when the priority list is in a RAND, LMF, LTF or LMLT and RAND, LMF, STF or LMST ordering respectively (compare the results given in Fig. 7.1,7.2 and 7.4,7.5); moreover, for the other ordering rules although the P.D.* and Q.A.D.* algorithms have better or indistinguishable average performance, the improvement is small (see Fig.7.6);

(8) the performance of the Q.A.D.* algorithm is always better than or indistinguishable to the P.D.* algorithm when identical priority lists are used (see Fig.7.7); and

(9) the performance of the Q.A.D.* algorithm under the considered ordering rules varies in a larger range than the P.D.* algorithm does for the corresponding ordering rules (see Fig.7.4 and 7.5);

in fact, the performance of the Q.A.D.* and P.D.* algorithms varies in an average of 5 and 2.5 percent (5% and 2.5%) respectively.

Nevertheless, the above mentioned observations show that the behaviour of the algorithms, based on the average performance, agrees in principle with the one predicted from the worst-case bounds in Chapter 5. Also, we should notice that ordering procedures with identical worst-case bounds may have significantly different expected performance. This is illustrated by the following ordering rules: RAND, LMF, LMLT and LTF in Fig.7.1 and RAND, LMF, LMST and STF in Fig.7.2. An explanation of this matter could be the fact that the performance of the P.D. and Q.A.D. algorithms, under the LTF or LMLT and STF or LMST rules respectively, is expected to be worse as compared to the corresponding performance of the algorithms under the RAND or LMF rules. Further, ordering rules with distinct, though close, bounds may possess expected performances which are indistinguishable by simulation techniques. This is illustrated by the results of the STF_{MIN}, STF_{MAX}, $LMST_{MIN}$ and $LMST_{MAX}$ ordering rules given in Fig.7.1, LTF_{MAX}, $LMLT_{MIN}$ and $LMLT_{MAX}$ in Fig.7.2 and by many ordering rules in Fig. 7.4 and 7.5. The Q.A.D. algorithm under the LTF_{MIN} ordering rule, which has the best worst-case bound (see Chapter 5), display very good performance characteristics. As it appears in Fig.7.2 the average difference between the performance of this algorithm and the optimal one is approximately 3.5 percent (3.5%). However, the average difference between the performance of the P.D. and Q.A.D. algorithms, under the STF, LMST and LTF_{MAX}, LMLT rules respectively, and the optimal one is approximately 14 and 12 percent (i.e., 14% and 12%). Furthermore, one can realise that in some cases, the performance of two ordering rules are ranked while their difference is less than 0.05 although we should expect them to be indistinguishable. This can be explained and it is not an anomaly. Actually, the trials of an experiment proceed until

all the confdience intervals of the statistics for each ordering rule and scheduling algorithm become less than 0.05 and hence, some of the ordering rules produce confidence intervals shorter than 0.05. Finally, when the mean flow time is chosen as the performance criterion and for the parameters used in this set of experiments, there is perfect agreement between the ranking of the expected and worst-case performance when the P.D. or Q.A.D. algorithm is used to construct the schedules. On the other hand, for P.D.* and Q.A.D.* algorithms we can say that there is a considerable agreement between the ranking for these two performance measures.

(b) When the final completion time is used as the performance criterion from the results given in Fig.7.8-7.12 we can observe:

(1) the average performance of the P.D. or the Q.A.D. algorithm under all the ordering procedures is generally increasing as the number of processors increases (see Fig.7.8 and 7.9);

(2) the utilisation of memory in contributing to the STF or LTF ordering offers a great deal of improvement when the P.D. algorithm is used (see Fig. 7.8);

(3) the P.D. algorithm under the LMLT ordering procedures have always had the best performance (see Fig.7.8); and

(4) the performance of the P.D. algorithm is always better than the one the Q.A.D. algorithm produces for all the ordering rules except the STF_{MIN} (see Fig.7.10-7.12).

All these observations agree with the behaviour of the algorithms predicted from the worst-case bounds in Chapter 6, except the anomaly mentioned in the 4^{th} observation. Moreover, we should notice that ordering rules with identical worst-case bounds may have different expected performances. This is illustrates by the RAND, STF_{MIN}, STF_{MAX} ordering rules, when the P.D. algorithm is used, and by the LTF_{MIN}, LTF_{MAX} orderings

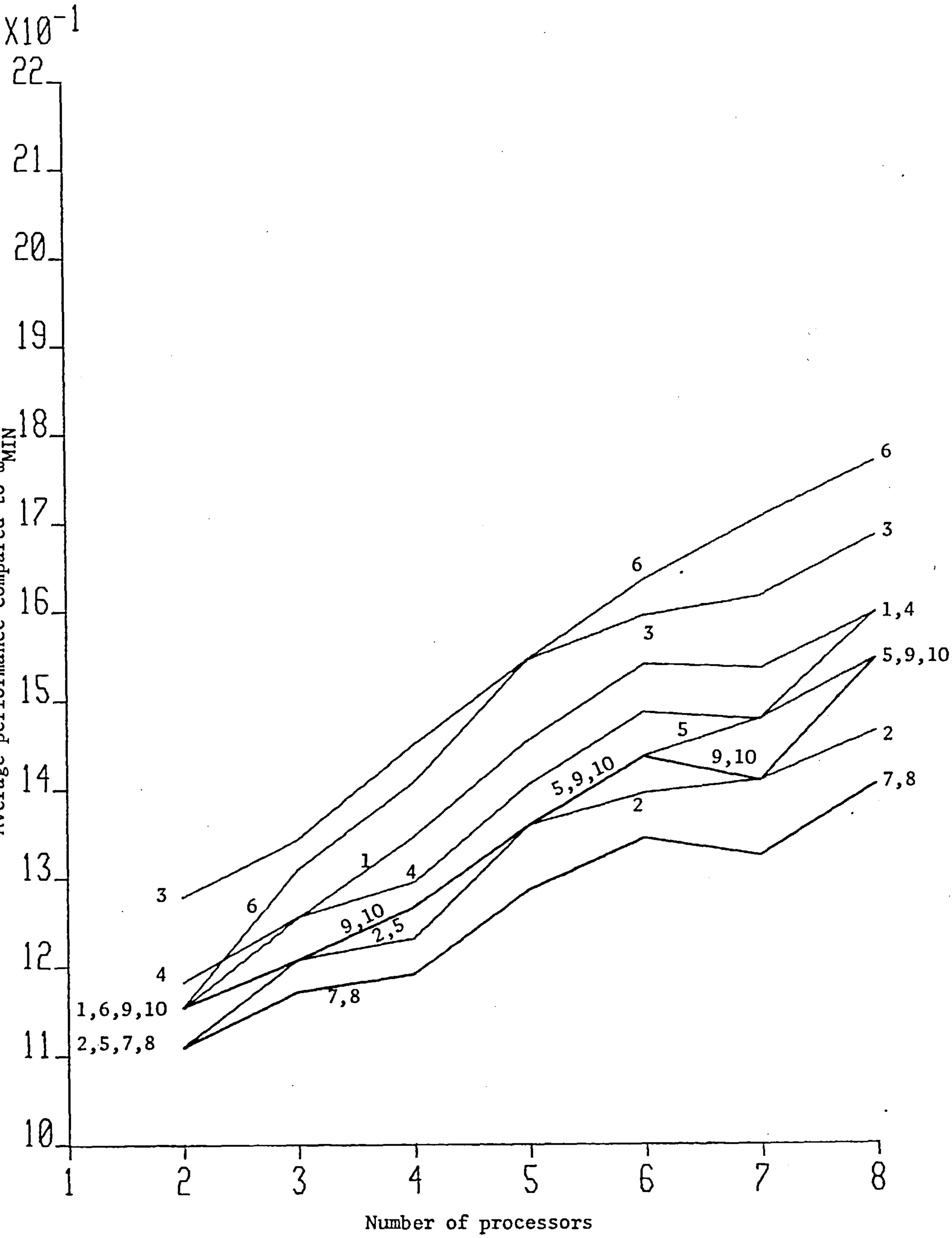


FIGURE 7.8: Test 1 - Completion time - P.D. algorithms

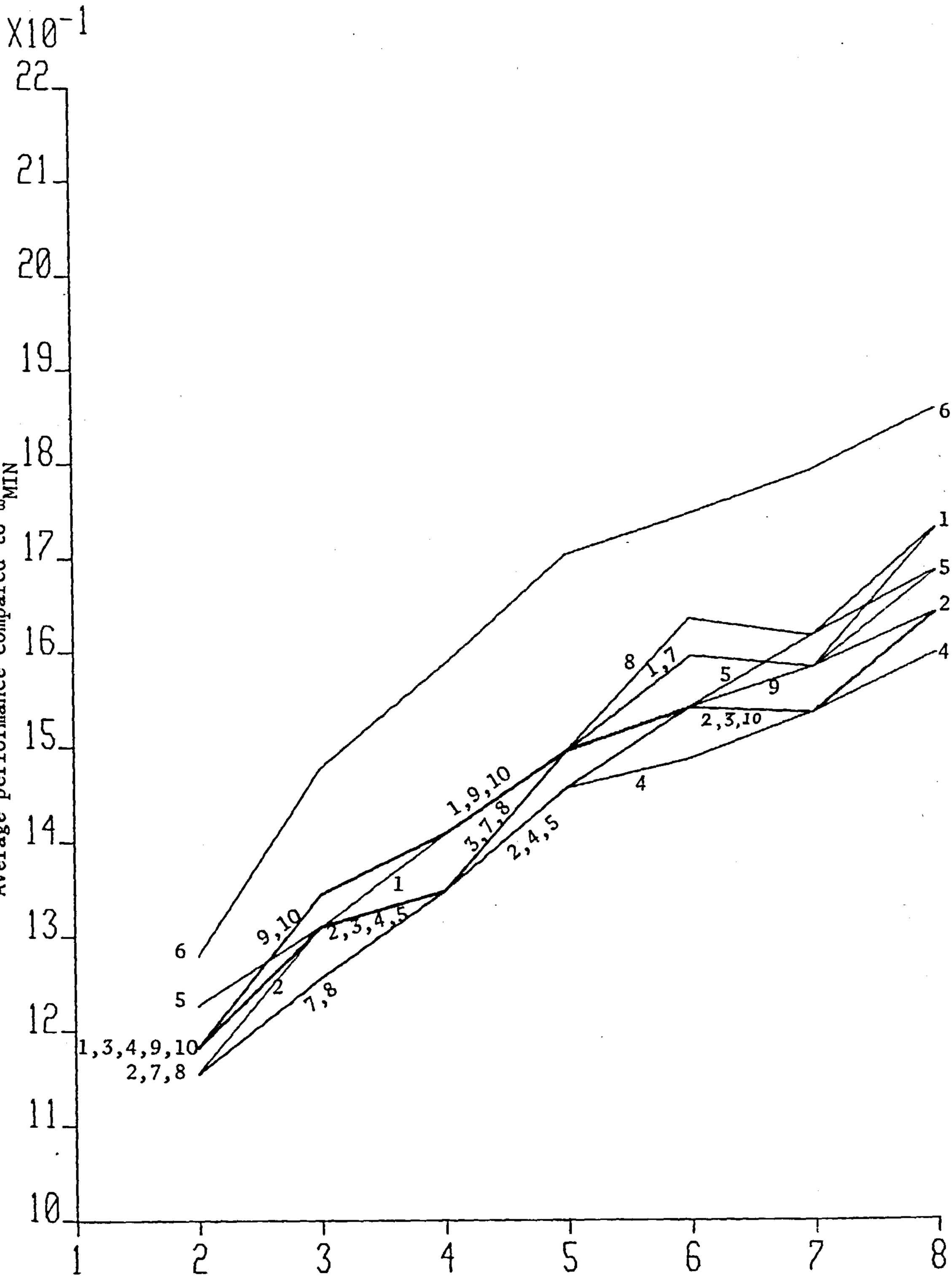


FIGURE 7.9: Test 1 - Completion time - Q.A.D. algorithms

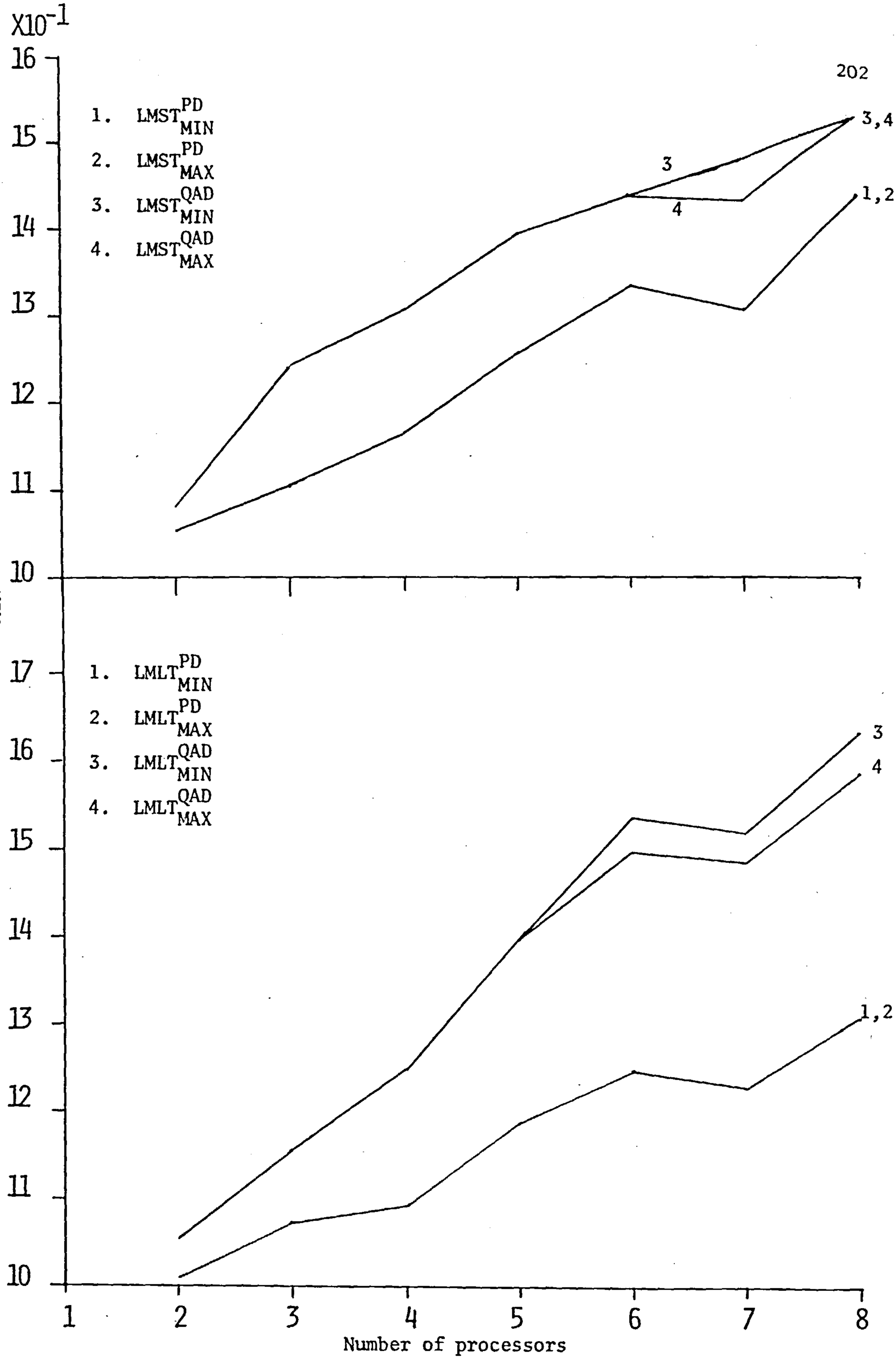


FIGURE 7.10: Test 1 - Completion time - Comparing P.D. and Q.A.D. algorithms

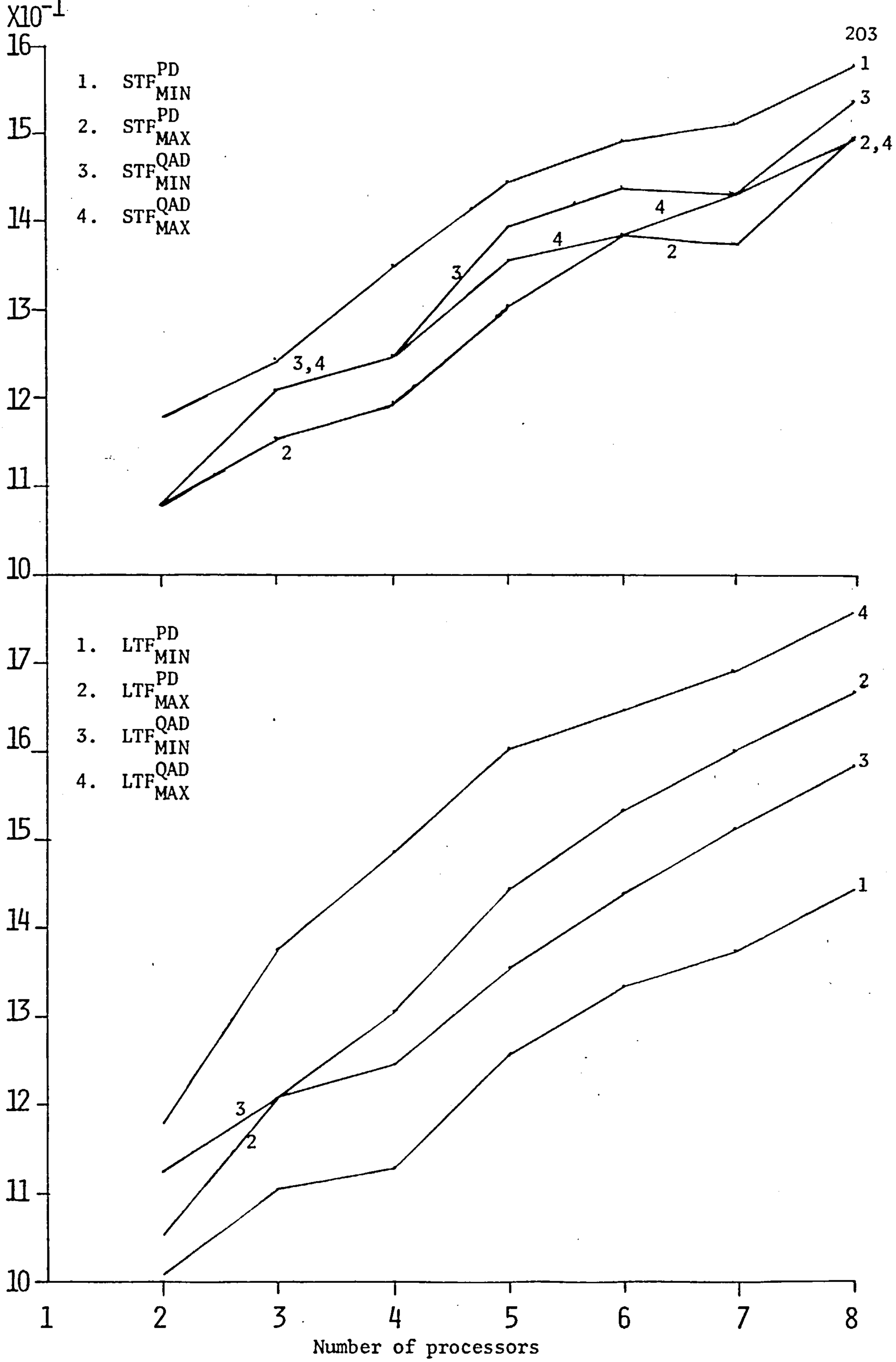


FIGURE 7.11: Test 1 - Completion time - Comparing P.D. and Q.A.D. algorithms

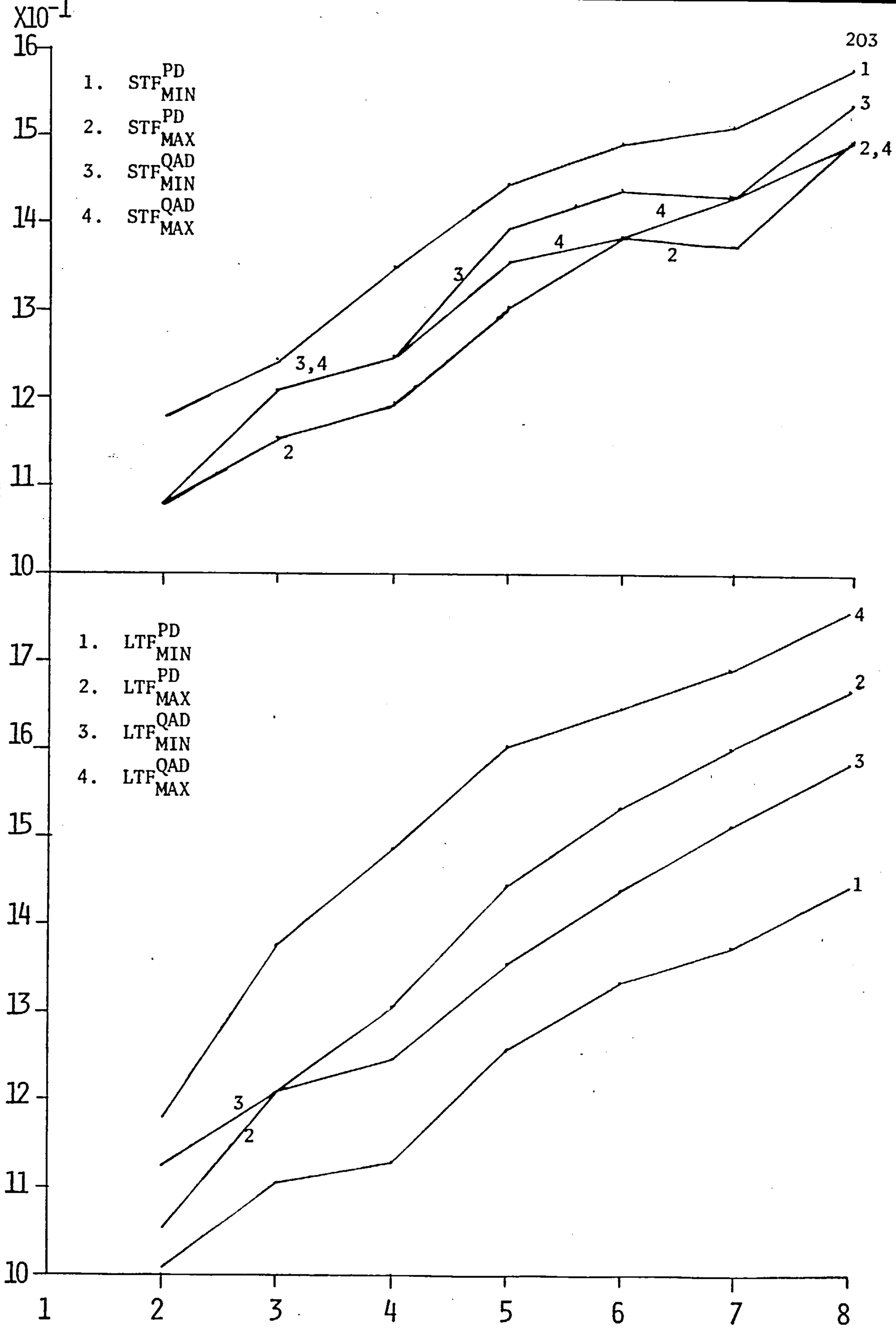


FIGURE 7.11: Test 1 - Completion time - Comparing P.D. and Q.A.D. algorithms

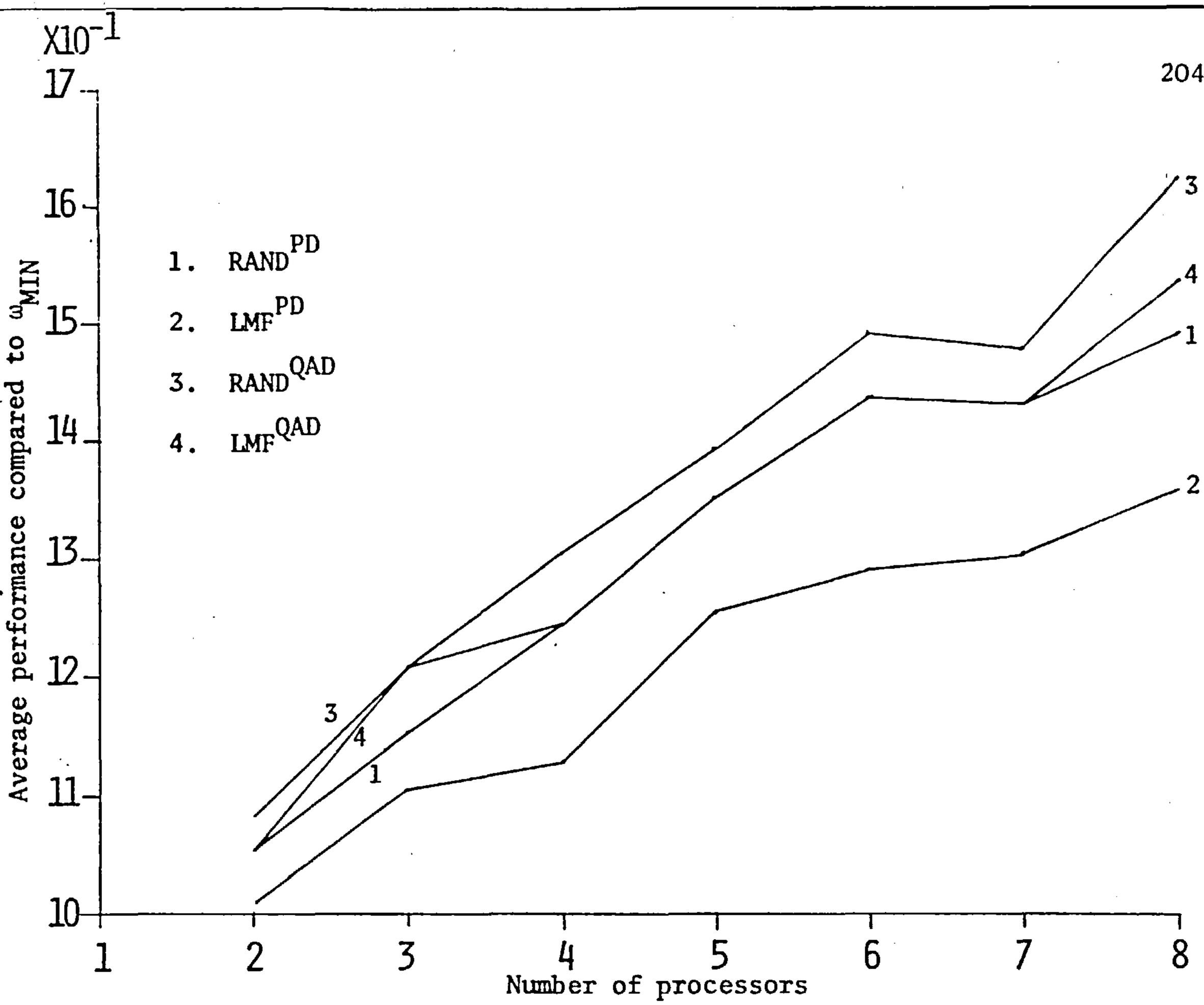


FIGURE 7.12: Test 1 - Completion time - Comparing P.D. and Q.A.D. algorithms

when the P.D. or Q.A.D. algorithm is used. Also, we should notice that both of the algorithms under any priority list may perform better than indicated in the Fig.7.9-7.12, since the estimated optimal value for the completion time is a lower bound on the true optimal. However, the most unexpected results is the performance of Q.A.D. algorithm under the LTF_{MAX} rule. This actually makes the ranking of the average performance, when the Q.A.D. algorithm is used, to disagree with the corresponding worst-case one. Apart from that, there is a perfect agreement between the rankings of average and worst-case performance for the P.D. algorithm and a considerable one for the Q.A.D. algorithm.

Although this test could not be regarded as substantial evidence for the behaviour of the average performance and hence further tests must be made,

this study is not intended to consider every possible combination or variation of the simulation parameters. Rather, fixed values or reasonable restrictions are placed on these parameters in order to narrow the number of tests which must be performed. The fixed value of a particular parameter was decided by making experiments similar to the former ones with different values for that parameter and choosing the one which gives the most reasonable results. Such experiments were carried out with the values of the various parameters as indicated below:

- the number of jobs in each trial to be 60 and 70;
- the number of classes in each task system to be 2 and 4;
- the distribution governing the ranking of the processors according to their processing power to be the normal and the uniform distribution;
- the percentage of difference between the speeds of the processors p_j to be decided from the uniform distribution in the ranges [0,0.25] and [0.75,1]; and finally
- the percentage of difference between the private memories to be $p^*=0.2$ and $p^*=0.9$.

It was found that increasing the number of jobs in each trial, decreasing or increasing the classes of jobs in the task system, using different distributions to decide the ranking of the processors and imposing the percentage of difference between the speeds or the memories to be small or large had negligible influence[†] in the ranking of the algorithms under the various ordering rules.

Now, since different values of the various parameters can not produce any significant changes in the results of the average performance of the algorithms, the values of these parameters can be considered reasonable and they will be used in the remaining simulation study. Also, this makes

---

[†]*By negligible influence we mean that the ranking is either identical to the first experiment or slight different, which does not damage the ranking of the algorithms when compared to the worst-case performance.*

the results of all the tests comparable. Moreover, our decision to consider models with 2 to 8 processors and their processing speed to differ up to 8 times is also reasonable, since most practical interest is concerned with systems of small numbers of processors and since it is difficult to find a mini- or micro- computer to be 8 times more powerful than another one, which is doing the same or similar functions. Further, an objection may be raised from the fact that the memory sizes differ in a regular manner, since such an arrangement does not allow those cases where there are significant variations in the differences between memory sizes, or where some memories are of the same size. However, the effect of these variations can be achieved with regular memory sizes by properly biasing the distribution function governing the jobs memory requirements.

In the remainder of this chapter we will study the results of the average performance of the algorithms for some additional tests. Actually, in these tests we use biased distribution functions for the time or memory requirements of the jobs. In detail, the various cases, which we consider, are summarised in Fig.7.13 using a tree structure. However, as can be seen, the number of tests are 9 and the first one, already completed corresponds to the path (1→2A→3A).

A uniform distribution of the job requirements may be represented as in Fig. 7.14(a). A uniformly distributed random variable y, in the range [0,1] is generated. Then, the reflection of y, in the line indicated, becomes a uniformly distributed random variable x in the range [0,100].

A biased distribution in the range [0,100] is shown in Fig.7,14(b). In this case the uniformly distributed random variable y is transformed into a biased random variable x when the reflection, in the line indicated, is performed.

Moreover, the biased distributions which will be used in the experiments of the following tests for the time and memory requirements are shown in Fig. 7.15 and 7.16 respectively.

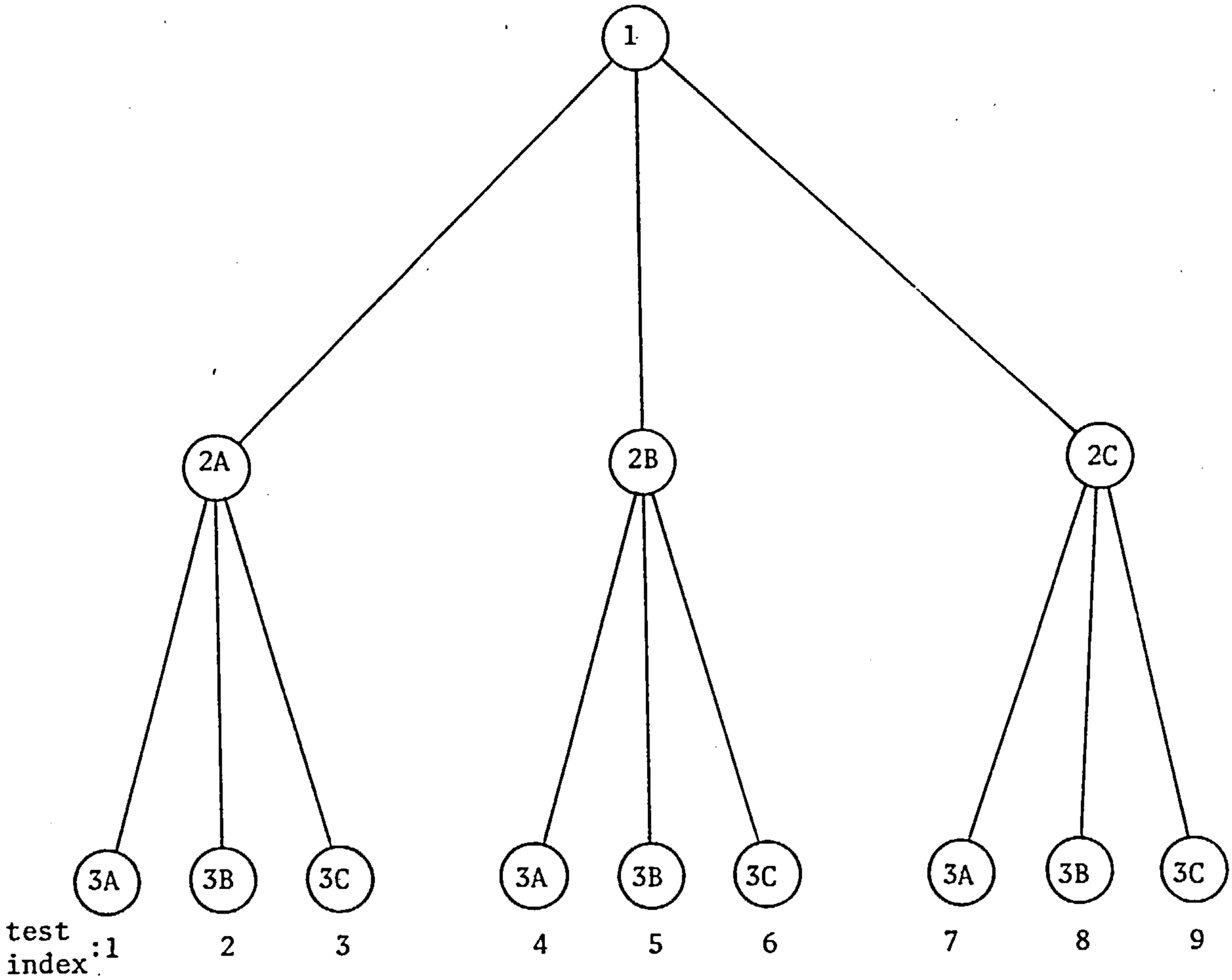


FIGURE 7.13: A tree to summarise the simulation study

Notations:

1 : a simulation study of the average performance for the considered algorithms;

2 : the distribution function of the memory requirements to be:

(A) uniform distribution;

(B) biased distribution favouring small memory requirements;

and (C) biased distribution favouring large memory requirements;

3 : the distribution function of the time requirements to be:

(A) uniform distribution;

(B) biased distribution favouring short time requirements;

and (C) biased distribution favouring long time requirements.

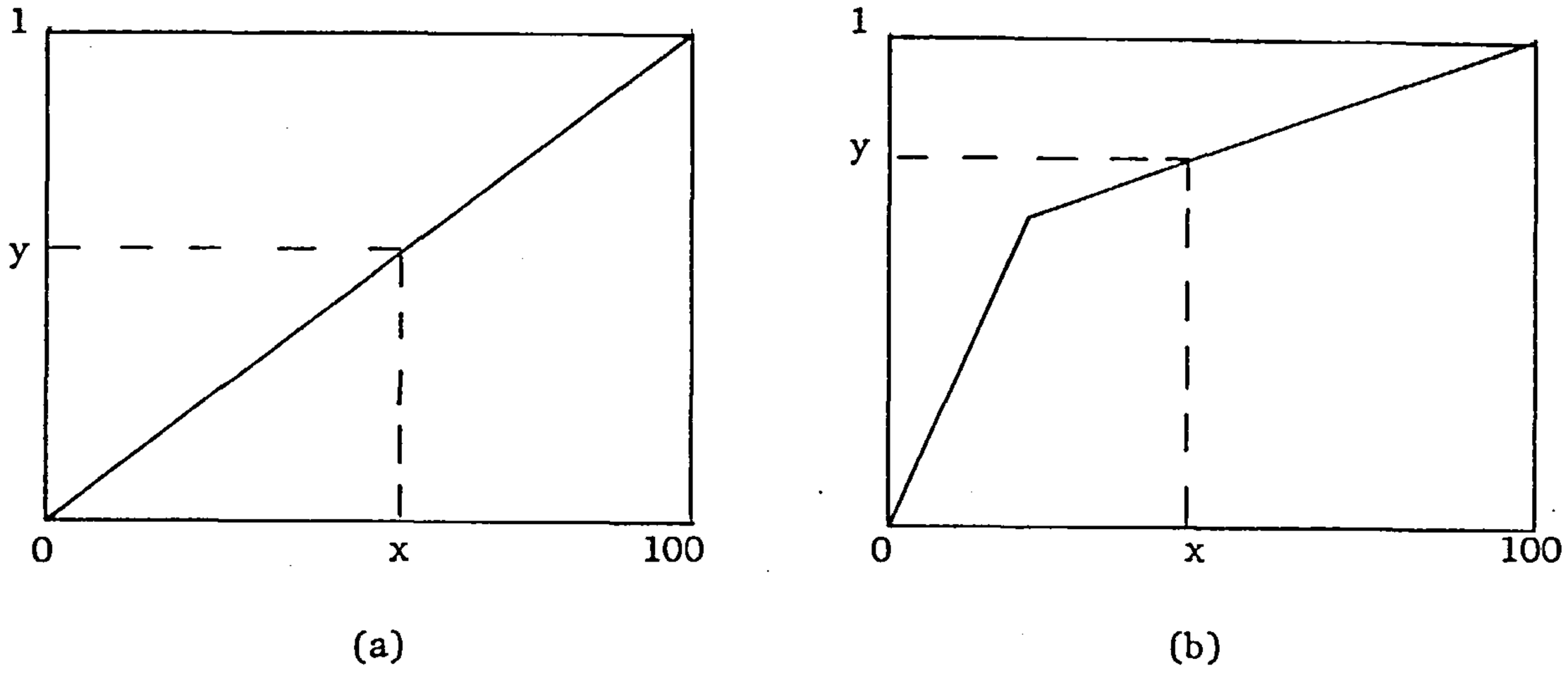


FIGURE 7.14: (a) A uniform distribution. (b) A biased distribution

First, we will show the results of the average performance of the algorithms as compared to the optimal one, when the mean flow time is used as the performance criterion. So, in Fig.7.17-7.20 the results of the 2nd test (i.e., the memory requirements of the jobs are selected from the uniform distribution and the time from the biased distribution favouring the small time requirement jobs) are given.

Generally, we can observe that the ratio of the average performance of the algorithms to the optimal one has been worsened in this test. Especially, for the LTF, LMLT, LMF, RAND and STF, LMST, LMF, RAND ordering rules when the P.D. and Q.A.D. algorithm is used respectively. Nevertheless, we should have expected such a behaviour, since the existence of few large jobs in the task system (i.e. 25%) and their allocation at the beginning of the schedule can only worsen the mean flow time. However, the ranking of the P.D. and Q.A.D. algorithms is exactly the same as test 1. On the other hand, although the ranking of the P.D.* and Q.A.D.* algorithms does not agree with the corresponding one of the 1st test, there is still a considerable agreement with the ranking of the worst-case bounds. Further, all the other observations made for the 1st test can be realised from the results of this test as well.

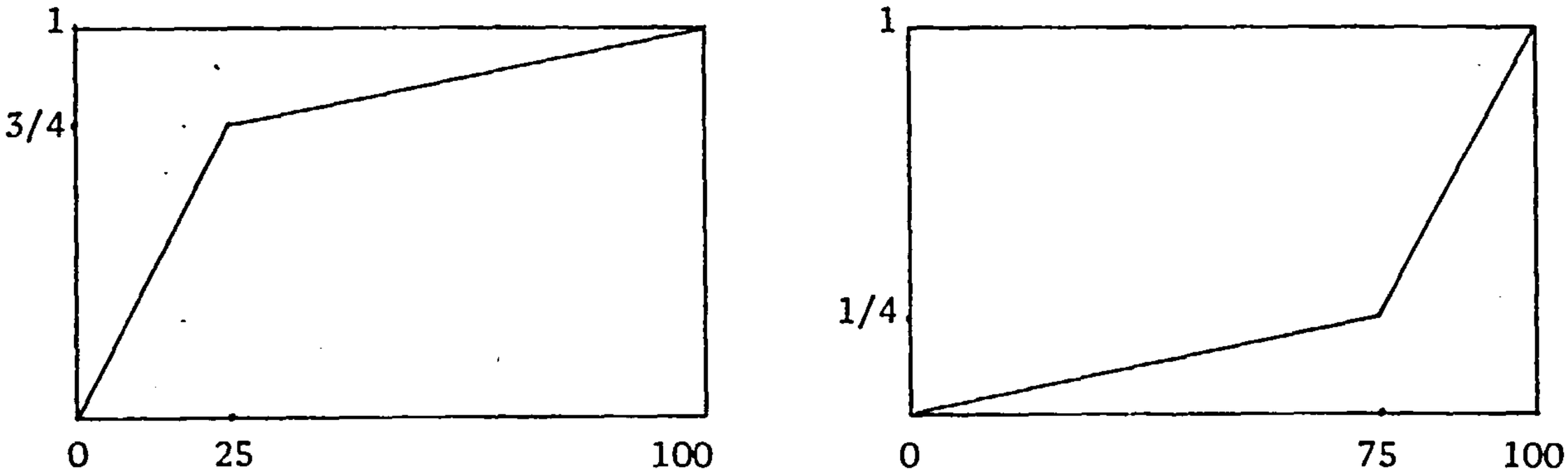


FIGURE 7.15: (a) A biased distribution favouring short time requirements
(b) A biased distribution favouring long time requirements.

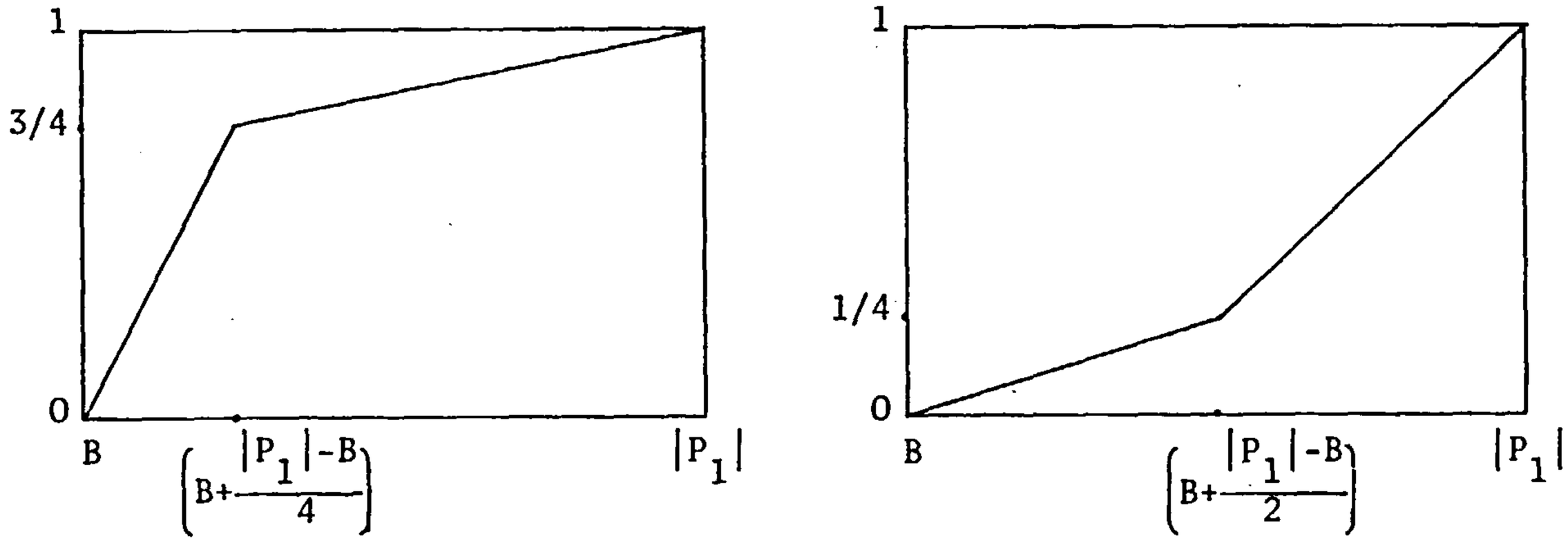


FIGURE 7.16: (a) A biased distribution favouring small memory requirements
(b) A biased distribution favouring large memory requirements.

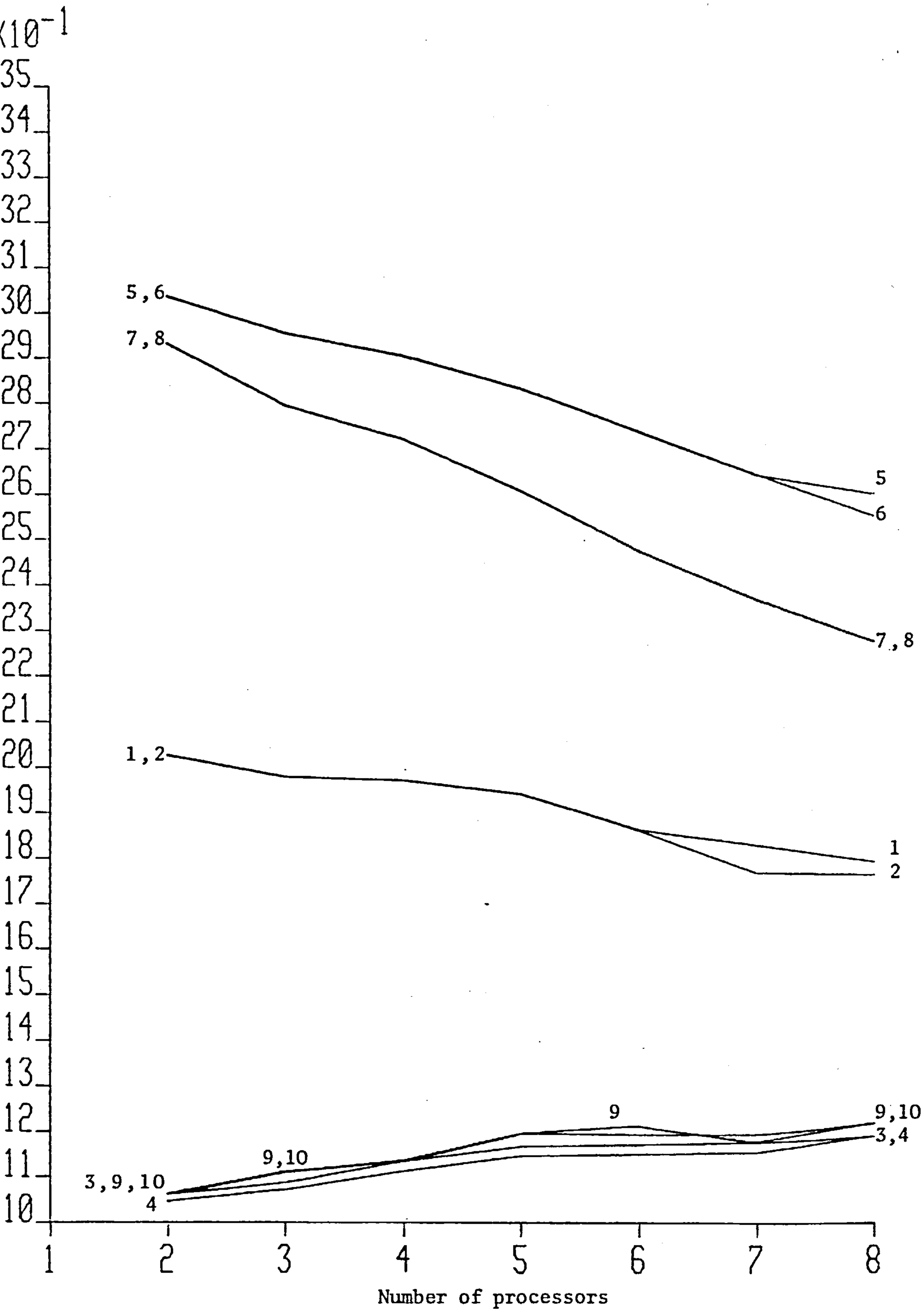


FIGURE 7.17: Test 2 - Mean flow time - P.D. algorithms

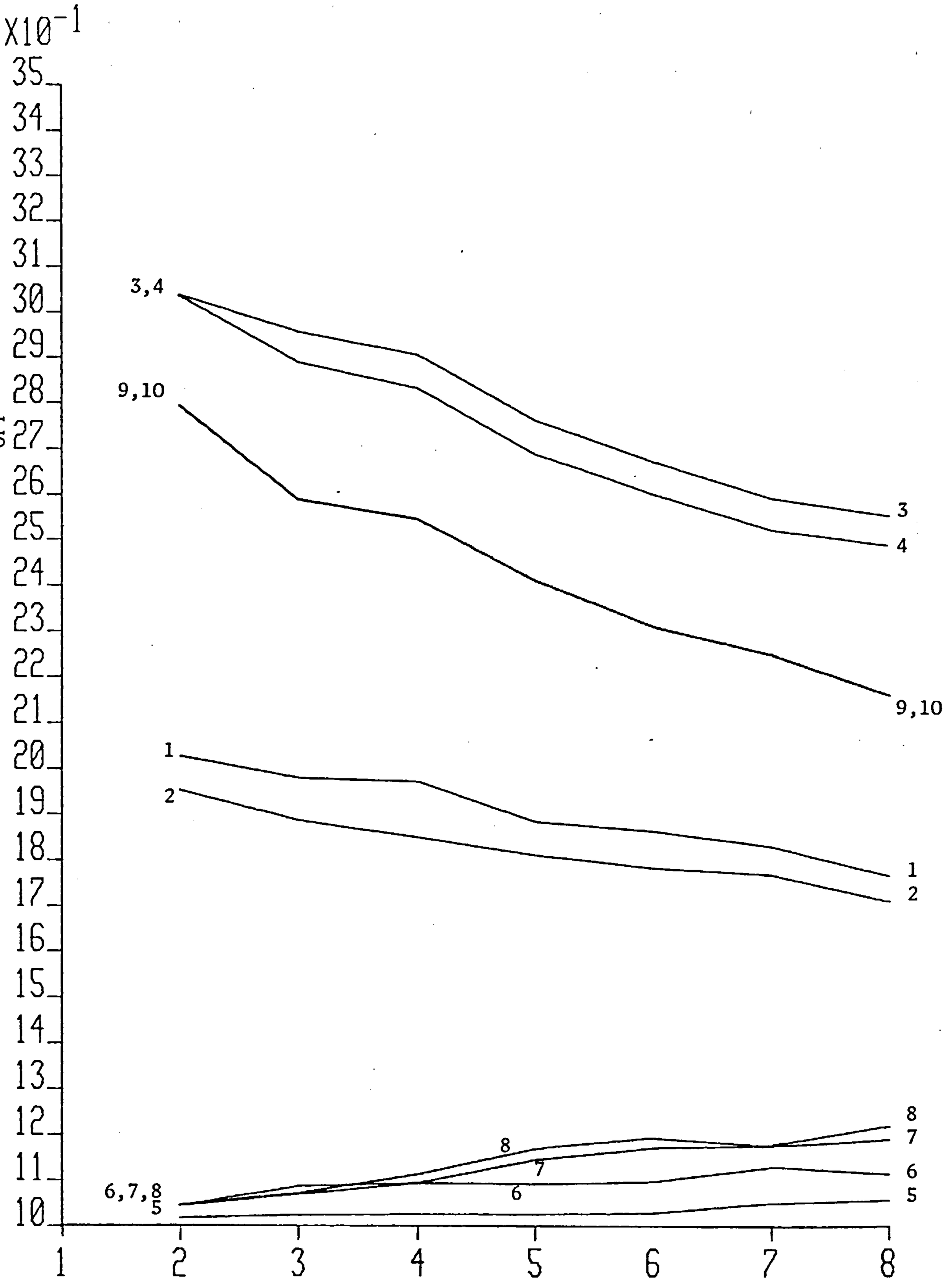


FIGURE 7.18: Test 2 - Mean flow time - Q.A.D. algorithms

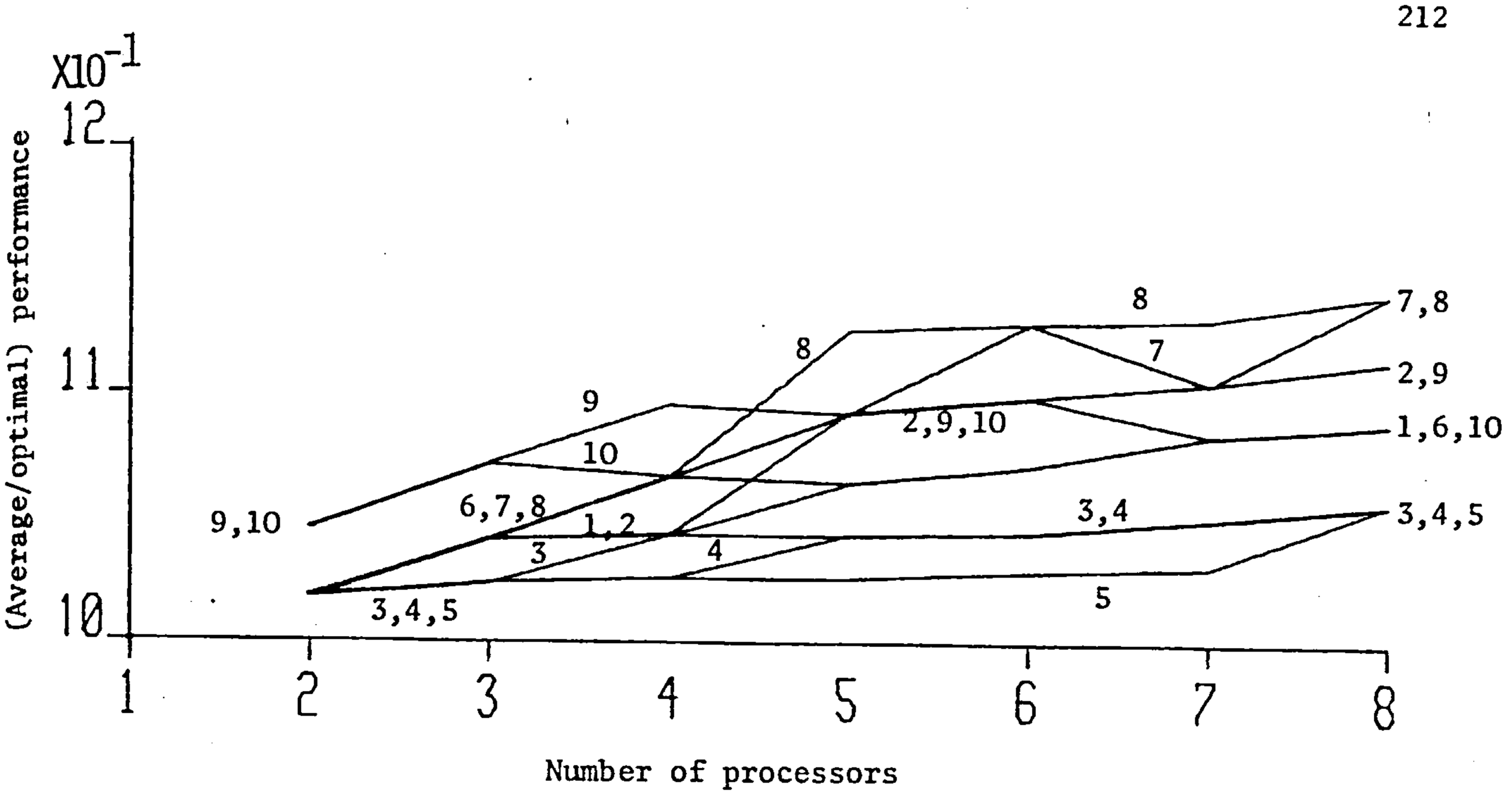


FIGURE 7.19: Test 2 - Mean flow time - Q.A.D.* algorithms

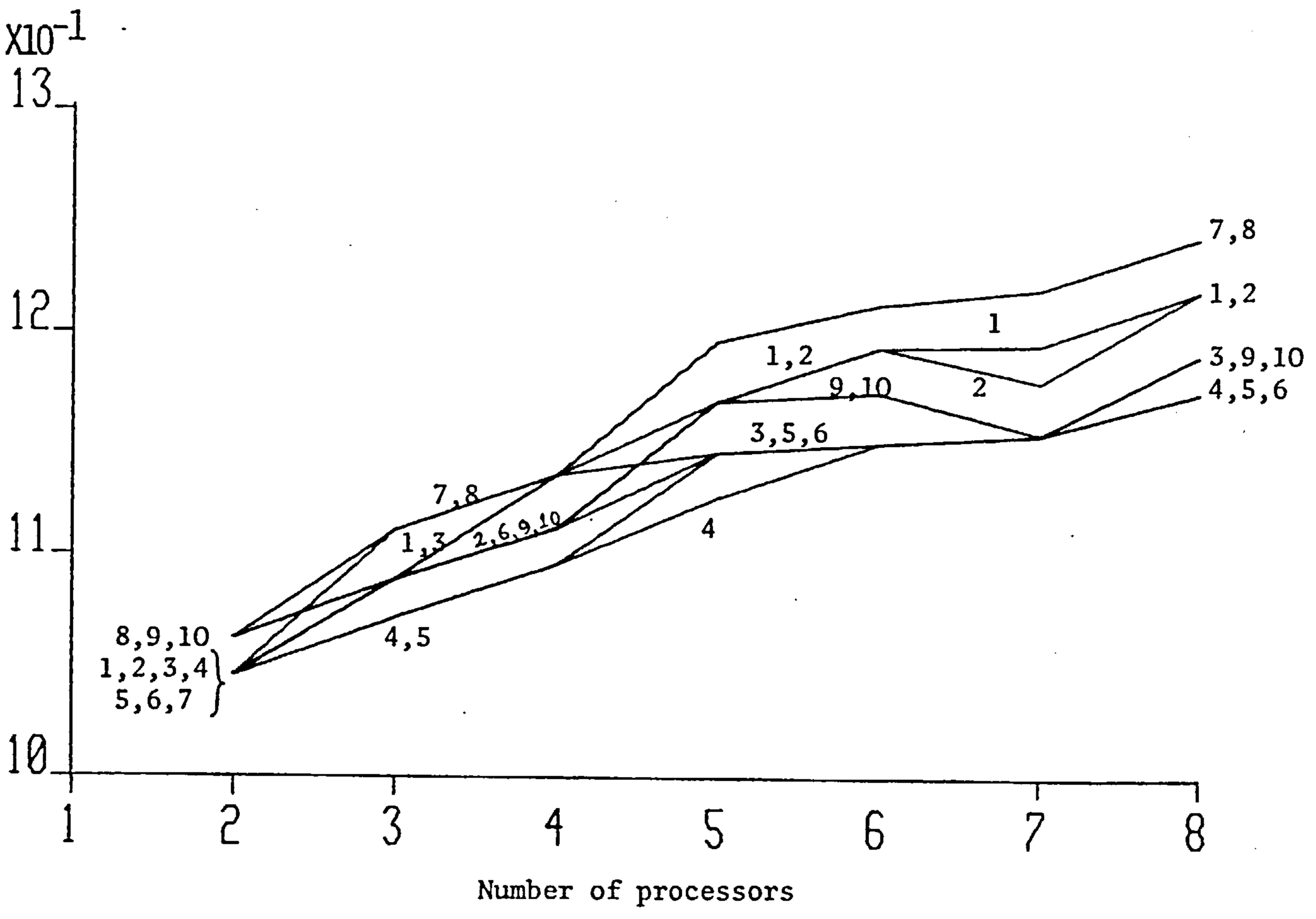


FIGURE 7.20: Test 2 - Mean flow time - P.D.* algorithms

Now, the results of another set of experiments, which complete the requirements of test 3, are shown in Fig.7.21-7.24. In contrast to the previous test, here the average performance of the P.D. and Q.A.D. algorithms, for the LTF, LMLT, LMF or RAND and STF, LMST, LMF or RAND ordering rules respectively, as compared to the optimal one is better than their corresponding ratio in the 1^{st} test. This is so, since most of the jobs (i.e., 75%) require execution time in the interval [75,100] and hence their arrangement at the beginning of the schedule cannot worsen the mean flow time as much as in test 1 or 2. Also, we can observe that the utilisation of memory in contributing to the LTF_{MAX} rule offers a better performance for the Q.A.D. algorithm. Although this observation contradicts the corresponding ones made in the previous tests, it does not damage the behaviour of the algorithms predicted by the worst-case bounds or the agreement of the rankings. Again, there is a considerable agreement between the ranking of the average and the corresponding worst-case performance for the P.D.* and Q.A.D.* algorithms.

The results of the set of experiments for test 4, where most of the jobs in the task systems require small memory, are given in Fig.7.25-7.28. Although, the ranking of the average performance of the algorithms differs slightly from the corresponding one which appeared in the previous tests, there is perfect agreement with the ranking of the worst-case bounds for the P.D. and Q.A.D. algorithms and a considerable one for the P.D.* and Q.A.D.* algorithms. Moreover, we can observe that:

- for the P.D. algorithm, the utilisation of memory in contributing to the STF ordering worsens the average performance significantly as compared to the previous tests;
- the LTF_{MAX} ordering rule gives a performance closer to the LTF_{MIN} ordering and always better than the $LMLT_{MAX}$ one, when the Q.A.D. algorithm is used.

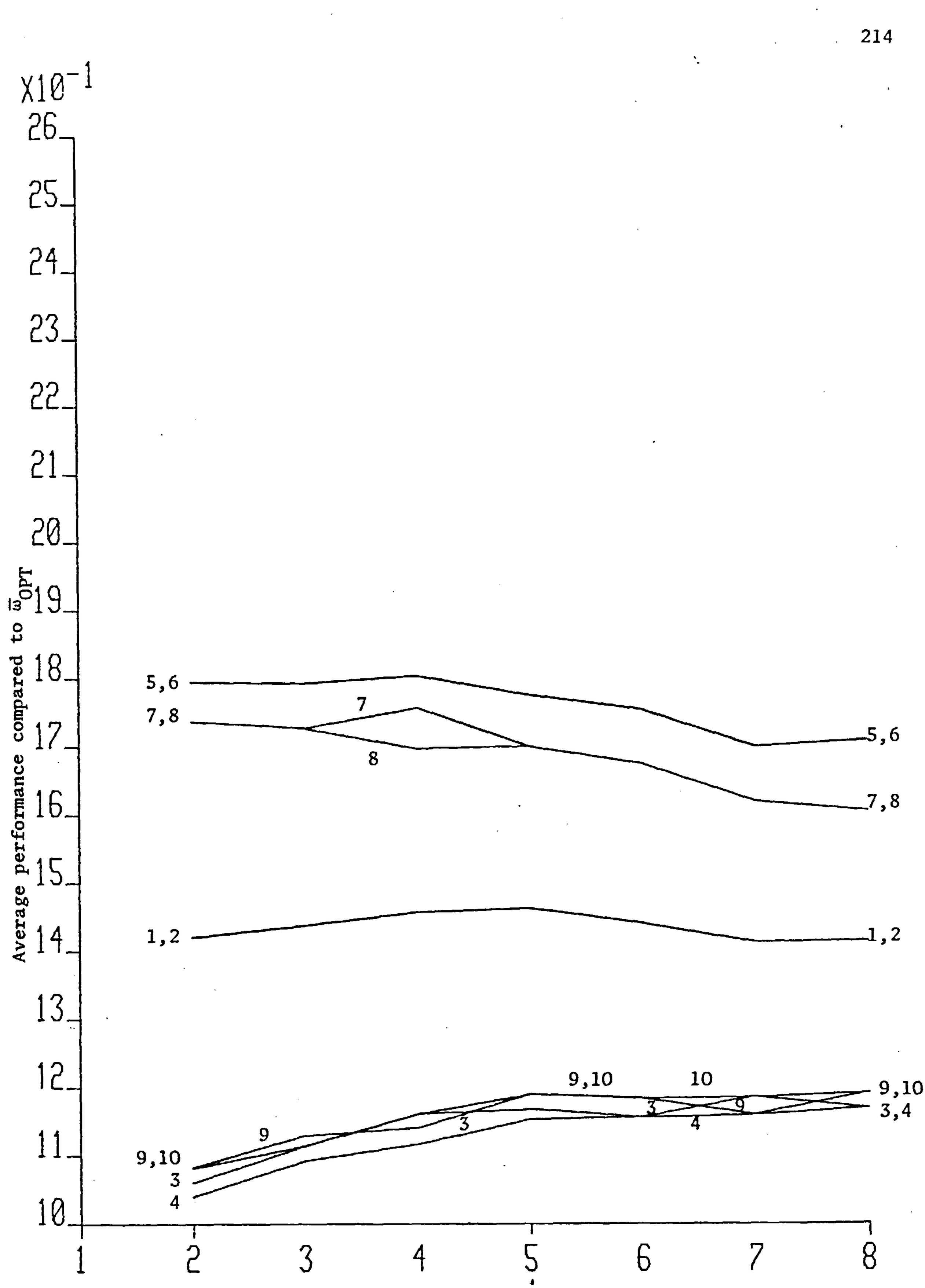


FIGURE 7.21: Test 3 - Mean flow time - P.D. algorithms

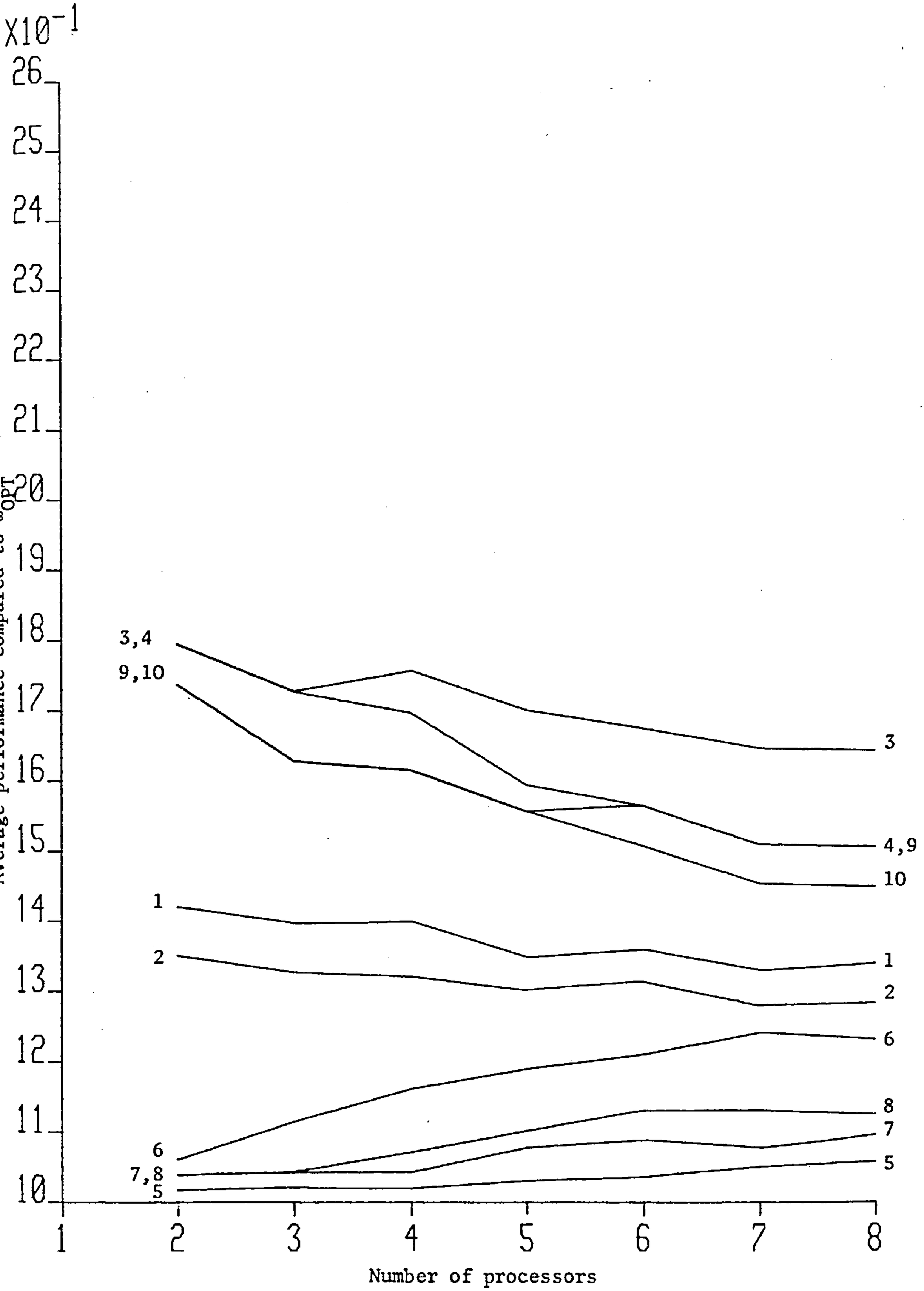


FIGURE 7.22: Test 3 - Mean flow time - Q.A.D. algorithms

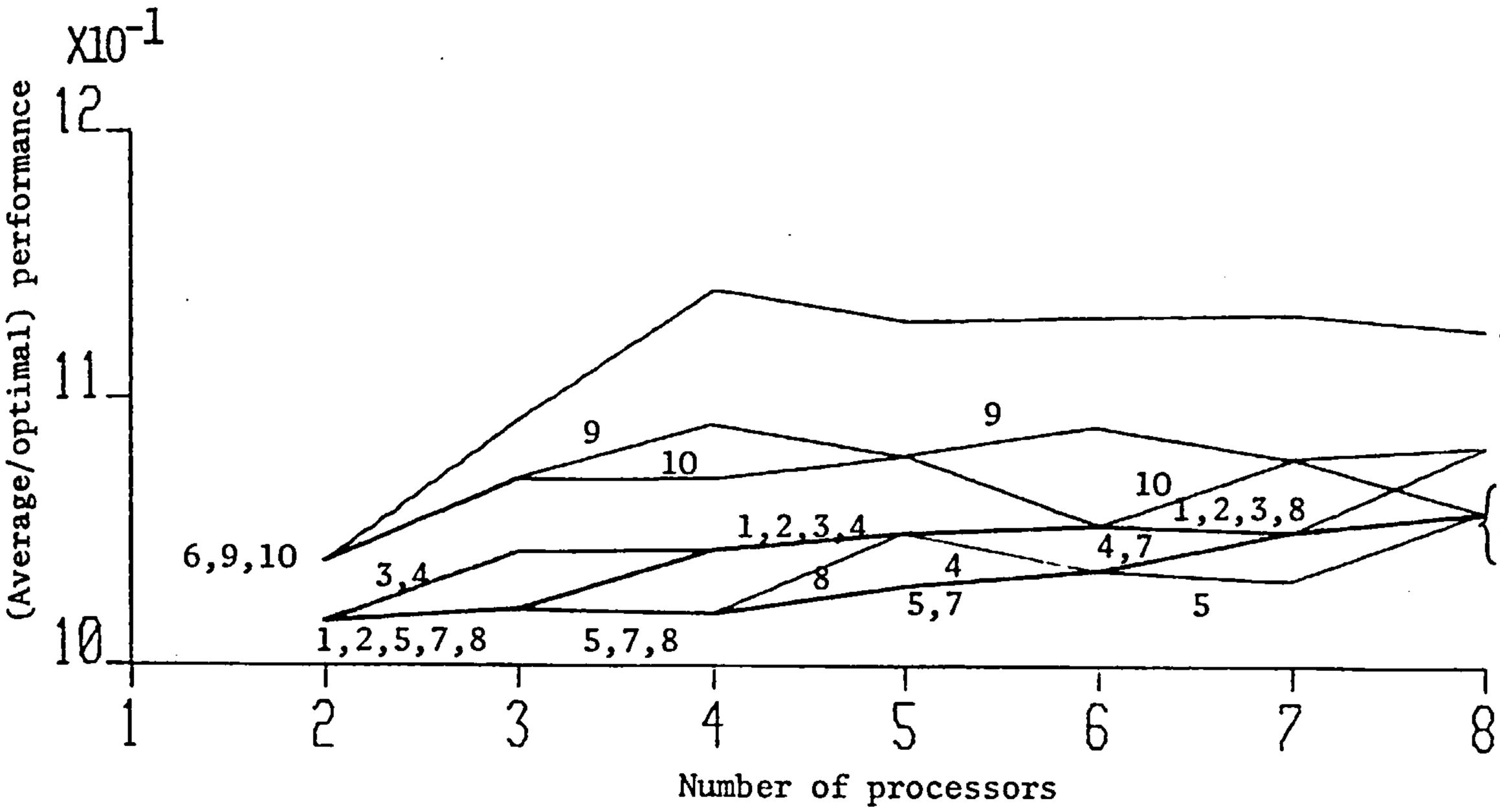


FIGURE 7.23: Test 3 - Mean flow time - Q.A.D.* algorithms

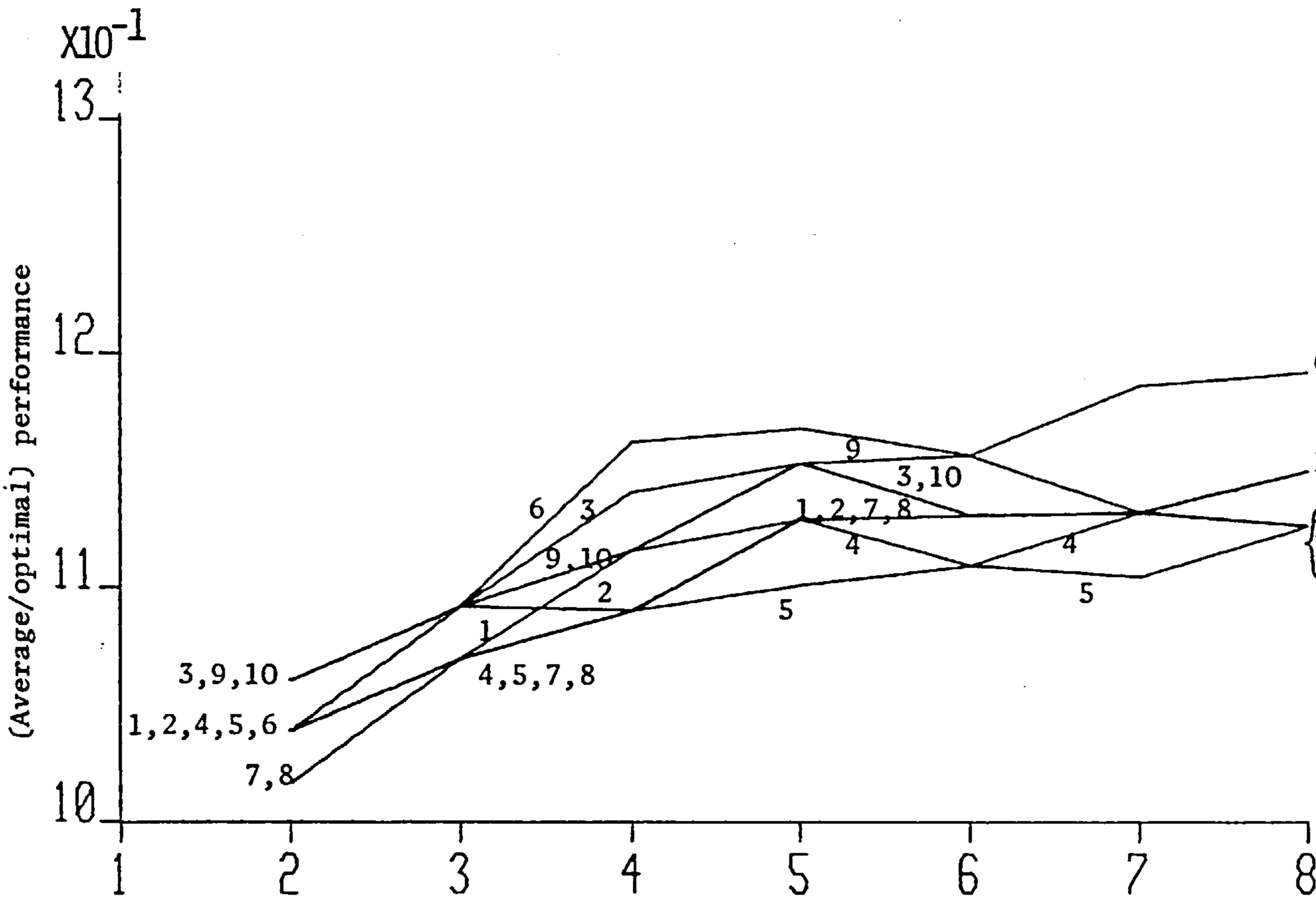


FIGURE 7.24: Test 3 - Mean flow time - P.D.* algorithms

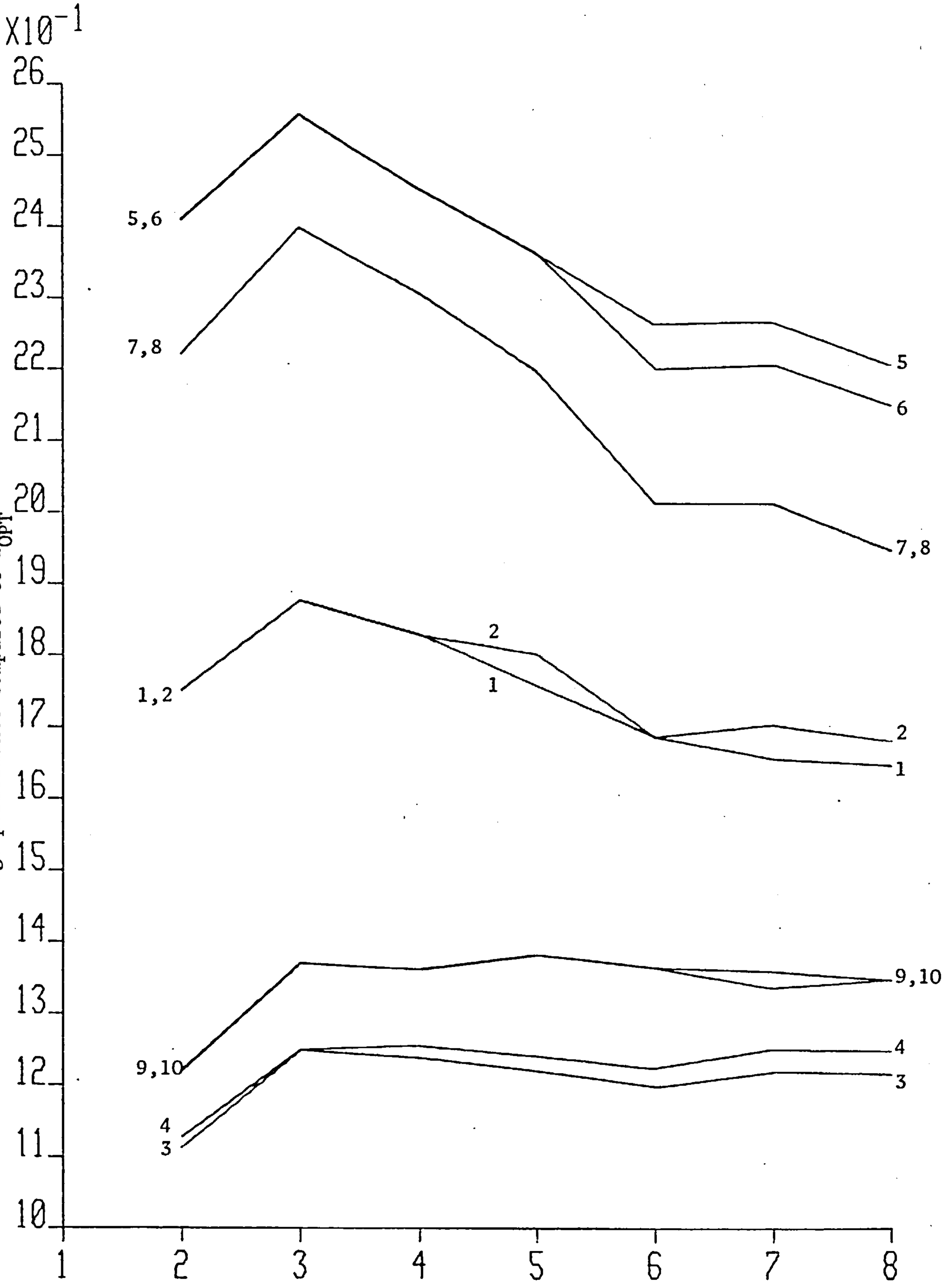


FIGURE 7.25: Test 4 - Mean flow time - P.D. algorithms

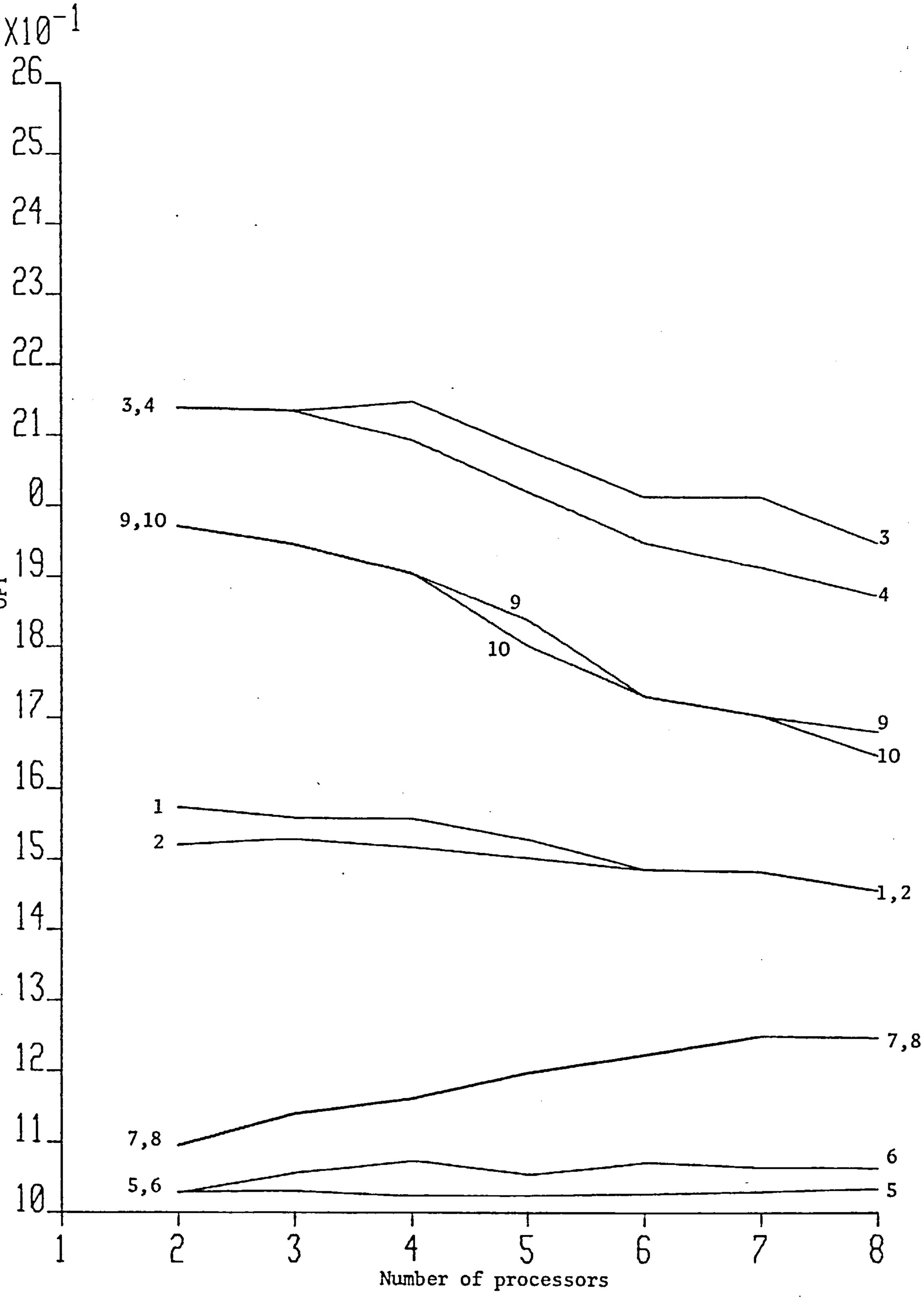


FIGURE 7.26: Test 4 - Mean flow time - Q.A.D. algorithms

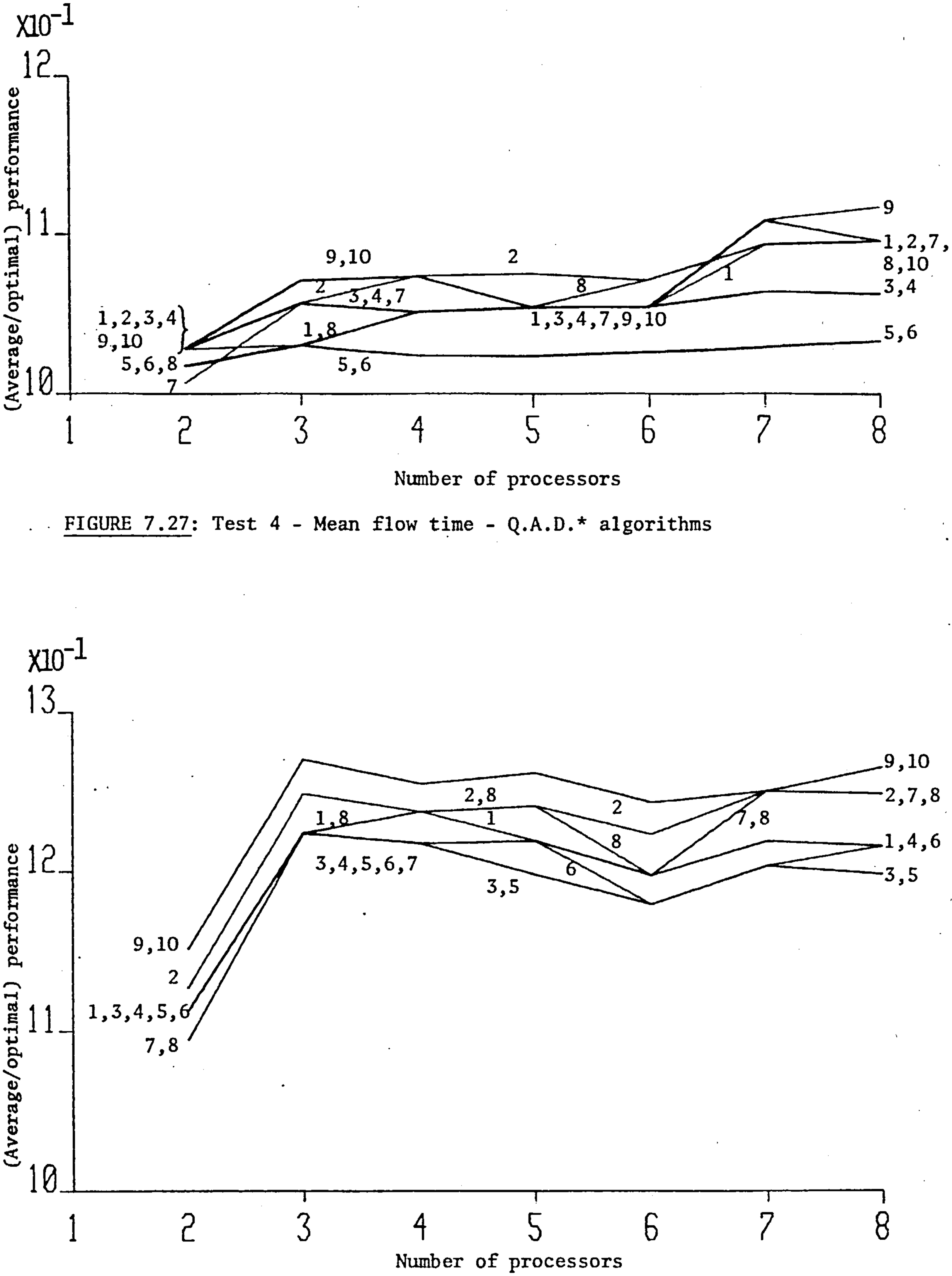


FIGURE 7.27: Test 4 - Mean flow time - Q.A.D.* algorithms

FIGURE 7.28: Test 4 - Mean flow time - P.D.* algorithms

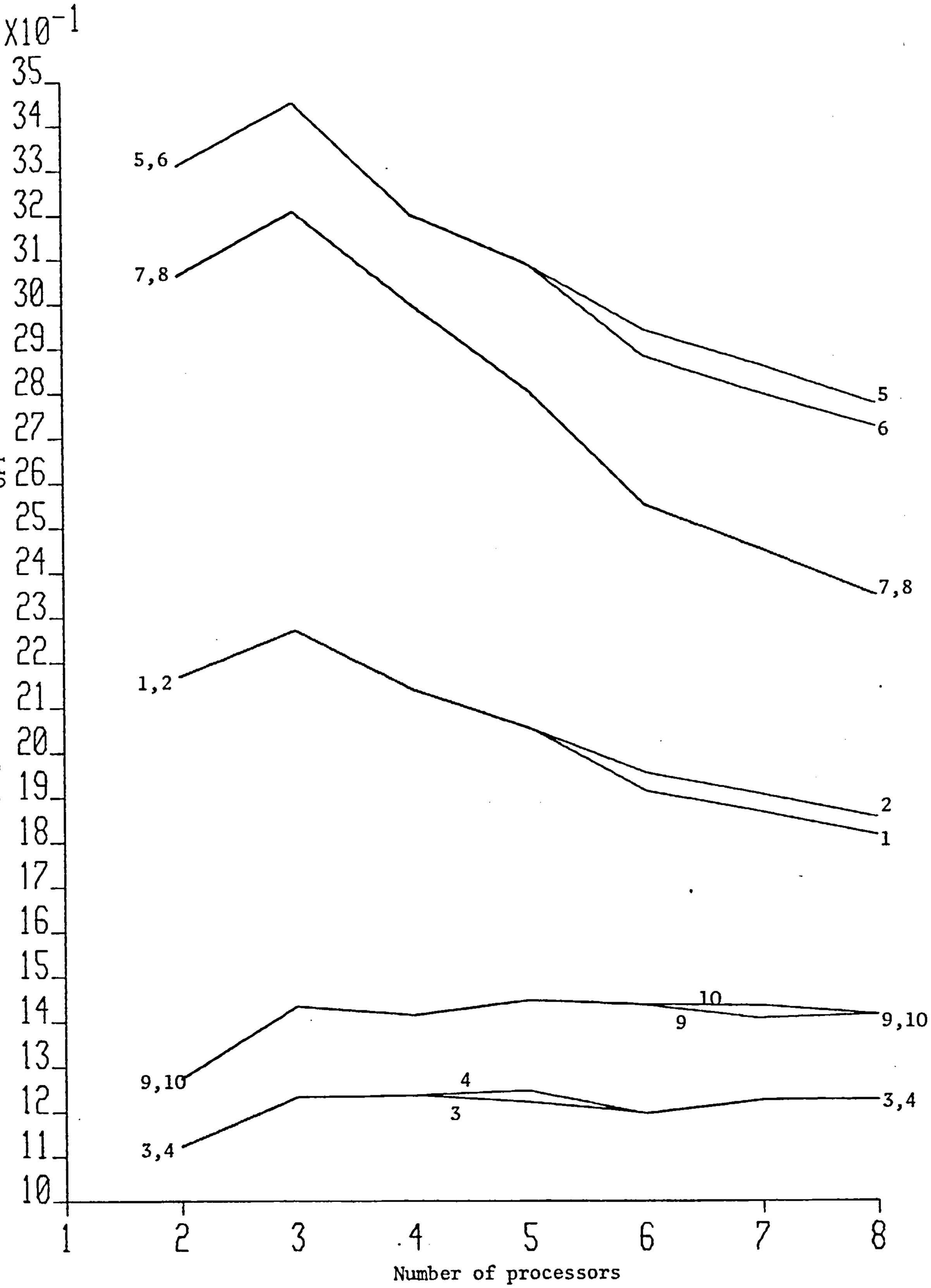


FIGURE 7.29: Test 5 - Mean flow time - P.D. algorithms

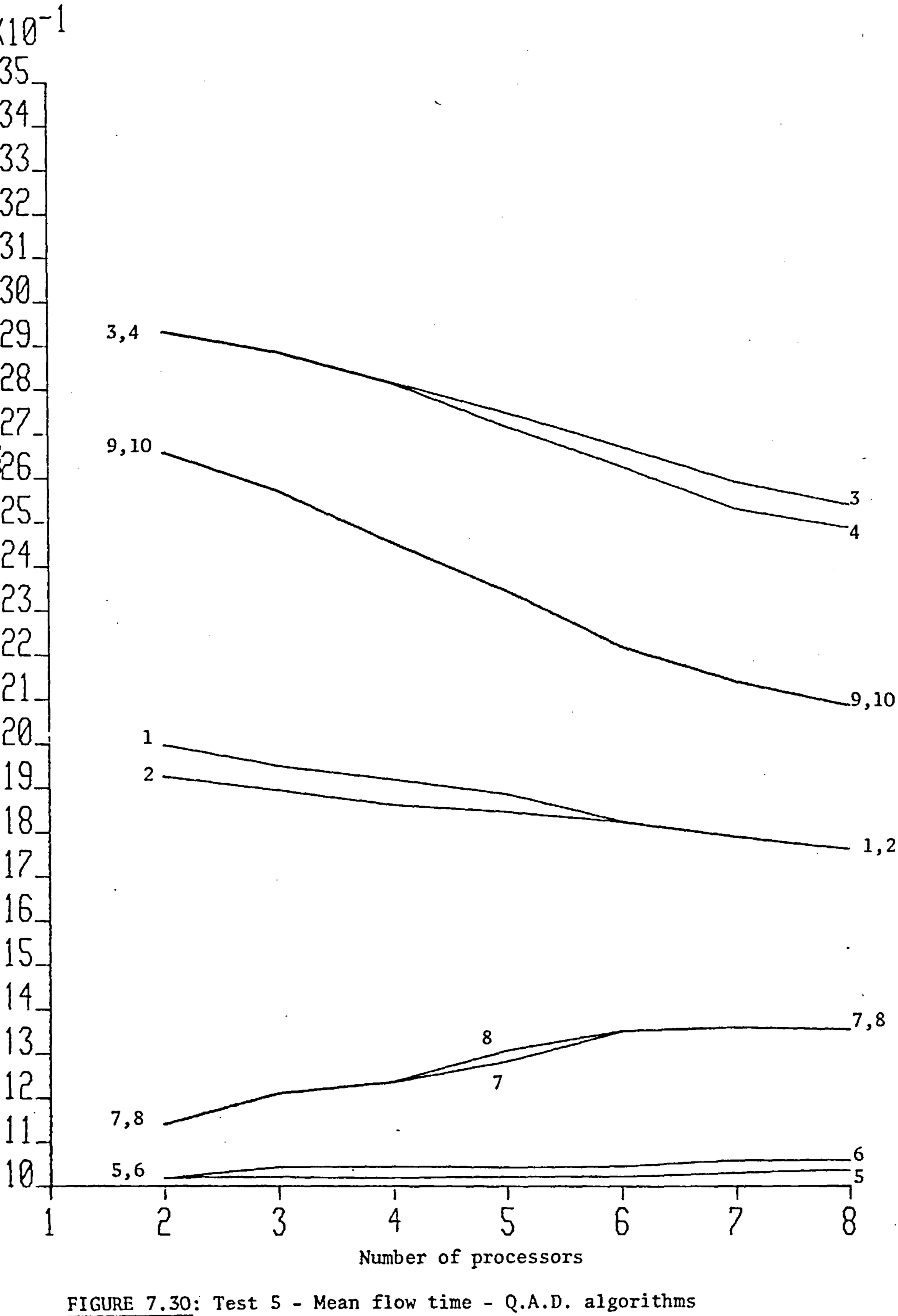


FIGURE 7.30: Test 5 - Mean flow time - Q.A.D. algorithms

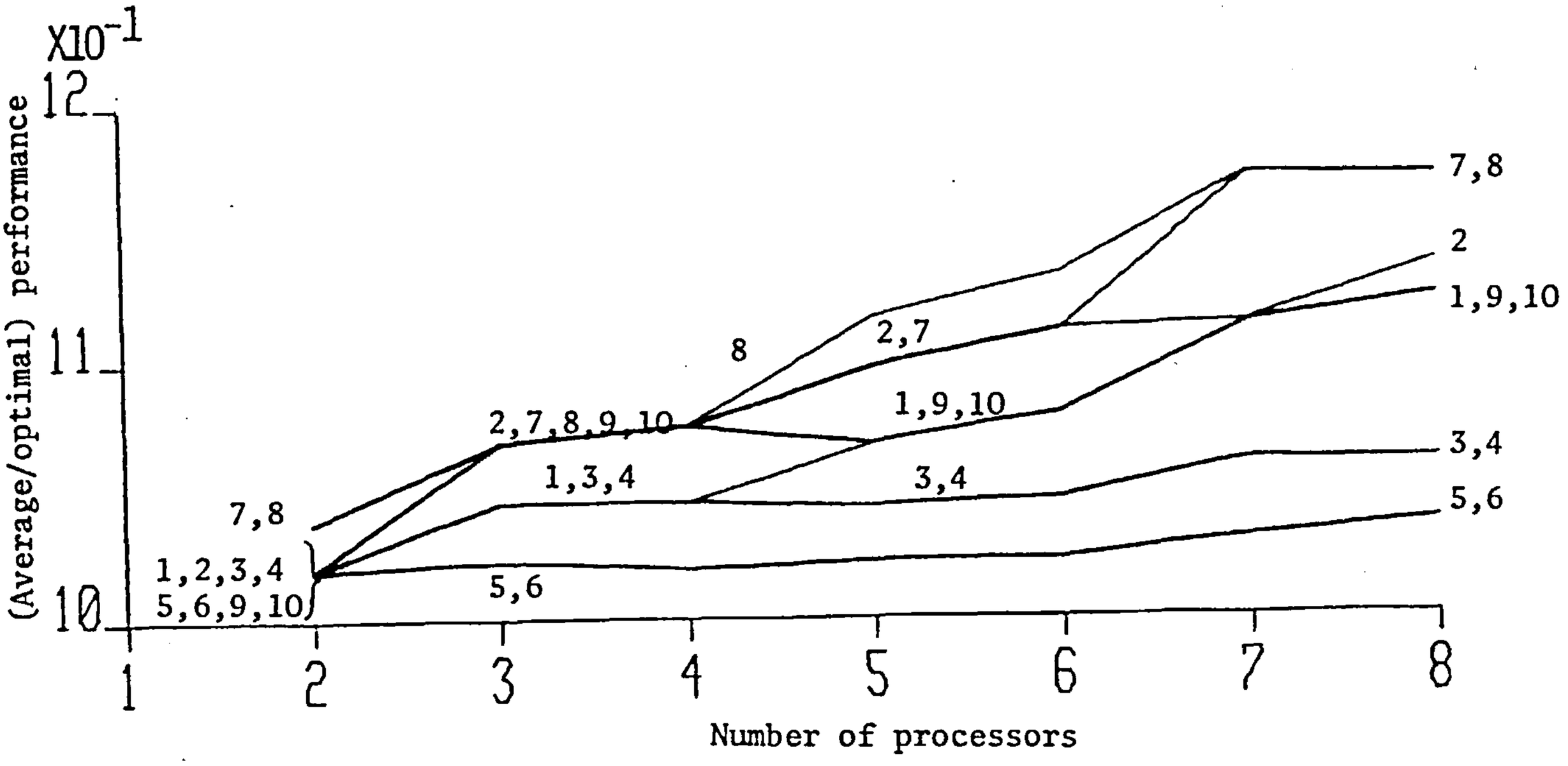


FIGURE 7.31: Test 5 - Mean flow time - Q.A.D.* algorithms

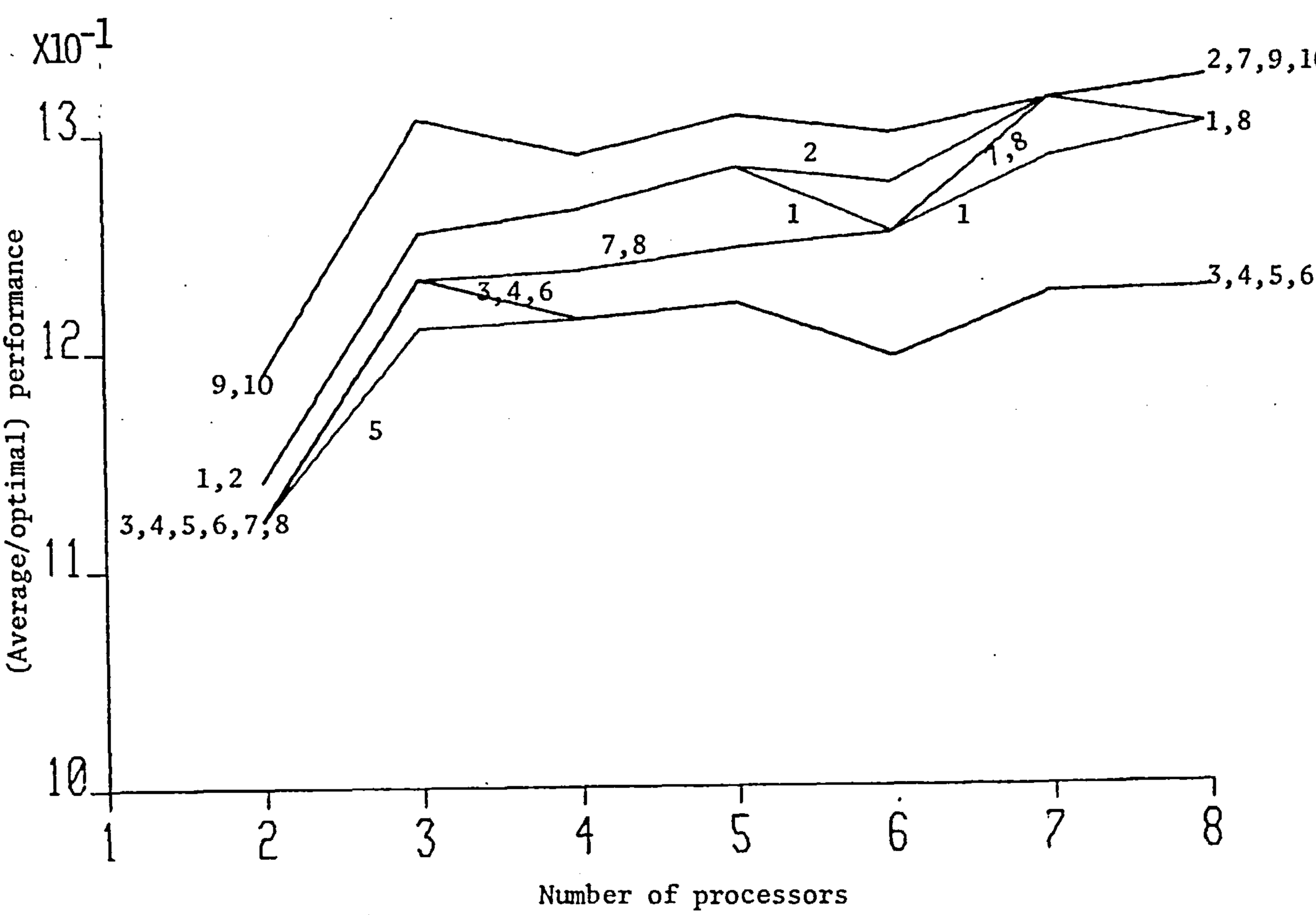


FIGURE 7.32: Test 5 - Mean flow time - P.D.* algorithms

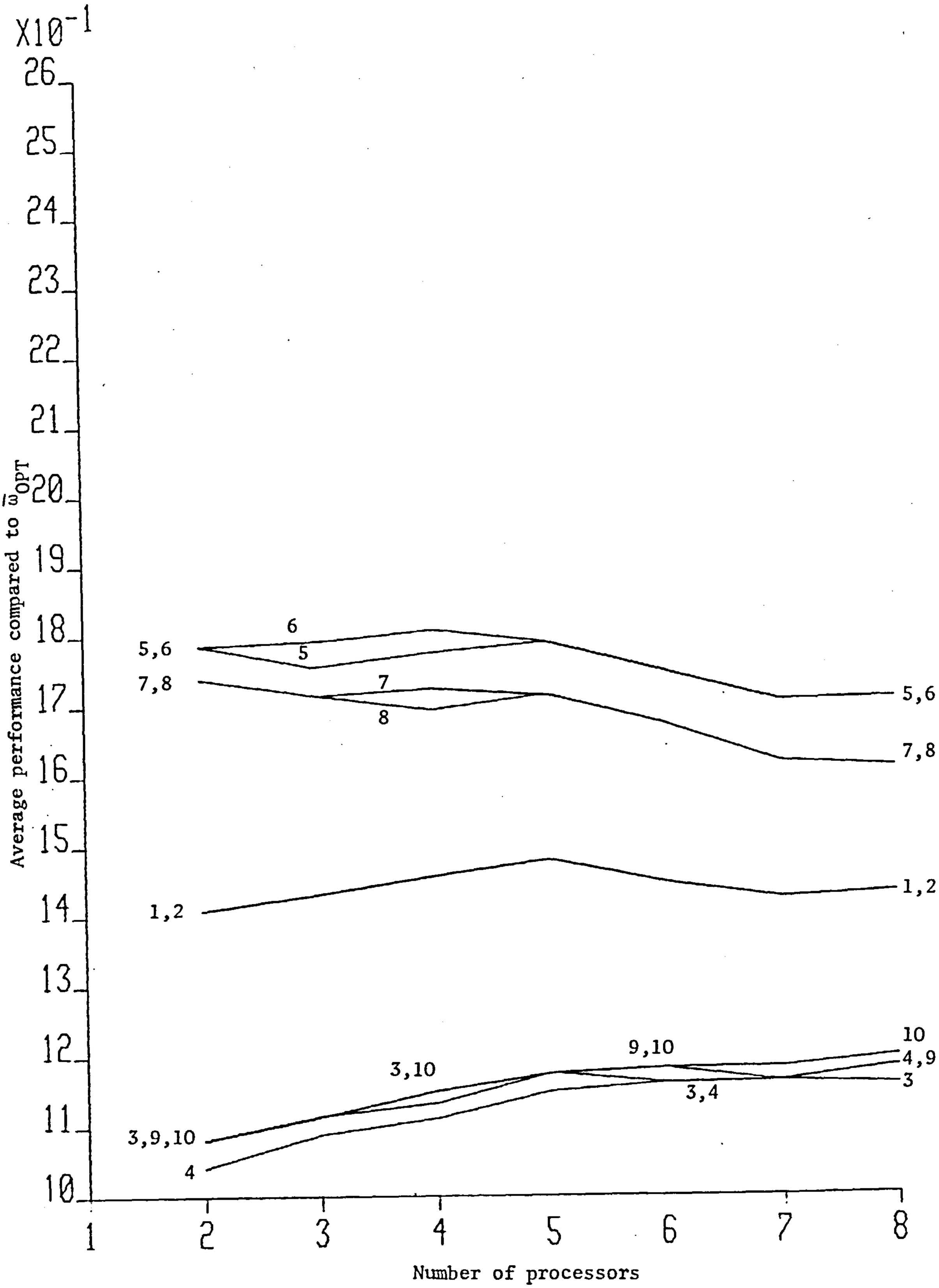


FIGURE 7.33: Test 9 - Mean flow time - P.D. algorithms

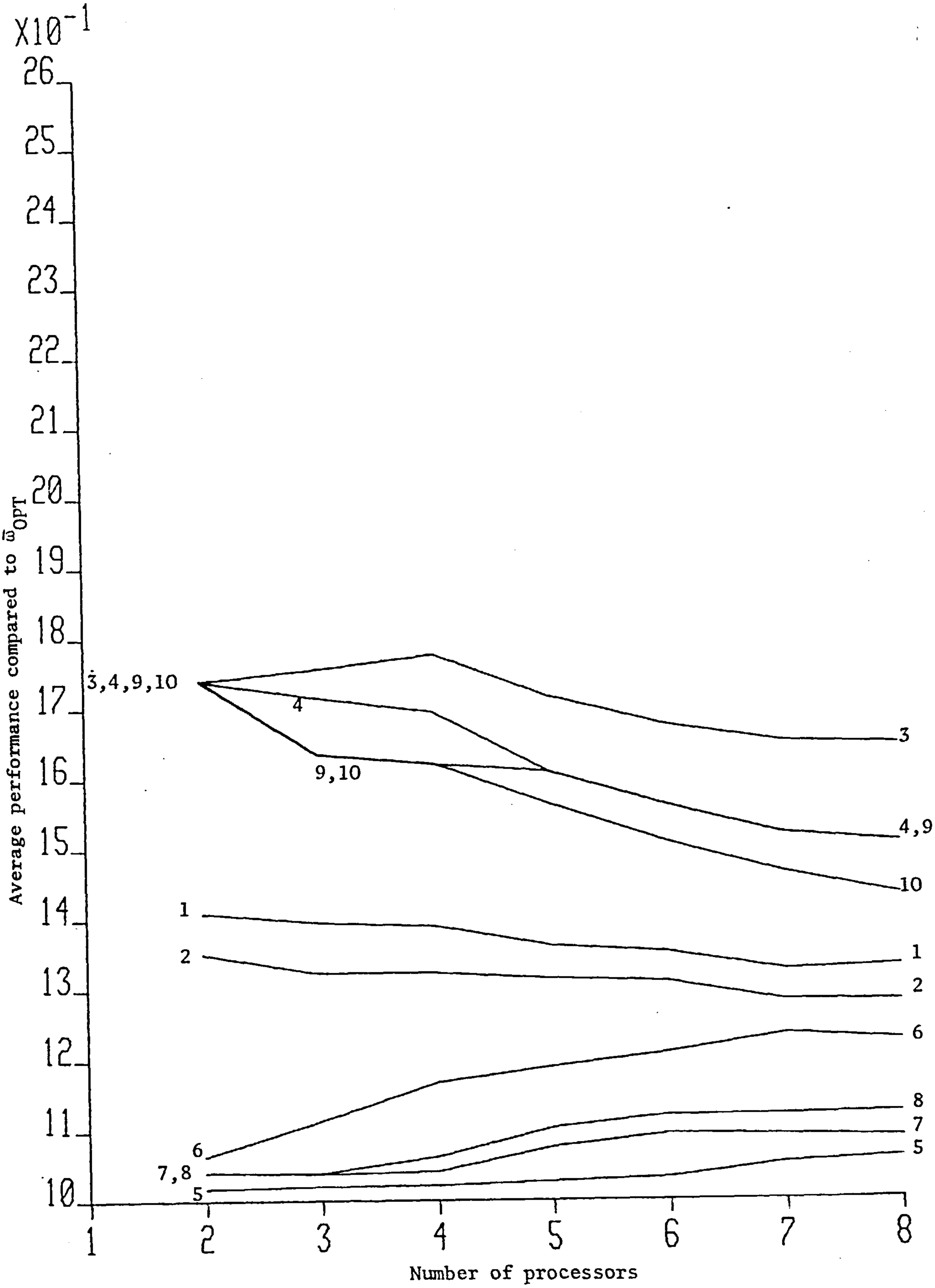


FIGURE 7.34: Test 9 - Mean flow time - Q.A.D. algorithms

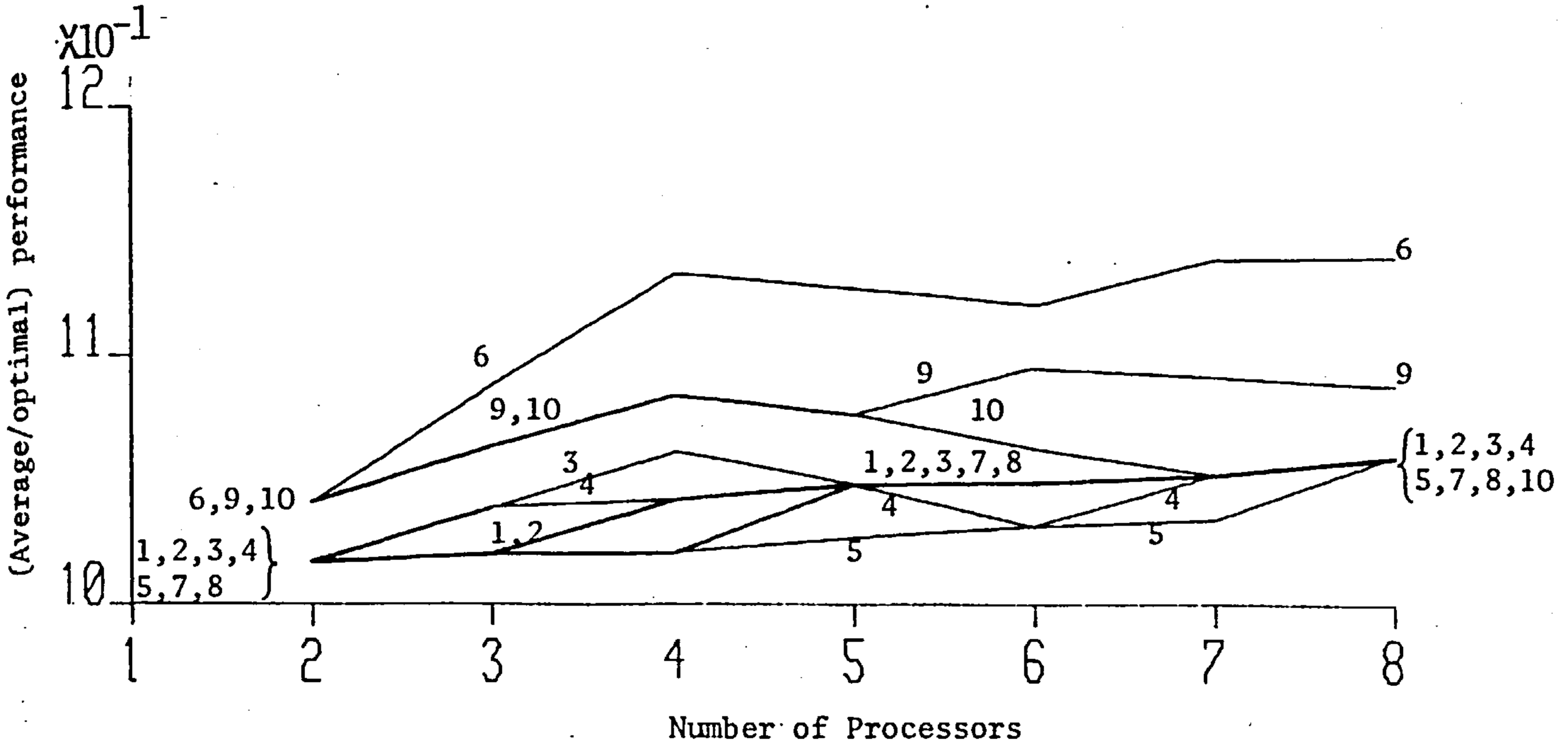


FIGURE 7.35: Test 9 - Mean flow time - Q.A.D.* algorithms

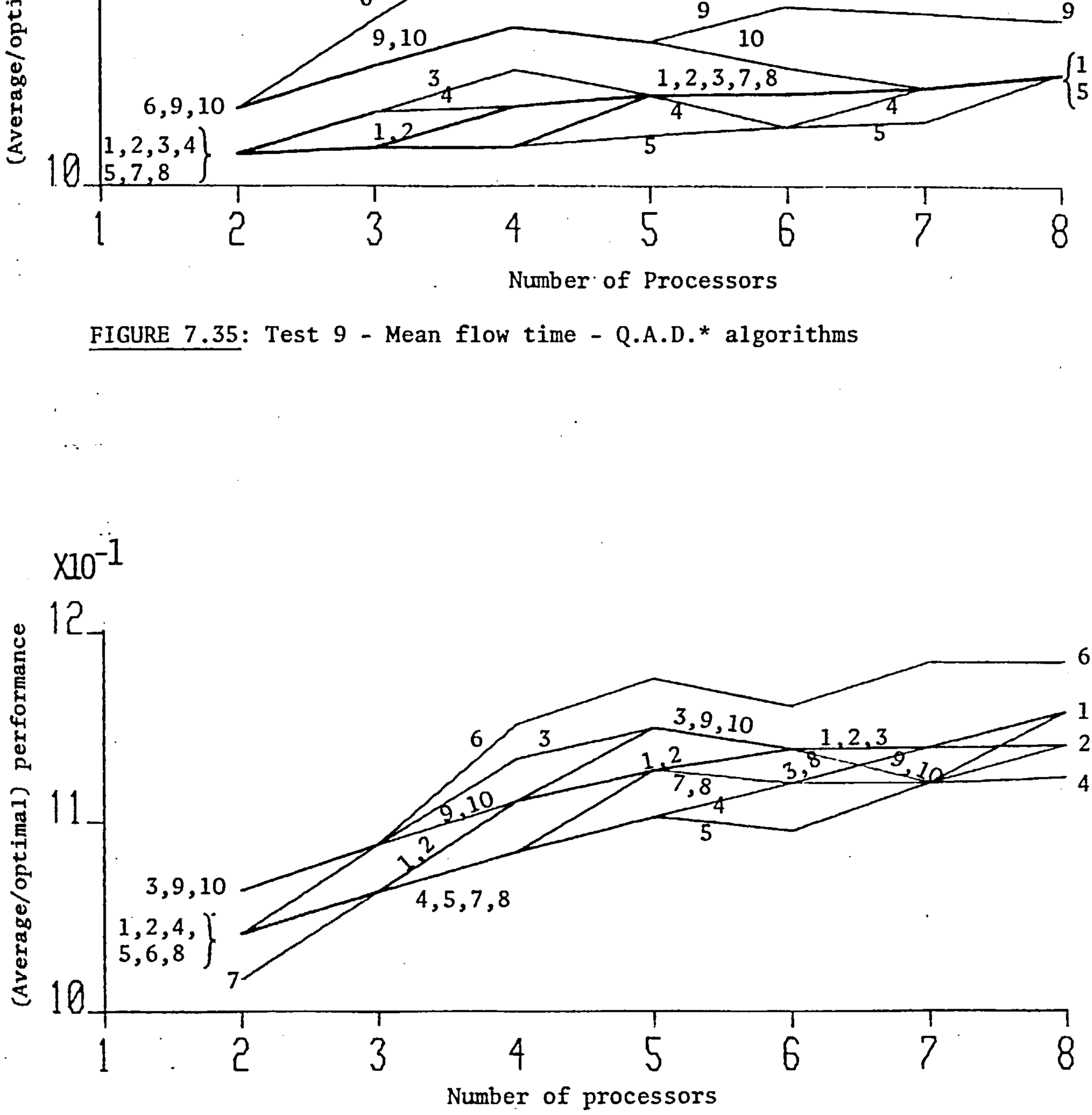


FIGURE 7.36: Test 9 - Mean flow time - P.D.* algorithms

However, such a behaviour of the performance agrees with the one which can be predicted by the worst-case bounds when most of the jobs in the task systems require small memories (since k tends to higher values, i tends to m and λ_i tends to λ).

The results of tests 5 and 6 agree with those of test 4 but they are affected, in the same way as in test 2 and 3, by the biased distributions favouring the short and long time jobs respectively. Finally, the results of tests 7,8 and 9, apart from small variations in the values of the ratio (average/optimal) performance, agree absolutely with the results of tests 1,2 and 3 respectively. As a sample, we give the results of the tests 5 and 9 in Fig.7.29-7.36.

Now, we proceed to show the results of the considered tests when the completion time is chosen as the performance criterion. The results of the 2^{nd} test are given in Fig.7.37 and 7.38. Although the ranking of the average performance of the P.D. algorithm under the various ordering rules is not the same as the corresponding one in test 1, there is a perfect agreement with the ranking of the worst-case bounds. Moreover, the performance of the algorithm under the LTF_{MAX} or STF_{MAX} rule is closer to the corresponding one of the LTF_{MIN} or STF_{MIN} ordering rule, respectively. This happens since most of the jobs in this test require short execution times and hence the differences between τ_j and σ_j for 75% of the jobs are small. On the other hand, the distance between the curves of the average performance of the LMST and LMLT ordering rules worsens as compared with test 1. The existence of a small number of jobs with long time requirements is the reason for this behaviour. Further, for the Q.A.D. algorithm there exists a considerable agreement between the average and the worst-case performance. Nevertheless, we should notice that the use of memory in contributing to the STF and LTF ordering rules worsens their average performance. This is not clear from the established worst-case bounds in section 6.3.

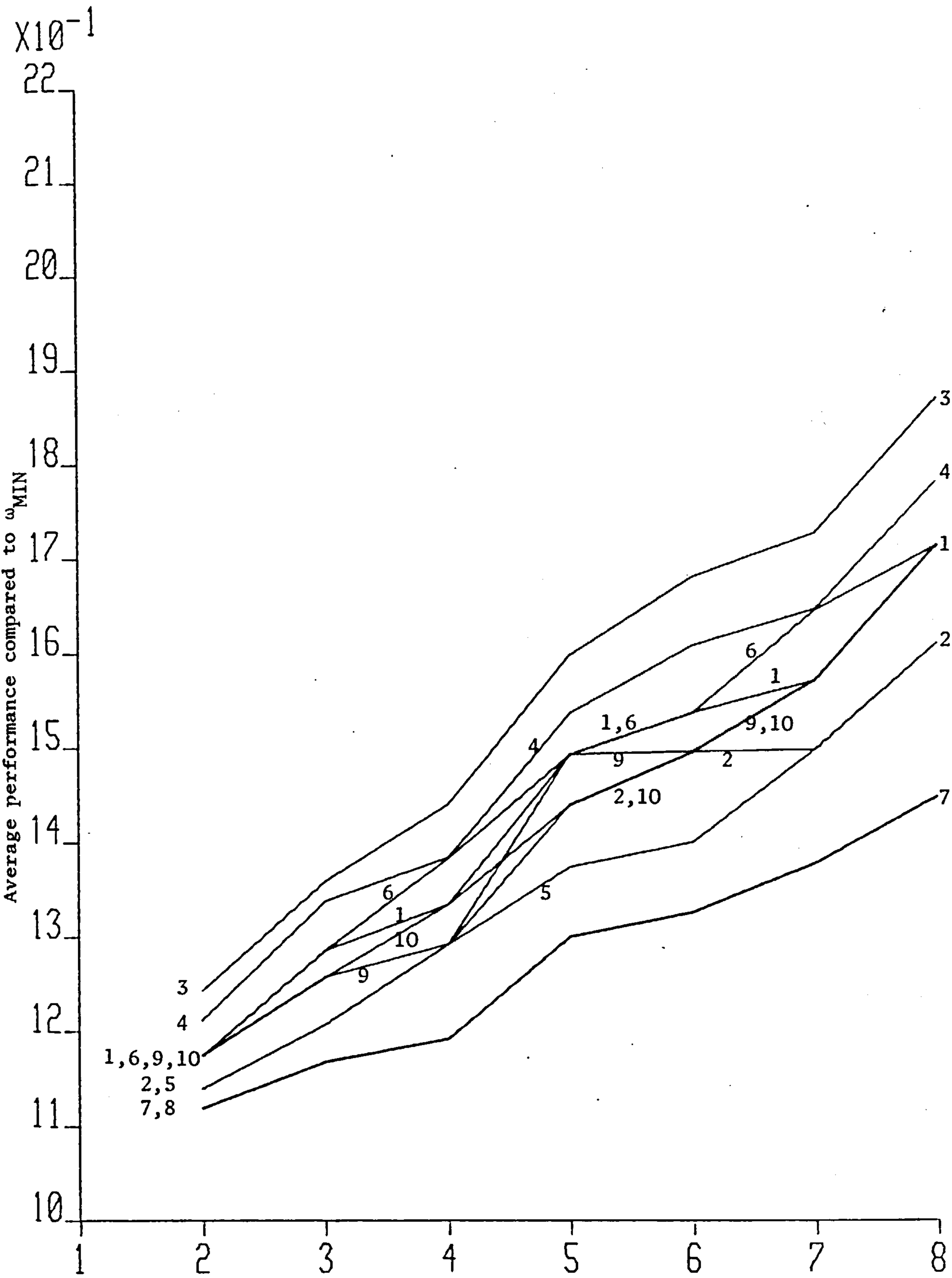


FIGURE 7.37: Test 2 - Completion time - P.D. algorithms

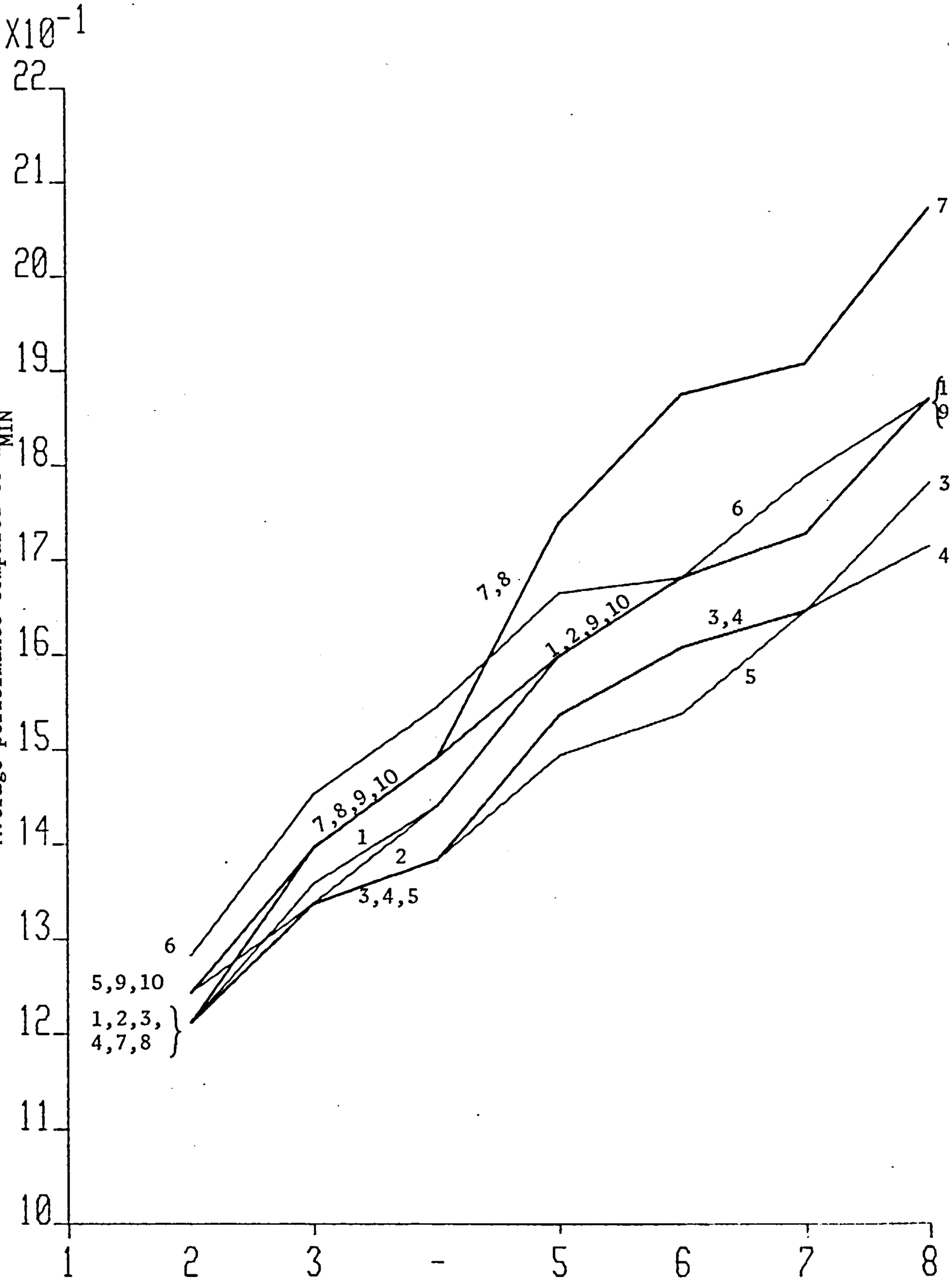


FIGURE 7.38: Test 2 - Completion time - Q.A.D. algorithms

The results of test 3, which are shown in Fig.7.39 and 7.40, are similar to the ones in test 1. However, the following distinct observations can be made for the P.D. algorithm:

- there is a larger difference in the distance between the performance curves of the STF_{MIN} and STF_{MAX} as well as between the LTF_{MIN} and LTF_{MAX} ordering rules; and
- the performance curves between the LMST and LMLT ordering rules are closer.

This behaviour is due to the large number of jobs with long time requirements and opposes the behaviour of those procedures in test 2.

Furthermore, the results of the experiments made for test 4 are presented in Fig. 7.41 and 7.42. Generally, the rankings of the average performance of both of the algorithms for the various ordering rules differ significantly as compared to the rankings of the previous tests. In detail, for the P.D. algorithm we can observe that:

- the RAND ordering, in some instances, has better performance than the LMF ordering while it always performs better than the LMST procedure;
- the RAND, STF, LMF and LMST as well as the LTF and LMLT ordering rules have an average performance very close to each other; and finally
- the utilisation of memory in contributing to the LTF_{MIN} ordering does not offer any improvement when $m \geq 3$ while it does so for some instances when contributing to LTF_{MAX}.

Such a behaviour was expected and actually, agrees with the one predicted by the worst-case bounds, since ℓ tends to m when most of the jobs require small memories. Therefore, the agreement between the rankings of the average and extreme performance of the P.D. algorithms is still perfect.

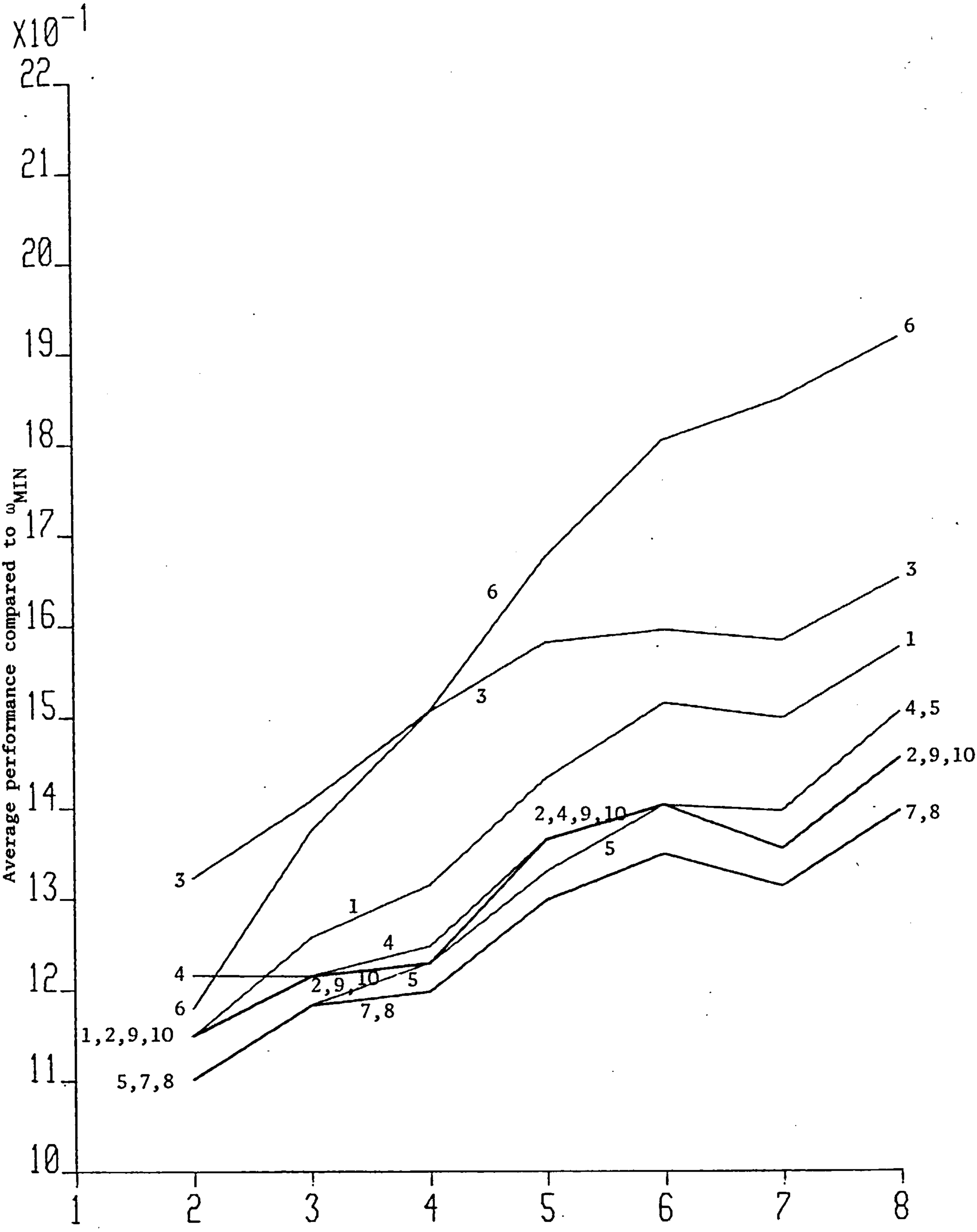


FIGURE 7.39: Test 3 - Completion time - P.D. algorithms

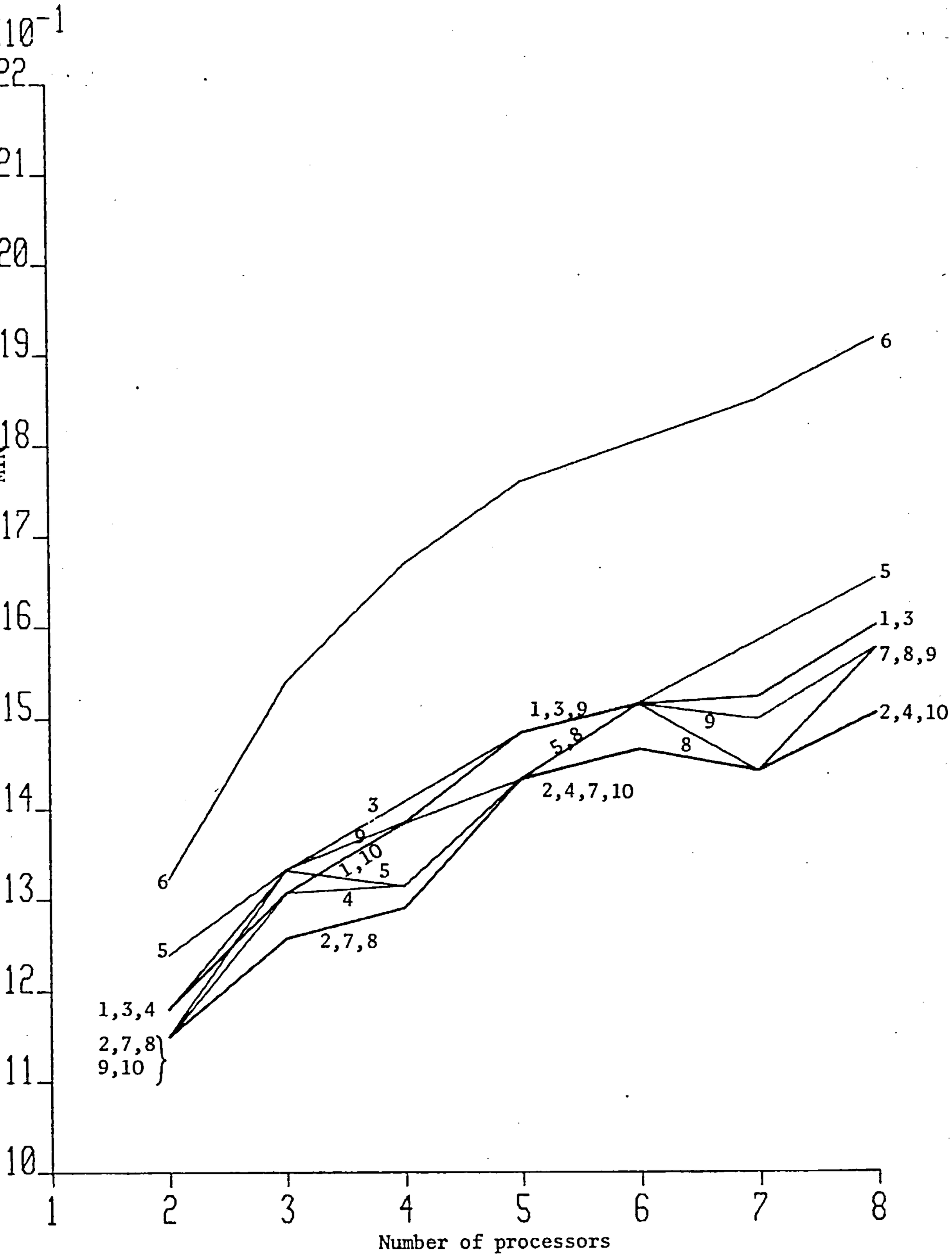


FIGURE 7.40: Test 3 - Completion time - Q.A.D. algorithms

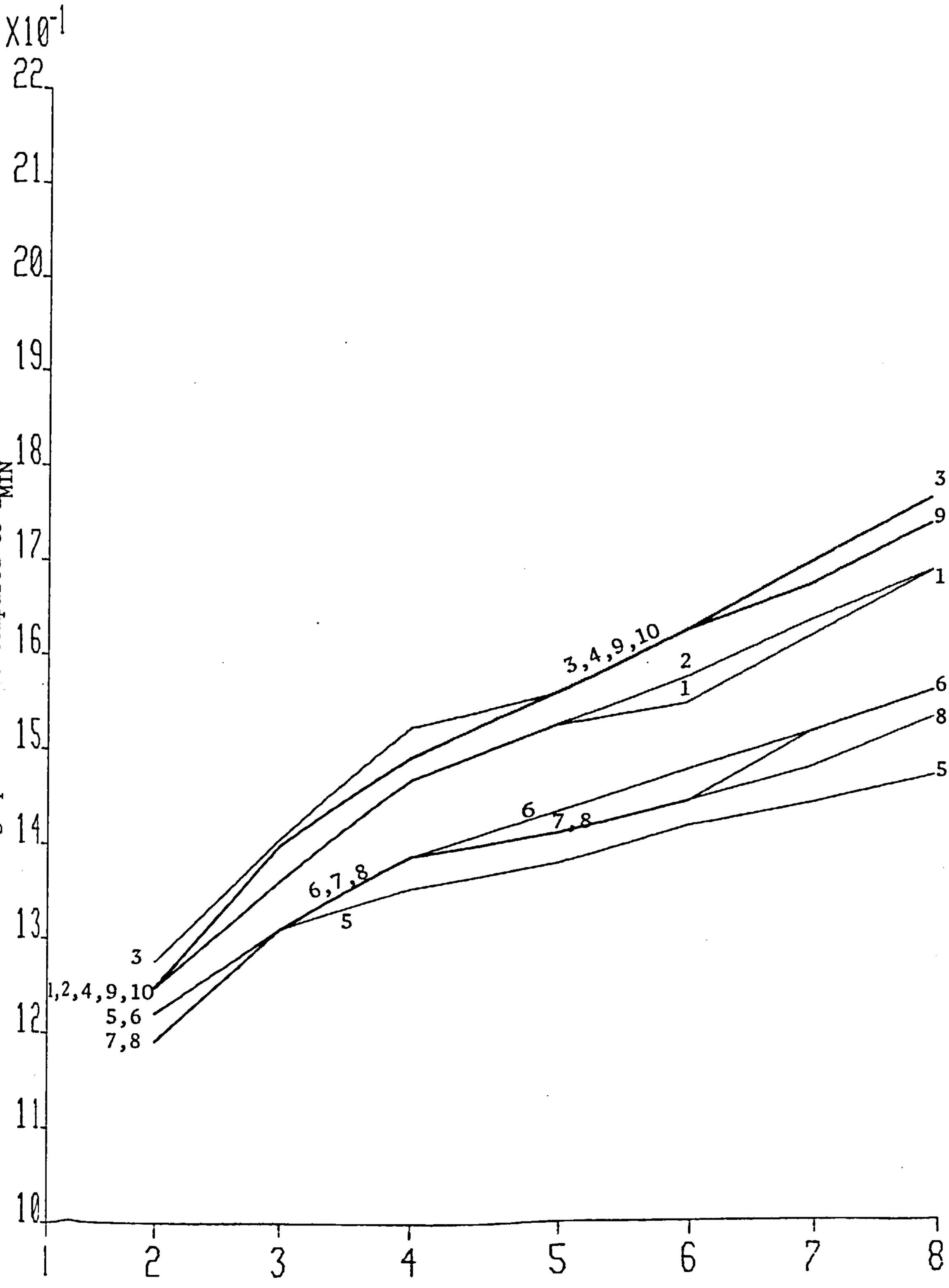


FIGURE 7.41: Test 4 - Completion time - P.D. algorithms

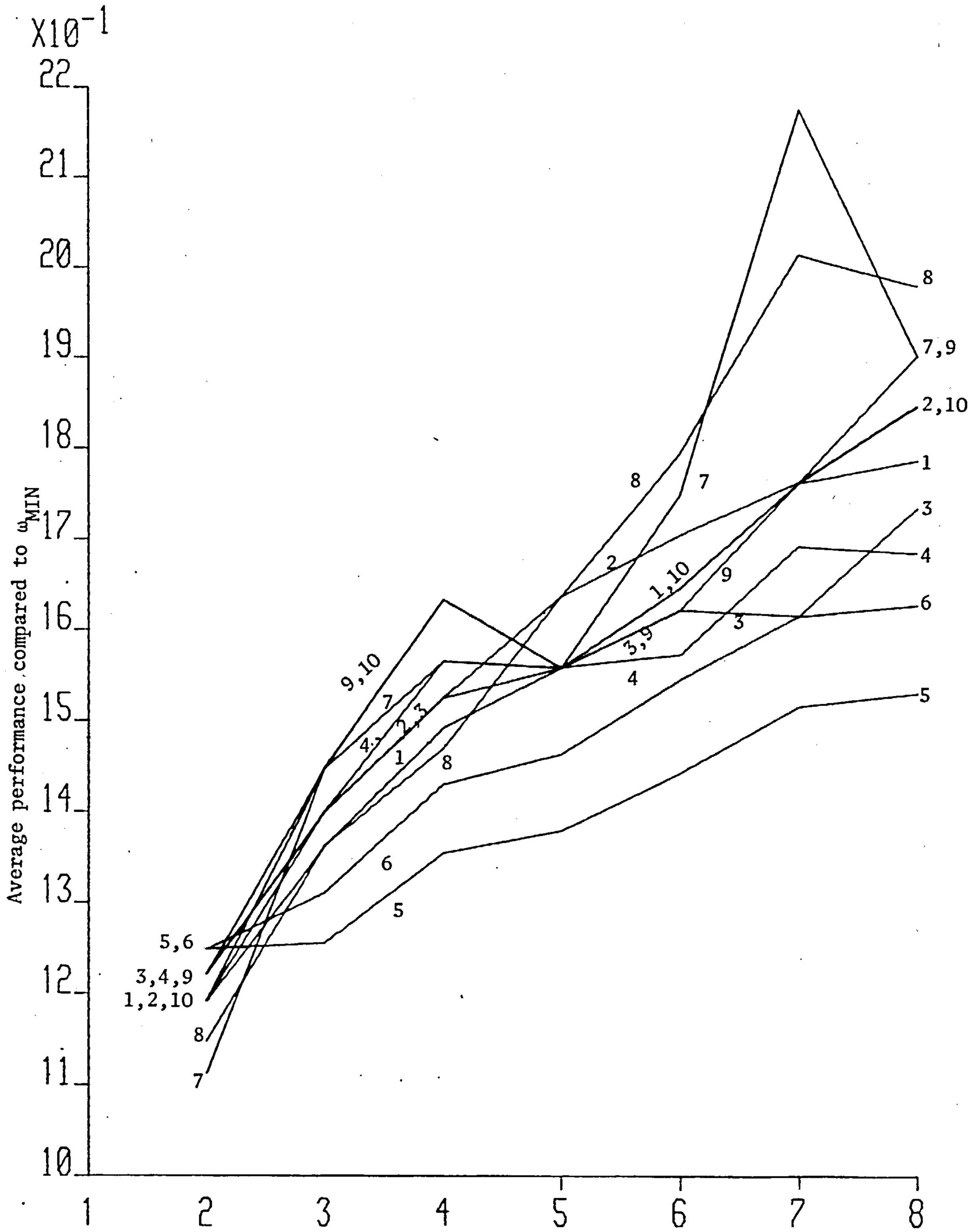


FIGURE 7.42: Test 4 - Completion time - Q.A.D. algorithms

Then, for the Q.A.D. algorithm we can observe that the utilisation of memory in contributing to the STF or LTF orderings does not offer any improvement if $m \geq 3$. This agrees with the worst-case bounds when ν_x and ℓ^* tend to m. However, we can say that generally, there is a considerable agreement between the rankings of the algorithms for the two performance measurements.

The results of the experiments for test 5 and 6 agree with those of test 4 but they are affected, in the same way as in test 2 and 3, by the biased distributions favouring the short and the long time jobs respectively. Moreover, the results of tests 7,8 and 9, apart from small variations in the values of the ratio (average/optimal) performance, agree exactly with the results of tests 1,2 and 3 respectively. As a sample, we present the results of the experiments made for test 5 and 9 in Fig. 7.43-7.46. Therefore, tests 5,6 and 7,8,9 behave in the same manner as compared to previous tests when the mean flow or the completion time is used as the performance criterion.

In view of the reasonable nature of the restrictions we have made for the various parameters, it is believed that the results derived from the previous experiments represent the fundamental characteristics of the average performance of the considered algorithms. Generally, we can summarise all the evidence presented in this chapter by the following conclusions. The behaviour of the average performance of the P.D. algorithms is exactly as has been predicted by the worst-case bounds when either the mean flow or the completion time performance criterion is used. The same happens for the Q.A.D. algorithms when the mean flow time is the performance criterion. For the other algorithms, when any of the two performance measurements are used, there is a considerable agreement in the behaviour predicted by the worst-case bounds and the one presented by the average performance in this chapter. So, a high degree of correlation exists between the ranking of the algorithms given by the extreme and expected performance. Finally, when the mean flow

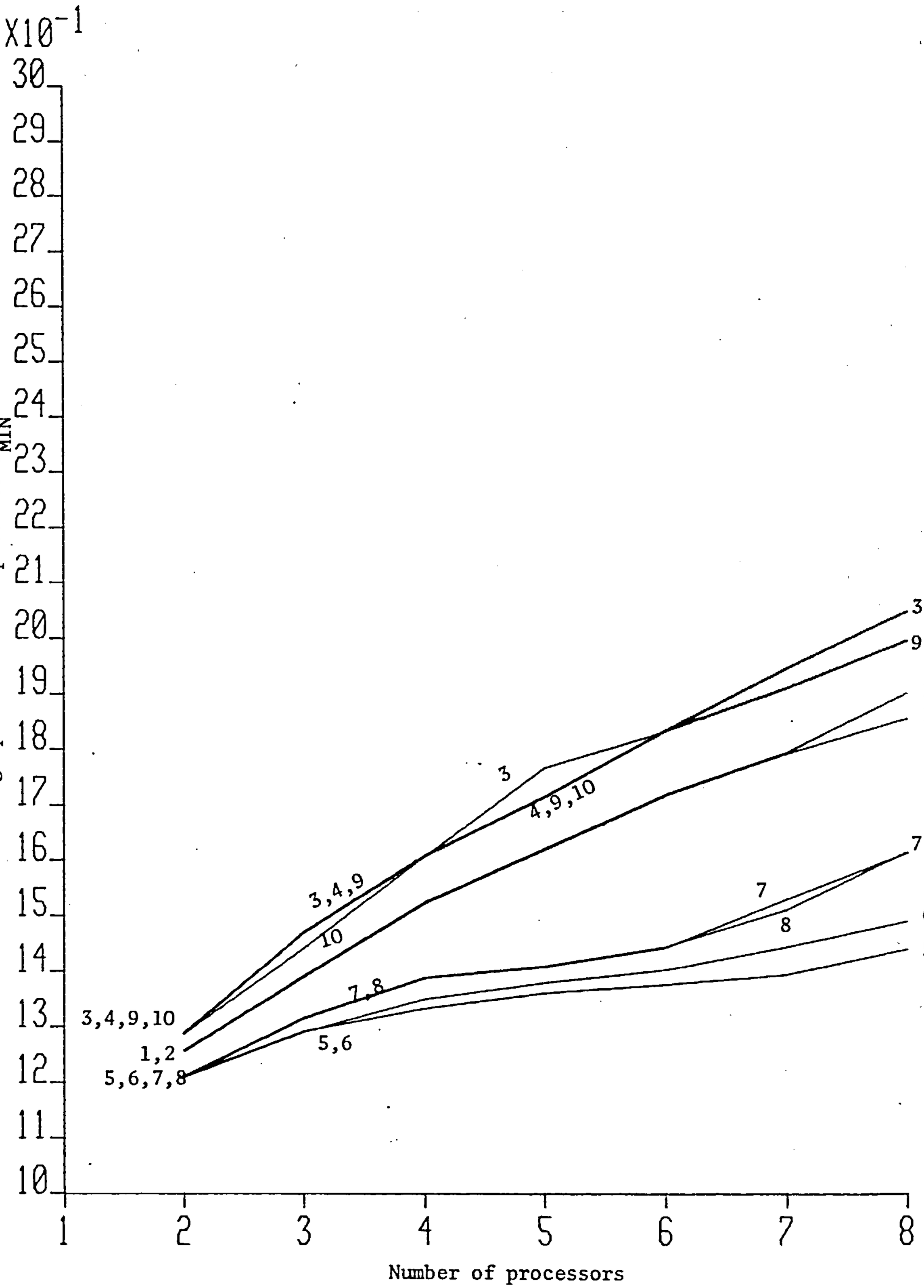


FIGURE 7.43: Test 5 - Completion time - P.D. algorithms

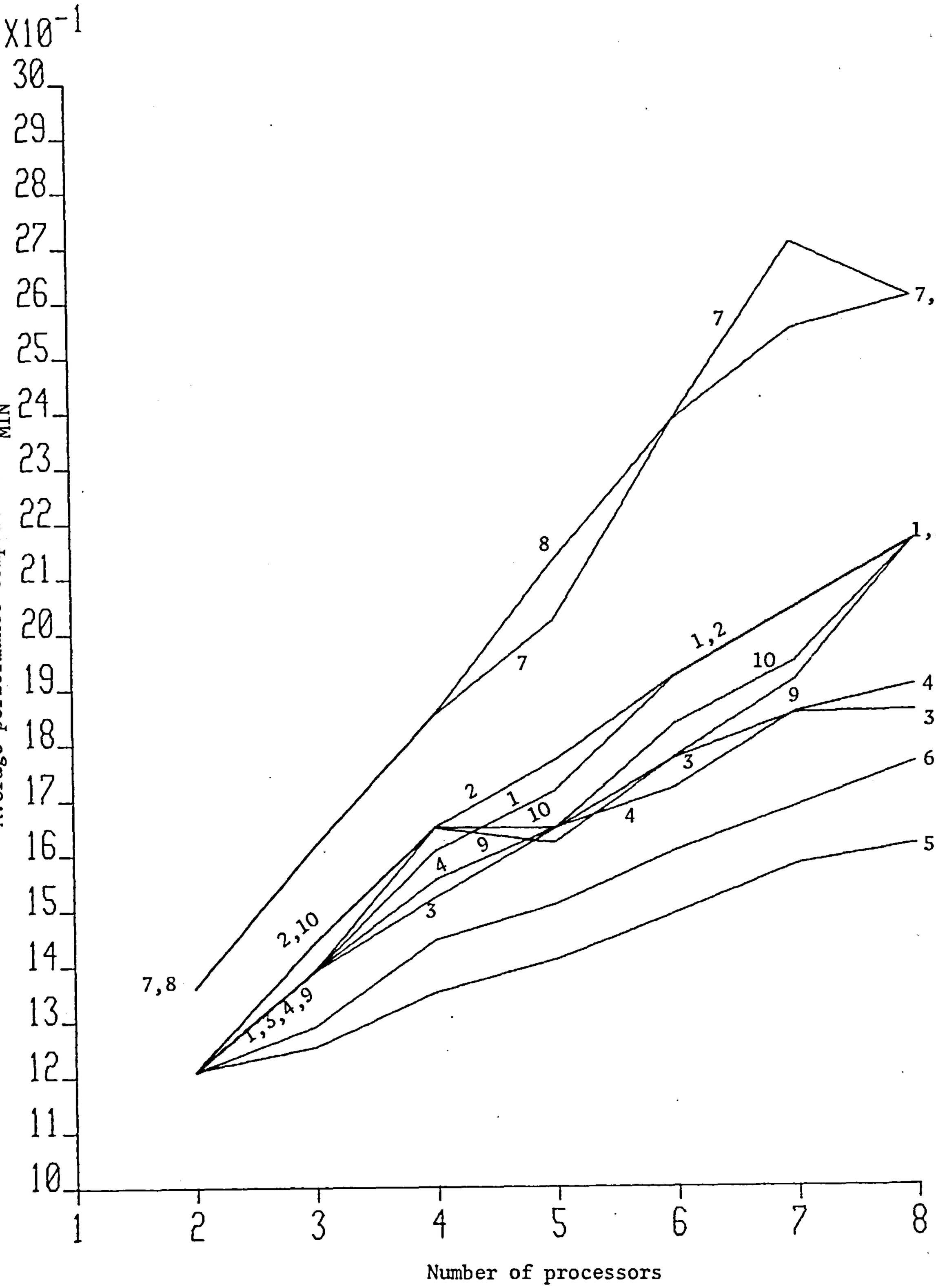


FIGURE 7.44: Test 5 - Completion time - Q.A.D. algorithms

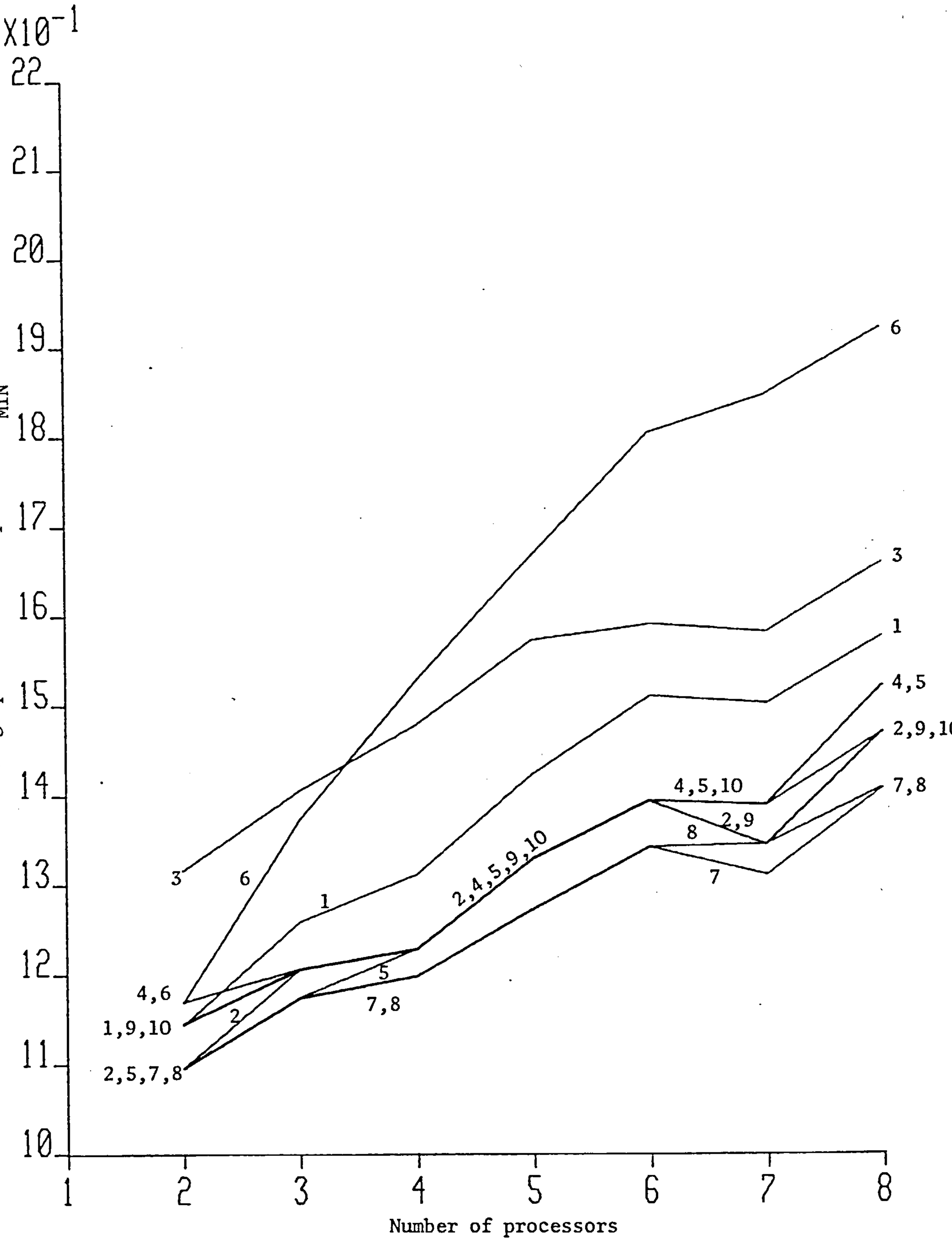


FIGURE 7.45: Test 9 - Completion time - P.D. algorithms

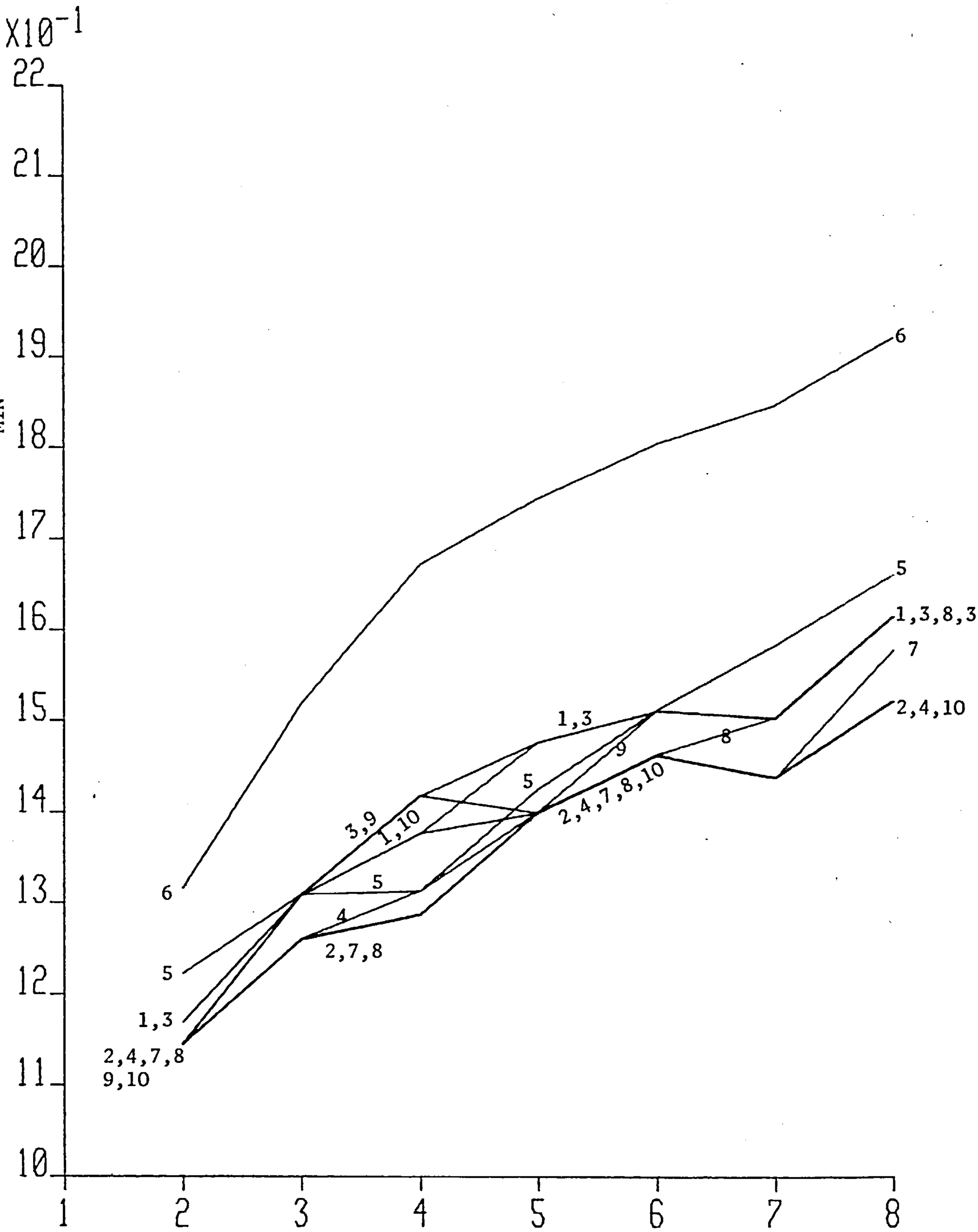


FIGURE 7.46: Test 9 - Completion time - Q.A.D. algorithms

time is the performance criterion the P.D. algorithm under the STF or LMST ordering, the Q.A.D. algorithm under the LTF or LMLT ordering and the P.D.* or Q.A.D.* algorithm under any ordering rule gives a desirable expected performance. On the other hand, when the completion time is the performance criterion, the P.D. algorithm under the LMLT orderings has always desirable expected performance and for some cases, the LMF, LMST or LTF rules present a competitive expected performance as compared to that of the LMLT ordering rules.

Chapter 8

CONCLUDING REMARKS

From the proven worst-case bounds given in Chapters 5 and 6, the simulation study presented in Chapter 7 and bearing in mind the discussions which analysed the results in each of the above mentioned chapters as well as the work previously done on related models (i.e., summarised in Chapter 3), we could draw the following conclusions.

First, the proven bounds in Chapters 5 and 6 are so informative as to make us thoroughly understand the behaviour of the considered algorithms, under the various pre-set ordering procedures, in different situations. This gives us the chance to choose the appropriate ordering rule and/or algorithm, in order to achieve desired performance goals, for a particular operational environment. So, when our interests are to obtain as good a turnaround performance of the system as possible, which is represented by the mean flow time, the best choice would be the Q.A.D. algorithm under the LTF_{MIN} rule for any operational environment. Clearly, variations of the jobs requirements in the task systems cannot affect greatly the worst-case bound for this ordering rule and hence it will always remain the best in comparison to the others. On the other hand, when we are interested to obtain better utilisation of the system resources (i.e. processors and memory in our case) then, the P.D. algorithm under the LMLT ordering rules would be the best choice for most of the cases. However, in situations where most of the jobs in the task system require small memories or alternatively most of the processors have the same private memory, then the P.D. or the Q.A.D. algorithm under the LTF ordering rules seems to be satisfactory choices and some times the best ones. Moreover, in such situations the Q.A.D. or the P.D.* algorithm under the LTF_{MIN} ordering rule would be the best choices if we would like the turnaround and the throughput of the system to be as good as possible with only a slight influence on the turnaround or throughput respectively. In other operational environments, the best algorithm to satisfy both performance criteria would be the P.D.* under the LMLT ordering rules.

The model which this thesis has examined can be considered as a generalised one compared to the ones already studied when independent jobs are to be scheduled in more than one processor. More exactly, if the parameter λ relaxes to 1 or $b_i=b_{i+1}$, $1\leq i\leq m-1$ then, our model becomes the one which Kafura and Shen have studied (see [Kaf],[KS1],[KS2]). In this case, the proven worst-case bounds are identical to the ones given in Appendices I-IV. Moreover, although the results given in Appendices I,II and IV are new and we cannot compare with others, the bounds proven in Appendix III are more informative and more general than the ones Kafura and Shen found. In fact, their bounds satisfy the corresponding ones given in Appendix III only for special values of the parameters. (see Tables 3.5 and 3.7, problem 4 and problems 8-11 respectively). Further, when $\ell=m$, our model becomes the classical (i.e., without any resource restrictions) heterogeneous multiprocessor model with non-identical or uniform processors. Clearly, for non-identical processors when ℓ or ν_i becomes m the bounds given in Theorems 5.5.1, 5.5.3 and 5.5.4 are better and more informative than the ones proven by Clark [Cℓ] (see Table 3.9). Also, the bounds in Theorems 6.2.1 and 6.2.5 for non-identical processors appear more informative and probably better than the ones proven by Ibara and Kim [IK] (see Table 3.8, problems 3,4 and 5), although even better bounds might be proven for an E.C.T. algorithm. Now, for the case of uniform processors when $\ell=m$ the bound given in Theorem 6.2.1 agrees with the one found by Jane W.S. Liu and Liu C.L. [LiL1] (see Table 3.5, problem 3); also the bounds given in Theorem 6.2.5 are more informative and in most cases better than the ones proven by the above mentioned authors [LiL1] (see Table 3.7, problem 5) and Gonzalez, Ibara and Sethi [GIS2] (see Table 3.8, problems 1 and 2). Finally, when λ or b_i relaxes to one (1) and ℓ to m, our model then represents the classical homogeneous multiprocessor model. For such a case, the bounds in Theorems 6.2.1 and 6.2.5 agree with the ones proven by

Graham [Gr,1,2] (see Table 3.5 problems 1 and 2) and Graham [Gr2,3] or Coffman and Sethi [CS2] respectively (see Table 3.7 problems 1 and 2), while the bounds in Theorems 5.4.1 or 5.4.2 relax to a slightly better but more informative bound than the one Bruno, Coffman and Sethi [BCS1] have proven (see Table 10, problem 7).

When the mean flow time is chosen as the performance criterion, then there are facts which support the idea for further research using the deterministic analysis. These are that there are not task systems which can cause the performance of the algorithms to deviate from the optimal mean flow time by the values allowed by the worst-case bounds, and the values of the average performance of the P.D. or the Q.A.D. algorithm under the STF, LMST or LTF, LMLT ordering rules respectively as well as the P.D.* and Q.A.D.* algorithms under any pre-set ordering are very close to the optimal value. In particular, more informative bounds as far as the Q.A.D. algorithm under the LTF_{MIN} ordering rule are of extreme importance. This is so because first, its expected performance value on average is not more than 5% different from the corresponding optimal one and second, its time complexity is $O(\max\{mn, n\log_2 n\})$ which compares favourably with the complexity $O(n^3)$ of Bruno's [Br1] optimal algorithm. Such an investigation should be approached by proving better (or more informative) upper or lower values to bound the quantities $\bar{\omega}$ and $\bar{\omega}_{OPT}$ and/or considering the range of difference in the time requirements of the jobs. In addition, the nature of the Q.A.D. algorithm itself might also help towards this investigation. On the other hand, it is also interesting to prove a worst-case bound for Bruno's algorithm when the completion time is considered as the performance criterion. Clearly, a promising bound would make this algorithm even more valuable.

Furthermore, the deterministic analysis appears to be a promising supplementary tool for systems where the expected performance is more meaningful than the guaranteed levels. This is true, since the behaviour

of the algorithms predicted by the worst-case bounds agreed, in most cases, with their behaviour as presented by the average performance results. In particular, the behaviour of the P.D. algorithm for both performance criteria as well as the Q.A.D. algorithm for the mean flow time criterion is exactly the same when we analyse their worst-case bounds or average performance results. For the other algorithms when any one of the two performance criteria is used, there is considerable agreement in their behaviour as can be predicted from the worst-case bounds and presented by the average performance results. This non-perfect agreement is believed to be relevant to the close worst-case bounds found for the P.D.* and Q.A.D* algorithms, and the inherent complexity of Q.A.D. algorithm as far as the completion time criterion is concerned.

However, the above mentioned promising conjecture for the determinisitic analysis needs further work to be fully justified. In actual fact, the queueing network theory or simulation techniques for a more detailed system, based on the abstract model we have considered, must be used in such an investigation to evaluate the expected performance of the algorithms under various ordering rules. Then, we will be in a position to say positively if the deterministic analysis can play a supplementary role for the performance evaluation of systems where the expected behaviour is more meaningful. A work recently published by Schewtman [Sc], alghtough it is not oriented towards the direction as described, it evaluates the expected performance of a number of Priority-Driven resource allocation functions, when the priority list is in various orderings for a multiprogramming system using simulation techniques; the selection of pre-set orderings based on the results presented in [Kr] and [KSS2], where a deterministic analysis was used. His primary conclusion was that both the pre-set ordering and the scheduling algorithm affect the performance of the system, but he did not examine the correlation between the results he found and the corresponding ones proved by deterministic analysis.

From the other point of view, since simple simulation studies on the abstract model can offer us information about the average performance behaviour of an algorithm, there is a question whether they are sufficient so that the deterministic analysis is superfluous rather than a supplementary tool. Actually, we cannot trust such simulation studies without the support of the worst-case bounds because:-

- the rate of increase or decrease of the performance of the algorithms as the system parameters change values would not be known;
- the behaviour of an algorithm when for some operational environments its ranking changes could not be explained properly;
- the chosen configurations of the model as a sample to be simulated might not be sufficient; and finally
- we would not have the ability to work towards optimal or more elaborated heuristic scheduling algorithms.

As a result of the assumptions made for the task systems in this thesis (i.e., considering independent jobs with arbitrary time requirements), one can see that the studied model corresponds to a general purpose heterogeneous multiprocessor model with independent memories. So, it can be used as evidence in the development of a general workload system which could consist of a number of various mini- or micro-computers, each one having one or two extra functions. Moreover, our results could have had immediate applications to several problem areas in operations research, management science, industrial engineering or business administration. An example would be when a substantial typing work has to be carried out in a typing department, where there are typewriters with various capabilities and typists with different abilities for different kind of typing work.

An extension of the computing system, just mentioned above, would be a system with functionally dedicated processors, which could correspond to a so-called cluster network. A multiprocessor model corresponding to that

type of network has already been studied by Jane W.S. Liu and C.L. Liu [LiL3] using deterministic analysis. However, further investigation is needed for various heuristic pre-set orderings, since the idea to use such computing networks in the future is under great consideration.

Finally, we may conclude that the deterministic scheduling theory although it is always needed to predict the behaviour of systems where a guaranteed level of performance must be provided, it may be a supplementary tool for systems where the expected behaviour is more meaningful. In addition, because of the importance of producing informative worst-case bounds, we must always try to include in the analysis as many of the critical parameters, which affect the performance of the algorithm, as possible.

REFERENCES

[ACD] Adam T.L., K.M. Chandy and J.R. Dickson, *"A comparison of list schedules for parallel processing systems"*, Communications of ACM, Vol.17, No.12, (Dec.1974), 685-690.

[AUH] Aho A.V., J.E. Hopcroft and J.D. Ullman, *"The Design and Analysis of Computer Algorithms"*, Addison-Wesley, Reading, Mass., 1974.

[Ba] Baer J.L., *"Optimal scheduling on two processors of different speeds"*, Computer Architectures and Networks, 27-45, E. Gelenbe and R. Mahl (Eds.), North Holland Pub. Company,1974.

[Bak] Baker K., *"Introduction to Sequencing and Scheduling"*, John Wiley & Sons, 1974.

[BCS1] Bruno J., E.G. Coffman, Jr. and R. Sethi, *"Scheduling independent tasks to reduce mean finishing time"*, Communications of ACM, Vol.17, No.7 (1974), 382-387.

[BCS2] Bruno J., E.G. Coffman, Jr. and R. Sethi, *"Algorithms for minimising mean flow time"*, Proc. of IFIPS Congress, Aug.1974, North Holland Pub.Co., 504-510.

[Br1] Bruno J., *"A scheduling algorithm for minimsing mean flow time"*, Tech. Report No.141, Computer Science Dept., The Pennsylvania State University.

[Br2] Bruno J., *"Mean weighted flow-time criterion"*, in Computer and Job-Shop Scheduling Theory, 101-137, E.G. Coffman, Jr.(Ed.), John Wiley & Sons, 1976.

[CD] Coffman E.G. Jr. and P.J. Denning, *"Operating Systems Theory"*, Prentice-Hall, Englewood Cliffs, N.J. 1973.

[CG] Coffman E.G. Jr. and R.L. Graham, *"Optimal scheduling for two processor systems"*, Acta Informatica, Vol.1, 3 (1972), 200-213.

[Ch] Chen N.F., *"An analysis of scheduling algorithms in multiprocessor computing systems"*, Ph.D. Thesis, May 1975, Univ. of Illinois at Urbana-Champaign.

[Cl] Clark D., *"Scheduling independent tasks on non-identical parallel machines to minimise mean flow-time"*, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, June 1974.

[CLi] Chen N.F. and C.L. Liu, *"On a class of scheduling algorithms for multiprocessor computing systems"*, Proceedings of the 1974 Sagamore Computer Conf. on Parallel Processing (1974), 1-16.

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

[Co] Coffman E.G., Jr. (Ed.), *"Computer and Job-Shop Scheduling Theory"*, John Wiley & Sons, Inc., 1976.

[Coo] Cook S.A., *"The complexity of theorem proving procedures"*, Proceeding 3rd ACM Symposium on Theory of Computing, 1971, 151-158.

[CS1] Coffman E.G., Jr. and R. Sethi, *"Algorithms minimising mean flow-time - Schedule length properties"*, Acta Informatica, Vol.6, No.1 (1976), 1-14.

[CS2] Coffman E.G., Jr. and R. Sethi, *"A generalised bound on LPT sequencing"*, Proceedings of International Symposium on Computer Performance Modelling, Measurement and Evaluation, March 1976, 306-310.

[EN1] Evans D.J. and I.A. Newman, *"How many minis make a midi?"* Computer Education, Nov.1975, 24-25.

[EN2] Evans D.J. and I.A. Newman, *"Usage considerations in multiple processor systems"*, Software World, Vol.5, Nos. 9 & 10, 69-80.

[F1] Farber D., *"Networks: an introduction"*, Datamation, Apr. 1972, 36-39.

[F2] Farber D., *"A ring network"*, Datamation, Feb. 1975, 45-46.

[FKN] Fujii M., T. Kasami and K. Ninomiya, *"Optimal sequence of two equivalent processors"*, SIAM J. on Computing, Vol.17, No.3 (1969), 784_789. Erratum 20, 1 (1971), 141.

[FV] Ferreira R.C. and D. Vojnovic, *"Multiminicomputers: a perspective on the next five years"*, in Infotech's State of Art Report, "Minis versus mainframes", Vol.2, 1978, 129-152.

[G] Gonzalez M.J., Jr., *"Deterministic processor scheduling"*, Computing Surveys, Vol.9, No.3 (Sept. 1977), 173-204.

[GG1] Garey M.R. and R.L. Graham, *"Bounds on scheduling with limited resources"*, Operating Systems Review, Vol.7, No.4 (1973), 104-111.

[GG2] Garey M.R. and R.L. Graham, *"Bounds for multiprocessor scheduling with resource constraints"*, SIAM J. on Computing, Vol.4, No.2 (Jun 1975), 187-200.

[GGJ] Garey M.R., R.L. Graham and D.S. Jonson, *"Performance guarantees for scheduling algorithms"*, Operations Research Vol.26, No.1 (Jan-Feb 1978), 3-21.

[GGJY] Garey M.R., R.L. Graham, D.S. Jonson and A.C.-C. Yao, *"Resource constrained scheduling as generalised bin packing"*, Journal of Combinatorial Theory (A), 21(1976), 257-298.

[GIS] Gonzalez T., O.H. Iraba and S. Sahni, *"Bounds for LPT schedules on uniform processors"*, SIAM J. on Computing Vol.5 (1977), 155-166.

[GJ1] Garey M.R. and D.J. Jonson, *"Complexity results for multiprocessor scheduling under resource constraints"*, SIAM J. on Computing Vol.4, No.4, (Dec. 1975), 397-411.

[GJ2] Garey M.R. and D.J. Jonson, *"Approximation algorithms for combinatorial problems: an annotated bibliography"*, in Algorithms and Complexity: New directions and recent results, 41-52, J.F. Traub (Ed.), Academic Press, New York 1976.

[Gr1] Graham R.L., *"Bounds for certain multiprocessing anomalies"*, Bell System Tech. Journal Vol.45, (1966), 1563-1581.

[Gr2] Graham R.L., *"Bounds on multiprocessing timing anomalies"*, SIAM J. on Applied Mathematics, Vol.17, No.2 (1969), 416-429.

[Gr3] Graham R.L., *"Bounds on multiprocessing anomalies and related bin packing algorithms"*, Proceedings of AFIPS Conf. Vol.40 (1972), 205-217.

[Gr4] Graham R.L., *"Bounds on the performance of scheduling algorithms"*, in Computer and Job-Shop Scheduling Theory, 165-227, E.G. Coffman, Jr. (Ed.), John Wiley & Sons, 1976.

[Ha] Hardcastle, A.R.K., *"Multi-minis versus large mainframes"*, in Infotech's State of Art Report "Minis versus mainframes", Vol.2, 1978, 109-120.

[HLS] Horvath E.C., S. Lam and R. Sethi, *"A level algorithm for preemptive scheduling"*, Journal of the ACM, Vol.24, No.1 (Jan. 1977), 32-43.

[Ho1] Horn W.A., *"Single-machine job sequencing with tree-like precedence ordering and linear delay penalties"*, SIAM J. on Applied Mathematics, Vol.23 (1972), 189-202.

[Ho2] Horn W.A., *"Minimising average flow-time with parallel machines"*, Operations Research, Vol.21, (1973), 846-847.

[HoS1] Horowitz E. and S. Sahni, *"Exact and approximate algorithms for scheduling non-identical processors"*, Journal of the ACM, Vol.23, No.2, (1976), 317-327.

[HoS2] Horowitz E. and S. Sahni, *Fundamentals of Computer Algorithms"*, Computer Science Press, Inc., 1978.

[Hu] Hu T.C., *"Parallel sequencing and assembly line problems"*, Operations Research, Vol.9, No.6 (1961), 841-848.

[IK] Ibara O.H. and C.E. Kim, *"Heuristic algorithms for scheduling independent tasks on non-identical processors"*, Journal of the ACM, Vol.24, No.2,(Apr. 1977), 280-289.

[Kaf] Kafura D.G., *"Analysis of scheduling algorithms for a model of multiprocessor computer system"*, Ph.D. Thesis, Purdue Univ., West Lafayette, Dec. 1974.

[Karl] Karp R.M., *"Reducibility among combinatorial problems"*, in Complexity of Computer Computation, 85-104, R.E. Miller and J.W. Thatcher (Eds.), Plenum Press, New York, 1972.

[Kar2] Karp R.M., *"On the computational complexity of combinatorial problems"*, Networks, Vol.5, (1975), 45-68.

[Kau] Kaufman M.T., *"An almost-optimal algorithm for the assembly line scheduling problem"*, IEEE Transactions on Computers, Vol.C-23 (1974), 1169-1174.

[Kn] Knuth D., *"The Art of Computer Programming Vol.1"*, "Fundamental Algorithms", Addison-Wesley, Pub.Co., second printing 1976.

[Koh] Kohler W.H., *"A preliminary evaluation of the critical path method for scheduling tasks on multiprocessor systems"*, IEEE Transactions on Computers, Vol.C-24, No.12 (Dec.1975), 1235-1238.

[Kr] Krause K.L., *"Analysis of computer scheduling with memory constraints"*, Ph.D. Thesis, Purdue Univ.,West Lafayette, Dec. 1973.

[Kre] Kreyszig E., *"Introductory Mathematical Statistics: Principles and Methods"*, J. Wiley & Sons, Inc., 1970.

[KS1] Kafura D.G. and V.Y. Shen, *"Scheduling independent processors with different storage capacities"*, Proceedings of ACM 1974 Annual Conf., Nov.1974, 161-166.

[KS2] Kafura D.G. and V.Y. Shen, *"Task scheduling on a multiprocessor system with independent memories"*, SIAM J. on Computing, Vol.6, No.1 (March 1977), 167-187.

[KS3] Kafura D.G. and V.Y. Shen, *"An algorithm to design the memory configuration of a computer network"*, Journal of the ACM, Vol.25, No.3 (July 1978), 365-377.

[KSS1] Krause K.L., V.Y. Shen and Schwetman H.D., *"A task scheduling algorithm for a multiprogramming computer system"*, Proc. Fourth Symp. Operating Systems Principles, Oct. 1973, 112-118.

[KSS2] Krause K.L., V.Y. Shen and Schwetman H.D., *"Analysis of several task scheduling algorithms for a model of multiprogramming computer systems"*, Journal of the ACM Vol.22, No.4 (Oct. 1975), 522-550.

Errata: J. of the ACM, Vol.24, No.3 (July, 1977), 527.

[Li1] Liu C.L., *"Optimal scheduling on multiprocessor computing systems"*, Proceedings, Conf. on Switching and Automata Theory, Maryland, Oct. 1972, 155-160.

[Li2] Liu C.L., *"Deterministic job scheduling in computing systems"*, Report, Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, 1976.

[LiL1] Liu J.W.S. and C.L. Liu, *"Bounds on scheduling algorithms for heterogeneous computing systems"*, Proceedings of the 1974 IFIP Congress, 349-353.

[LiL2] Liu J.W.S. and C.L. Liu, *"Performance analysis of heterogeneous multiprocessor computing systems"*, in Computer Architecture and Networks, 331-343, E. Gelenbe and R. Mahl (Eds.), North Holland Publishing Co., 1974.

[LiL3] Liu J.W.S. and C.L. Liu, *"Performance analysis of multiprocessor systems containing functional dedicated processors"*, Acta Informatica, Vol.10 (1978), 95-104.

[LS] Lam S. and R. Sethi, *"Worst-case analysis of two scheduling algorithms"*, SIAM J. on Computing, Vol.6, No.3 (Sept.1977), 518-536.

[LY] Liu J.W.S. and A. Yang, *"Optimal scheduling of independent tasks on heterogeneous systems"*, Proceedings of the ACM, 1974, 38-45.

[MC1] Muntz R.R. and E.G. Coffman, Jr., *"Optimal preemptive scheduling on two-processor systems"*, IEEE Transactions on Computers, Vol.C-18, No.11.(1969), 1014-1020.

[MC2] Muntz R.R. and E.G. Coffman, Jr., *"Preemptive scheduling of real time tasks on multiprocessor systems"*, Journal of the ACM, Vol.17, No.2 (1970), 324-338.

[McN] McNaughton R., *"Scheduling with deadlines and loss functions"*, Management Science Vol.6, No.1 (Oct. 1959), 1-12.

[MF75] Minicomputer Forum 1975, *"Multi-minicomputer systems"*, Proceedings of Minicomputer Forum 1975, London, 44-54.

[Sa] Sahni S.K., *"Algorithms for scheduling independent tasks"*, Journal of the ACM, Vol.23, No.1 (Jan.1976), 116-127.

[SaH] Sahni S.K., and E. Horowitz, *"Combinatorial algorithms: Reducibility and Approximation"*, Operations Research, Vol.26, No.5, (Sept-Oct. 1978), 718-759.

[Sc] Schewtman D.H., *"Job scheduling in multiprogrammed computer systems"*, Software-Practice and Experience, Vol.8 (1978), 241-255.

[Sel] Sethi R., *"Algorithms for minimal length schedules"*, in Computer and Job-Shop Scheduling Theory, 51-99, E.G. Coffman, Jr. (Ed.), John Wiley & Sons, 1976.

[Se2] Sethi R., *"Scheduling graphs on two processors"*, SIAM J. on Computing, Vol.5, No.1 (March 1976), 73-82.

[Se3] Sethi R., *"On the complexity of mean flow-time scheduling"*, Maths of Operations Research, Vol.2, No.4 (Nov. 1977), 320-330.

[Sh] Shah V., *"Networking with minis"*, Data Systems, April 1977, 17-20.

[SS] Schindler S. and W. Simonsmeier, *"Scheduling independent tasks on different processors"*, Proceedings, 2nd Annual Computer Science Conf., Detroit, Feb. 1974.

Bericht Nr. 74-05, Technische Universitat Berlin, Fachbereich 20 - Kybernetik, May 1974.

[Ul] Ullman J.D., *"Polynomial complete scheduling problems"*, Operating Systems Review, Vol.7, No.4 (1973), 96-101.

[U2] Ullman J.D., *"Complexity of sequencing problems"*, in Computer and Job-Shop Scheduling Theory, 139-164, E.G. Coffman, Jr. (Ed.), John Wiley & Sons, 1976.

[Wh] White C.H. (Ed.), *"Multiminicomputers versus mainframes"* in Infotech's State of Art Report "Minis versus mainframes", Vol.1, 1978, 105-115.

[WL] Wulf W.A. and R. Levin, *"A local network"*, Datamation, Feb.1975, 47-50.

[Wu] Wulf W.A., *"Minicomputer complexes: progress and prospects"* in Computer Science and Scientific Computing, 283-304, J.M. Ortega (Ed.), Academic Press, 1976.

[Ya] Yao A.C., *"On scheduling unit-time tasks with limited resources"*, Proceedings of the Sagamore Conf. on Parallel Processing, 1974, 17-36.

APPENDIX I

HOMOGENEOUS MULTIPROCESSOR SYSTEM WITH INDEPENDENT MEMORIES - WORST-CASE BOUNDS OF P.D. AND P.D.* SCHEDULING ALGORITHMS - MEAN FLOW TIME PERFORMANCE CRITERION

Consider a task system with n independent jobs to be scheduled on a homogeneous multiprocessor system with independent memories and a fixed number m of processors. Let $\bar{\omega}_{PD}$ be the mean flow time of the schedule constructed by the P.D. algorithm, when the priority list is formed by a heuristic ordering procedure, and $\bar{\omega}_{OPT}$ be the mean flow time of an optimal schedule.

For the cases where the priority list is constructed by the RAND, LMF, LTF or LMLT ordering rule, $\bar{\omega}_{PD}$ and $\bar{\omega}_{OPT}$ are given as follows:

$$\bar{\omega}_{PD} = \sum_{i=1}^{m} (n_i' t_1^{G_i} + (n_i'-1)t_2^{G_i} + \ldots + t_{n_i'}^{G_i}) \quad \dagger$$

and

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m} (t_1^{G_i} + t_2^{G_i} + \ldots + t_{n_i'}^{G_i}) \ .$$

Following a similar analysis with the one used previously to prove Theorem 5.3.1 the next theorem is established.

Theorem I.1: If the priority list is formed by the RAND, LMF, LTF or LMLT ordering rule, then

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} \leq n'_{max} \ ,$$

where

$$n'_{max} = \max_{1 \leq i \leq m} \{n_i'\} \ .$$

The following example shows that the worst-case performance of the P.D. algorithm under the ordering rules mentioned in Theorem I.1 can be approached asymptotically.

Example I.1: Let the task system $(J,\{m_j\},\{t_j\})$ be defined by:

J_{ir+1}: $(|P_{i+1}|,X)$, $0 \leq i \leq m-1$

J_{ir+j}: $(|P_{i+1}|,\varepsilon)$, $0 \leq i \leq m-1$, $2 \leq j \leq r$,

---

† *Here,* $\tau_j=\sigma_j=t_j$, $1 \leq j \leq n$. *Similarly* $\tau_j^{F_i}=\sigma_j^{F_i}=t_j^{F_i}$ and $\tau_j^{G_i}=\sigma_j^{G_i}=t_j^{G_i}$.

where $n=mr$, $n_i=r$ for $i=1(1)m$, $\{\varepsilon,X\}\in R^+$ and $\varepsilon<<X$.

Clearly, the priority list $L_1=(J_1,J_2,\ldots,J_n)$ is in LMF or LMLT ordering, whilst the priority list $L_2=(J_1,J_{r+1},J_{2r+1},\ldots,J_{(m-1)r+1},J_2,J_3,\ldots,J_r,J_{r+2},\ldots,J_{2r},\ldots,J_{(m-1)r+2},\ldots,J_{mr})$ is in LTF ordering. We can choose as RAND ordering any of these two lists. The schedule resulting from the priority list L_1 or L_2, when the P.D. algorithm is used, is shown in Fig.I.1. The corresponding optimal schedule is given in Fig.I.2.

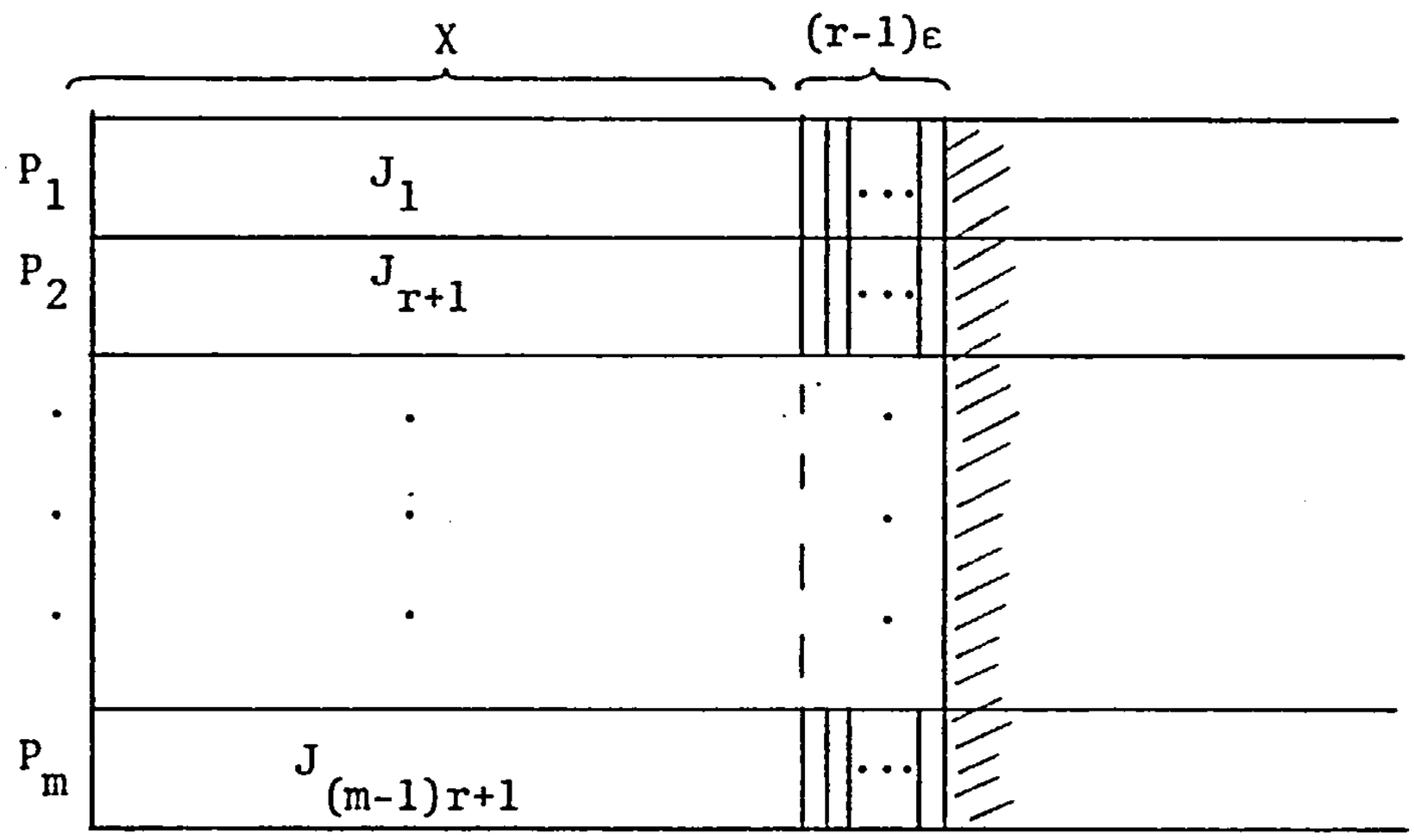


FIGURE I.1: Worst-case schedule to illustrate Theorem I.1.

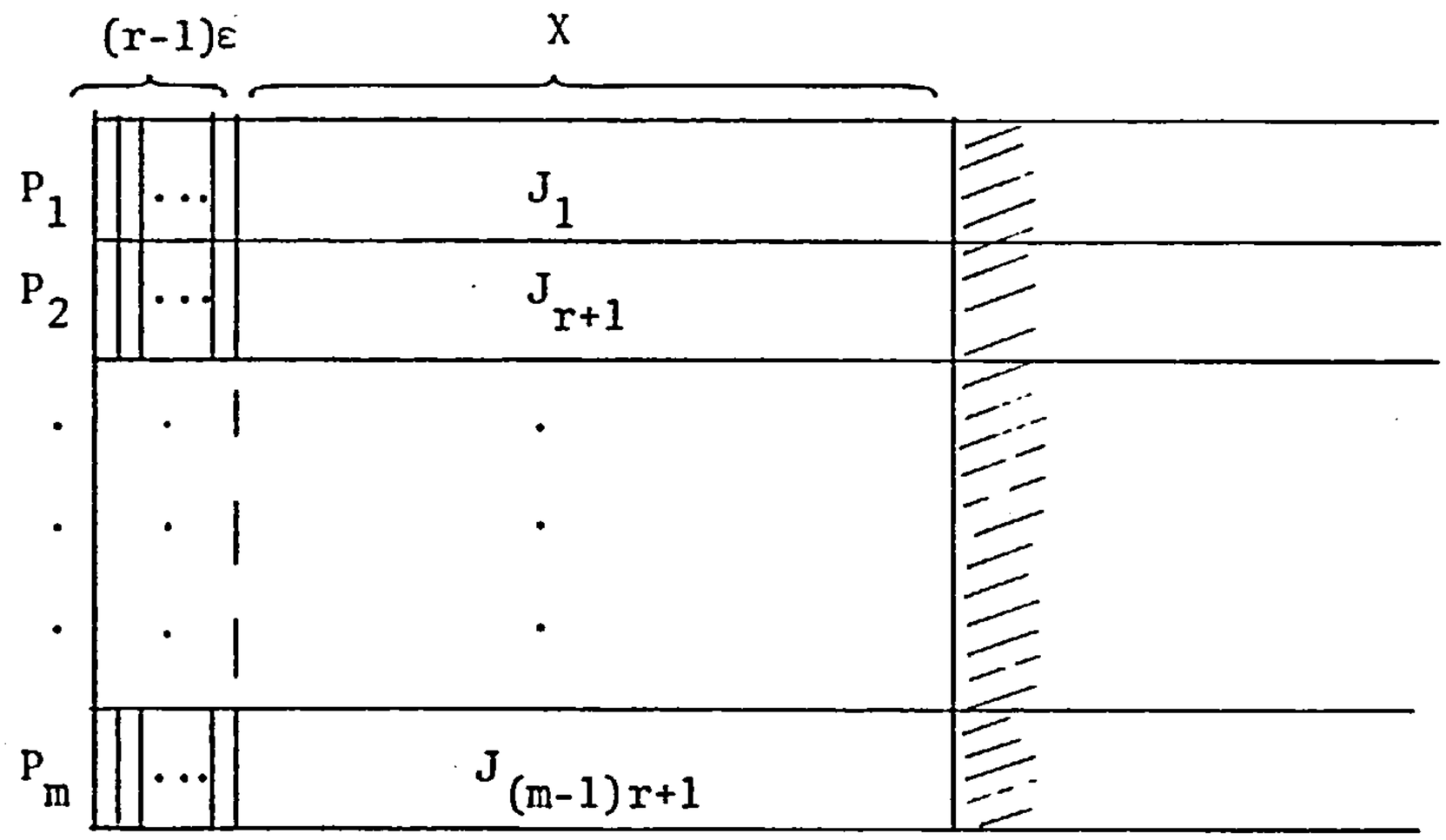


FIGURE I.2: Optimal schedule to illustrate Theorem I.1

The ratio of the mean flow times of these two schedules is

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} = \frac{\sum_{i=1}^{m} [rX+(r-1)\varepsilon+\ldots+2\varepsilon+\varepsilon]}{\sum_{i=1}^{m} [r\varepsilon+(r-1)\varepsilon+\ldots+2\varepsilon+X]} = \frac{mrX+f_1(\varepsilon)}{mX+f_2(\varepsilon)}$$

But,

$$\lim_{\varepsilon\to 0} \frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} = \frac{mrX}{mX} = r = n'_{max} ,$$

which is the value predicted by Theorem I.1.

Equality in Theorem I.1 holds only if $n'_{max}=1$.

Furthermore, let the priority list be in STF ordering. Then, $\bar{\omega}_{PD}$ and $\bar{\omega}_{OPT}$ are given by

$$\bar{\omega}_{PD} = \sum_{i=1}^{m} (n_i! t_{n_i!}^{G_i} + (n_i!-1) t_{(n_i!-1)}^{G_i} + \ldots + t_1^{G_i}) \qquad (I.1)$$

and

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m} [(t_1^{G_i}+t_2^{G_i}+\ldots+t_{\nu_i}^{G_i}) + 2(t_{\nu_i+1}^{G_i}+\ldots+t_{2\nu_i}^{G_i}) + \ldots + \left\lceil \frac{n_i!}{\nu_i} \right\rceil (t_{\left\lfloor \frac{n_i!}{\nu_i} \right\rfloor \nu_i+1}^{G_i} + \ldots + t_{\left\lceil \frac{n_i!}{\nu_i} \right\rceil \nu_i}^{G_i})] \qquad (I.2)$$

or

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m} (t_1^{G_i}+t_2^{G_i}+\ldots+t_{n_i!}^{G_i}) , \qquad (I.3)$$

where $\nu_i = \max_{J_j \in G_i} \{\ell_j\}$, $t_{n_i!}^{G_i} \leq t_{n_i!-1}^{G_i} \leq \ldots \leq t_1^{G_i}$ and $t_{n_i!+1}^{G_i} = \ldots = t_{\left\lceil \frac{n_i!}{\nu_i} \right\rceil \nu_i}^{G_i} = 0$ for i=1(1)m.

Consequently, using a similar analysis as in Theorem 5.3.3 the following theorem is derived.

Theorem I.2: If the priority list is formed by the STF ordering rule, then

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} \leq \min\left\{ \max_{1\leq i\leq m} \left\{ \nu_i - \frac{(\nu_i-1)}{2\left\lceil \frac{n_i!}{\nu_i} \right\rceil} \right\} , \left(\frac{n'_{max}+1}{2}\right) \right\} .$$

Obviously, the value of the second factor is less than the value of the first if $n'_{max} \leq \left\lfloor \frac{3m}{2} \right\rfloor - 1$. Equality holds only if $n'_{max}=1$.

Finally, let the priority list be ordered by the LMST rule. Then, $\bar{\omega}_{PD}$ and $\bar{\omega}_{OPT}$ are bounded respectively by

$$\bar{\omega}_{PD} < \sum_{i=1}^{m-1} [(k+1)t_1^{F_i} + (k+2)t_2^{F_i} + \ldots + (k+n_i)t_{n_i}^{F_i}] + (t_1^{F_m} + 2t_2^{F_m} + \ldots + n_m t_{n_m}^{F_m}) \quad (I.4)$$

and

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m} \sum_{r=1}^{\lceil n_i/i \rceil} [r(t_{(r-1)i+1}^{F_i} + \ldots + t_{ri}^{F_i})] \quad (I.5)$$

or

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m} (t_1^{F_i} + t_2^{F_i} + \ldots + t_{n_i}^{F_i}), \quad (I.6)$$

where $k = \max_{1 \leq i \leq m} \{n_i! - n_i\}$, $t_1^{F_i} \geq t_2^{F_i} \geq \ldots \geq t_{n_i}^{F_i}$ and $t_{n_i+1}^{F_i} = \ldots = t_{\lceil \frac{n_i}{i} \rceil i}^{F_i} = 0$ for 1(1)m.

Again, using a similar analysis as in Theorem 5.3.5 the following theorem is established.

Theorem I.3: If the priority list is formed by the LMST ordering procedure, then we have

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{\max\left\{\max_{1 \leq i \leq m-1}\left\{\max_{1 \leq r \leq \lceil \frac{n_i}{i} \rceil}\left\{\frac{k}{r} + i - \frac{(i-1)}{2r}\right\}\right\}, \left\lceil m - \frac{(m-1)}{2\lceil \frac{n_m}{m} \rceil}\right\rceil\right\}\right.$$

$$\left. \max\left\{\max_{1 \leq i \leq m-1}\left\{k + \frac{(n_i+1)}{2}\right\}, \left(\frac{n_m+1}{2}\right)\right\}\right\} .$$

Moreover, the bound $\left\lceil m - \frac{(m-1)}{2\lceil \frac{n_m}{m} \rceil}\right\rceil$ is a best possible if k=0. This can be realised by considering the next example.

Example I.2: Let the task system $(J, \{m_j\}\{t_j\})$ be defined by:

$$J_j: (|P_j|, X), \quad 1 \leq j \leq m-1$$

$$J_j: (|P_m|, 1), \quad m \leq j \leq m+X-1 ,$$

where $X \in Z^+$, $X = rm = n_m$ and $n = m+X-1$.

Obviously, the list $L=(J_1, J_2, \ldots, J_n)$ is in LMST ordering. The schedule resulting from this list, when the P.D. algorithm is used, is shown in Fig.I.3,

whereas the corresponding optimal is given in Fig.I.4.

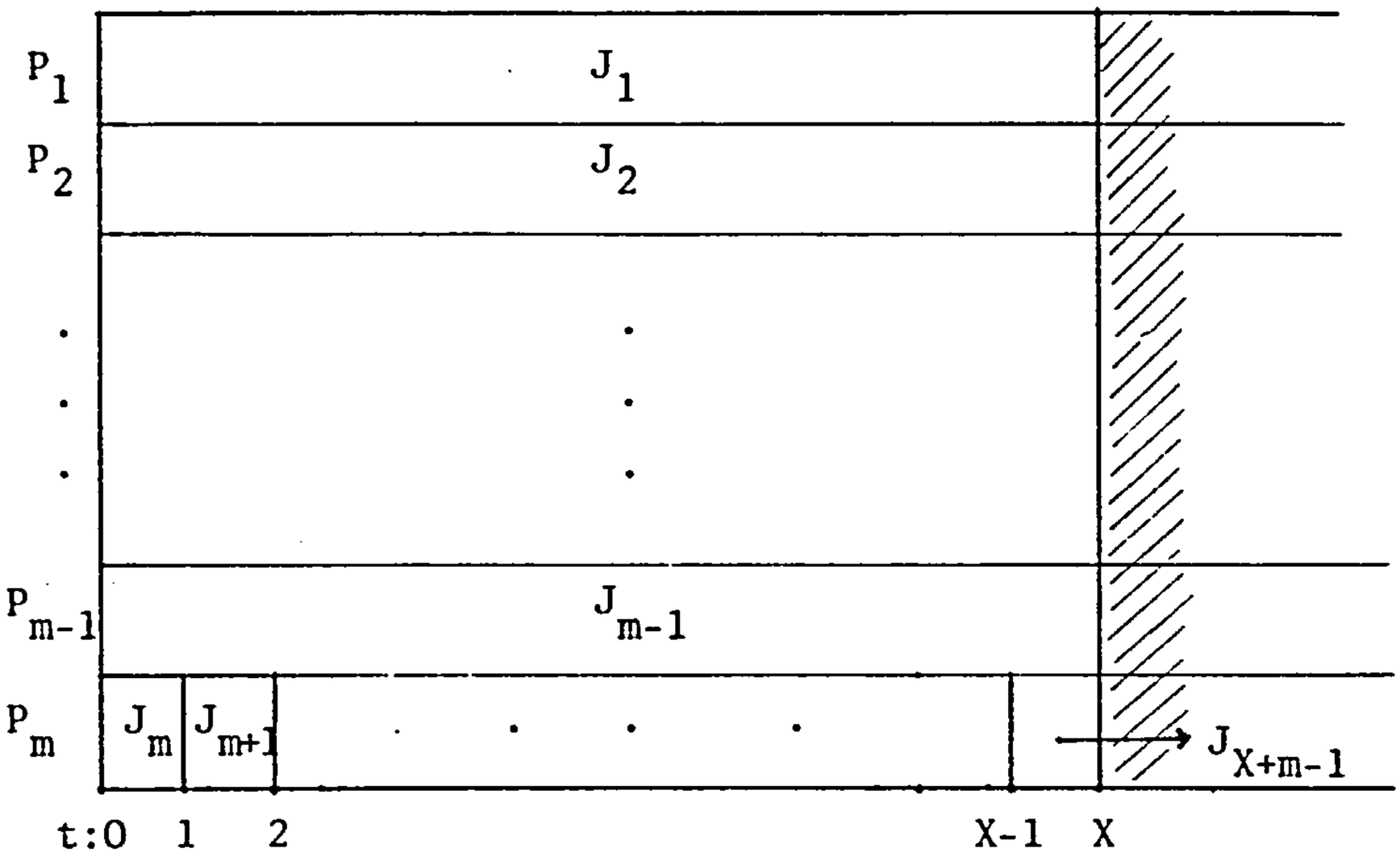


FIGURE I.3: Worst-case schedule to illustrate the bound $\left(m - \frac{(m-1)}{2\left\lceil \frac{n_m}{m} \right\rceil}\right)$ in Theorem I.3

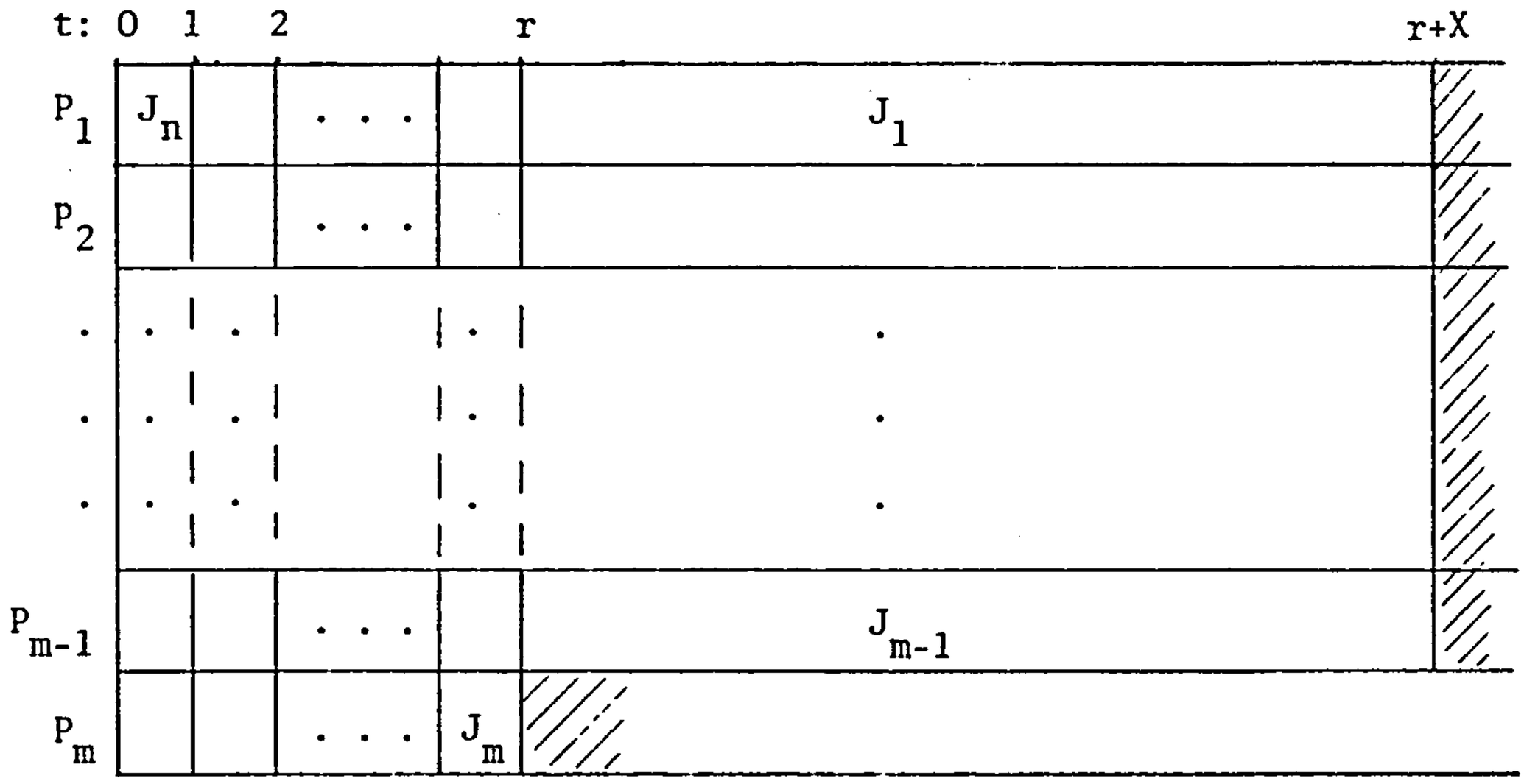


FIGURE I.4: Optimal schedule for the given task system in Example I.2

The ratio of the mean flow times of these two schedules is:

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} = \frac{m[2(m-1)+(n_m+1)]}{2(m^2-1)+(n_m+m)} \quad ,$$

which by our choice of n_m, can be made to approach $\left(m - \frac{(m-1)}{2\left\lceil \frac{n_m}{m} \right\rceil}\right)$ arbitrarily close.

The bound of theorem I.3, for k=0 becomes

$$\frac{\bar{\omega}_{PD}}{\bar{\omega}_{OPT}} < \min\left\{ \max_{\left\lceil \frac{m+1}{2} \right\rceil \leq i \leq m} \left\{ i - \frac{(i-1)}{2\left\lceil \frac{n_i}{i} \right\rceil} \right\} \quad , \quad \left(\frac{n_{max}+1}{2}\right) \right\} \quad .$$

In addition, if k=0 and $n_i \geq n_{i+1}$, $1 \leq i \leq m-1$, then, $\bar{\omega}_{PD}=\bar{\omega}_{OPT}$ (i.e., LMST ordering produces optimal schedules).

Now, let $\bar{\omega}_{PD*}$ be the mean flow time of the schedules constructed by the P.D.* algorithm, when the priority list is formed by a heuristic ordering procedure.

Clearly, when the priority list is formed by the RAND, LTF or STF ordering rule then, $\bar{\omega}_{PD*}$ and $\bar{\omega}_{OPT}$ are bounded to their corresponding values in the set of inequalities (I.1)-(I.3). Therefore, the following theorem can be established.

Theorem I.4: If the priority list is formed by the RAND, LTF or STF ordering rule, then

$$\frac{\bar{\omega}_{PD*}}{\bar{\omega}_{OPT}} \leq \min\left\{ \max_{1 \leq i \leq m} \left\{ \nu_i - \frac{(\nu_i - 1)}{2\left\lceil \frac{n_i}{\nu_i} \right\rceil} \right\} \quad , \quad \left(\frac{n'_{max}+1}{2}\right) \right\} \quad .$$

If the priority list is in the RAND or LTF ordering then the bound $\max_{1 \leq i \leq m} \left\{ \nu_i - \frac{(\nu_i - 1)}{2\left\lceil \frac{n'_i}{\nu_i} \right\rceil} \right\}$ is a best possible one. This can be realised by considering the Example I.2. On the other hand, if the priority list is in STF ordering, then the P.D. and P.D.* algorithms produce identical schedules.

However, when the priority list is in a LMF, LMLT or LMST ordering then the values of $\bar{\omega}_{PD*}$ and $\bar{\omega}_{OPT}$ are bounded as their corresponding ones in the sets of inequalities (I.1)-(I.3) as well as (I.4)-(I.6). Such a behaviour of the quantities $\bar{\omega}_{PD}$ and $\bar{\omega}_{OPT}$ leads to the establishment of the next theorem.

Theorem I.5: If the priority list is formed by the LMF,LMLT or LMST ordering procedure, then

$$\frac{\bar{\omega}_{PD*}}{\bar{\omega}_{OPT}} < \min\left\{\max\left\{\max_{1\leq i\leq m-1}\left\{\max_{1\leq r\leq\left\lceil\frac{n_i}{i}\right\rceil}\left\{\frac{k}{r}+i-\frac{(i-1)}{2r}\right\}\right\},\ \left\{m-\frac{(m-1)}{2\left\lceil\frac{n_m}{m}\right\rceil}\right\}\right\}\right.,$$

$$\max\left\{\max_{1\leq i\leq m-1}\left\{k+\frac{(n_i+1)}{2}\right\},\ \left(\frac{n_m+1}{2}\right)\right\},$$

$$\left.\max_{1\leq i\leq m}\left\{\nu_i-\frac{(\nu_i-1)}{2\left\lceil\frac{n_i}{\nu_i}\right\rceil}\right\},\ \left(\frac{n'_{max}+1}{2}\right)\right\}.$$

The bounds $\left\{m-\frac{(m-1)}{2\left\lceil\frac{n_m}{m}\right\rceil}\right\}$ and $\max_{1\leq i\leq m}\left\{\nu_i-\left(\frac{\nu_i-1}{2\left\lceil\frac{n'_i}{\nu_i}\right\rceil}\right)\right\}$ are best possible ones.

This can also be realised by considering the Example I.2. However, schedules with k=0 and $n_i\geq n_{i+1}$, $1\leq i\leq m-1$, which have been produced by the P.D.* algorithm under the LMF,LMLT or LMST ordering rule are optimal ones.

APPENDIX II

HOMOGENEOUS MULTIPROCESSOR SYSTEM WITH INDEPENDENT MEMORIES - WORST-CASE BOUNDS OF Q.A.D. AND Q.A.D* SCHEDULING ALGORITHMS - MEAN FLOW TIME PERFORMANCE CRITERION

Consider a task system $(J,\{m_j\},\{t_j\})$ with n independent jobs to be scheduled on a homogeneous multiprocessor system with independent memories and a fixed number m of processors. Let $\bar{\omega}_{QAD}$ be the mean flow of the schedule constructed by the Q.A.D. algorithm, when the priority list is formed by a heuristic ordering procedure, and $\bar{\omega}_{OPT}$ be the mean flow time of an optimal schedule.

For the cases where the priority list is constructed by the RAND, LMF, STF or LMST ordering rule, then following a similar analysis as the one used previously in Theorem 5.5.1, the next theorem can be established.

Theorem II.1: If the priority list is in a RAND, LMF STF or LMST ordering, then

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} < n'_{max} \quad .$$

Moreover, there are examples which approach the bound n'_{max} asymptotically. In fact, Example I.1 in Appendix I is one of them. However, the priority lists of that example must be modified so that the LMF, LMST or STF orderings are presented.

Furthermore, let the priority list be in LTF ordering. Then, the values of $\bar{\omega}_{QAD}$ and $\bar{\omega}_{OPT}$ are given by

$$\bar{\omega}_{QAD} = \sum_{i=1}^{m} (n_i' t_{n_i'}^{G_i} + (n_i'-1) t_{n_i'-1}^{G_i} + \ldots + t_1^{G_i}) \qquad \text{(II.1)}$$

and

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m} [(t_1^{G_i}+\ldots+t_{\nu_i}^{G_i}) + 2(t_{\nu_{i+1}}^{G_i}+\ldots+t_{2\nu_i}^{G_i}) + \ldots + \left\lceil \frac{n_i'}{\nu_i} \right\rceil (t_{\left\lfloor \frac{n_i'}{\nu_i} \right\rfloor \nu_i+1}^{G_i} + \ldots + t_{\left\lceil \frac{n_i'}{\nu_i} \right\rceil \nu_i}^{G_i})] \qquad \text{(II.2)}$$

or

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m} (t_1^{G_i} + t_2^{G_i} + \ldots + t_{n_i'}^{G_i}) \quad , \qquad \text{(II.3)}$$

where $\nu_i = \max_{J_j \in G_i} \{\ell_j\}$, $t_1^{G_i} \geq t_2^{G_i} \geq \ldots \geq t_{n_i'}^{G_i}$ and $t_{n_i'+1}^{G_i} = \ldots = t_{\left\lceil \frac{n_i'}{\nu_i} \right\rceil \nu_i}^{G_i} = 0$ for i=1(1)m.

Consequently, the following theorem can be derived:

Theorem II.2: If the priority list is in LTF ordering, then

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} \leq \min\left\{ \max_{1\leq i\leq m} \left\{ \nu_i - \frac{(\nu_i - 1)}{2\left\lceil \frac{n_i}{\nu_i} \right\rceil} \right\}, \left(\frac{n'_{max}+1}{2}\right) \right\} .$$

Equality holds only if $n'_{max}=1$.

Now, if the priority list is in LMLT ordering, then the values of $\bar{\omega}_{QAD}$ and $\bar{\omega}_{OPT}$ are bounded as follows:

$$\bar{\omega}_{QAD} \leq \sum_{i=1}^{m} (t_1^{F_i} + 2t_2^{F_i} + \ldots + n_i t_{n_i}^{F_i}) \tag{II.4}$$

and

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m} \sum_{r=1}^{\lceil n_i/i \rceil} [r(t_{(r-1)i+1}^{F_i} + \ldots + t_{r_i}^{F_i})] \tag{II.5}$$

or

$$\bar{\omega}_{OPT} \geq \sum_{i=1}^{m} (t_1^{F_i} + t_2^{F_i} + \ldots + t_{n_i}^{F_i}) , \tag{II.6}$$

where $t_1^{F_i} \geq t_2^{F_i} \geq \ldots \geq t_{n_i}^{F_i}$ and $t_{n_i+1}^{F_i} = \ldots = t_{\left\lceil \frac{n_i}{i} \right\rceil i}^{F_i} = 0$ for i=1(1)m.

As a result, the following theorem can be established.

Theorem II.3: If the priority list is in LMLT ordering, then

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} \leq \min\left\{ \max_{\left\lceil \frac{m+1}{2} \right\rceil \leq i\leq m} \left\{ i - \frac{(i-1)}{2\left\lceil \frac{n_i}{i} \right\rceil} \right\}, \left(\frac{n_{max}+1}{2}\right) \right\} .$$

Schedules with $n'_i=n_i$, $1\leq i\leq m$, which are produced by the Q.A.D. algorithm under the LMLT ordering rule are optimal ones. Moreover, the bound

$$\max_{\left\lceil \frac{m+1}{2} \right\rceil \leq i\leq m} \left\{ i - \frac{(i-1)}{2\left\lceil \frac{n_i}{i} \right\rceil} \right\}$$ is a best possible one. This can be realised by considering the following example.

Example II.1: Let the task system $(J,\{m_j\},\{t_j\})$ be defined by:

$$J_{s_{i-1}+j} : (|P_i|, \varepsilon) \qquad \text{for } 1 \leq i \leq m-1, \qquad 1 \leq j \leq n_i \text{ , } s_i = s_{i-1} + n_i, \; s_0 = 0$$

$$J_{n-n_m+j} : (|P_m|, 1) \qquad \text{for } 1 \leq j \leq n_m \text{ ,}$$

where ε is a very small positive quantity, $n_m \leq n_i$, $1 \leq i \leq m-1$, $n_i = n_{i+1}$, $1 \leq i \leq m-2$ and $n_m = r \times m$ (n_m is a multiple of the number of processors).

Clearly, the priority list $L=(J_1, J_2, \ldots, J_n)$ is in LMLT ordering. The schedule resulting from this list, when the Q.A.D. algorithm is used, is shown in Fig.II.1, whereas the corresponding optimal is given in Fig.II.2.

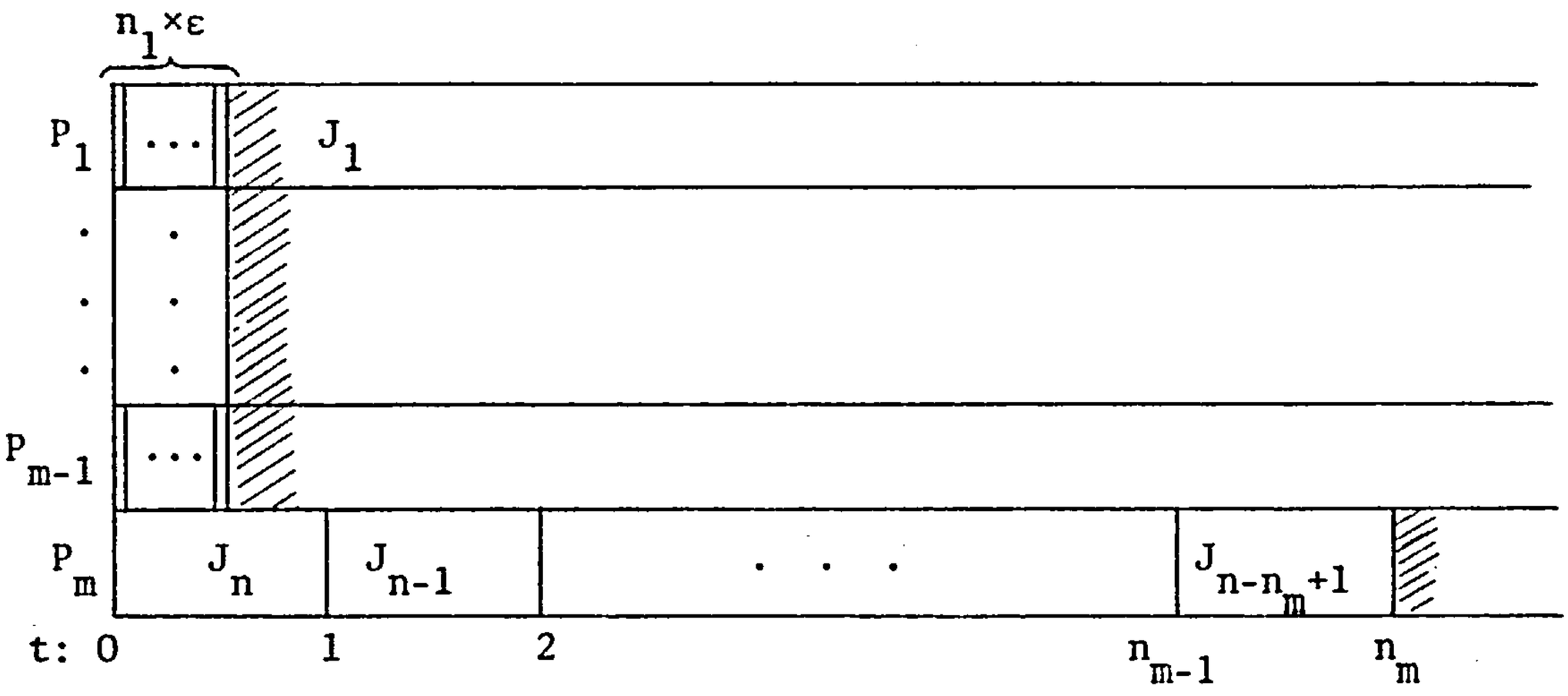


FIGURE II.1: The schedule of the Q.A.D. algorithm for the priority list L given in Example II.1

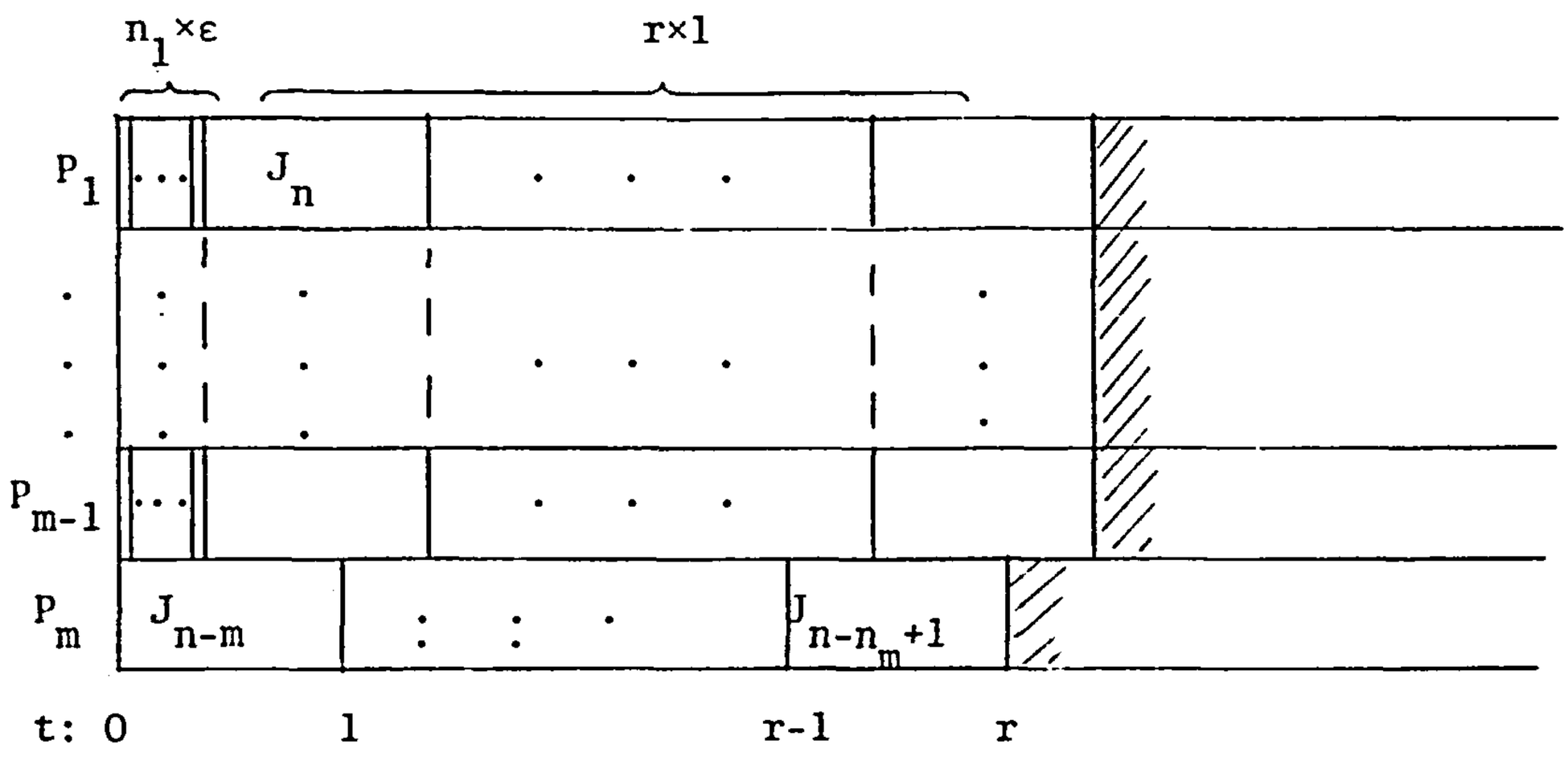


FIGURE II.2: An optimal schedule for the given task system in Example II.1

The ratio of the mean flow times of these two schedules is:

$$\frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} = \frac{\frac{rm(rm+1)}{2} + f_1(\varepsilon)}{m(\frac{r(r+1)}{2})+f_2(\varepsilon)} \quad .$$

But,

$$\lim_{\varepsilon \to 0} \frac{\bar{\omega}_{QAD}}{\bar{\omega}_{OPT}} = \frac{rm+1}{r+1} = \frac{m(r+1)-m}{r+1} = (m - \frac{m}{r+1}) \ ,$$

which by our choice of r can be made to approach $(m - \frac{(m-1)}{2r})$ arbitrarily close.

In the remainder of this appendix, let $\bar{\omega}_{QAD*}$ be the mean flow time of the schedule constructed by the Q.A.D.* algorithm, when the priority list is formed by a heuristic ordering procedure.

Clearly, when the priority list is in a RAND or STF ordering, then $\bar{\omega}_{QAD*}$ obtains the value given by equation (II.1). Therefore, the following theorem can be established.

Theorem II.4: If the priority list is in RAND or STF ordering, then

$$\frac{\bar{\omega}_{QAD^*}}{\bar{\omega}_{OPT}} \leq \min \left\{ \max_{1\leq i\leq m} \left\{ \nu_i - \frac{(\nu_i - 1)}{2\left\lceil \frac{n_i}{\nu_i} \right\rceil} \right\} \ , \ \left(\frac{n'_{max}+1}{2} \right) \right\} \ .$$

There are examples which can approach the bound $\max_{1\leq i\leq m} \left\{ \nu_i - \frac{\nu_i - 1}{2\left\lceil \frac{n'_i}{\nu_i} \right\rceil} \right\}$ asymptotically. Example II.1 is one of them.

On the other hand, when the priority list is in LMF or LMST ordering and the Q.A.D.* algorithm is used to construct the schedules, the values of $\bar{\omega}_{QAD^*}$ and $\bar{\omega}_{OPT}$ are bounded to their corresponding values in the inequalities (I.1)-(I.3) and (I.4)-(I.6) (see Appendix I). As a result, the following theorem can be proved.

Theorem II.5: If the priority list is in LMF or LMST ordering, then

$$\frac{\bar{\omega}_{QAD^*}}{\bar{\omega}_{OPT}} < \min \left\{ \max \left\{ \max_{1\leq i\leq m-1} \left\{ \max_{1\leq r\leq \left\lceil \frac{n_i}{i} \right\rceil} \left\{ \frac{k}{r} + i - \frac{(i-1)}{2r} \right\} \right\} , \left\{ m - \frac{(m-1)}{2\left\lceil \frac{n_m}{m} \right\rceil} \right\} \right\} \right.$$

$$\left. \max \left\{ \max_{1\leq i\leq m-1} \left\{ k + \frac{(n_i+1)}{2} \right\} \ , \ \left(\frac{n_m+1}{2} \right) \right\} \right\} \ ,$$

$$\max_{1\leq i\leq m} \left\{ \nu_i - \frac{(\nu_i - 1)}{2\left\lceil \frac{n_i}{\nu_i} \right\rceil} \right\} \ , \ \left(\frac{n'_{max}+1}{2} \right) \right\} \ .$$

Moreover, there are examples which can approach the bounds $\max_{1\leq i\leq m} \left\{ \nu_i - \frac{(\nu_i - 1)}{2\left\lceil \frac{n_i}{\nu_i} \right\rceil} \right\}$ and $\left\{ m - \frac{(m-1)}{2\left\lceil \frac{n_m}{m} \right\rceil} \right\}$ asymptotically; again, Example II.1 is one of them.

Now, if the priority list is in LMLT ordering, the value which bounds $\bar{\omega}_{QAD}$ in the inequality (II.4) cannot be improved and so, the worst-case bound of Theorem II.3 is valid even when the Q.A.D.* algorithm is used to

construct the schedules. However, if the priority list is in a LTF ordering, the Q.A.D. and Q.A.D.* algorithm produce identical schedules. Finally, note that schedules with $n_i'=n_i$, $1\leq i\leq m$, which are produced by the Q.A.D.* algorithm while the priority list is in a LMF, LMT or LMST ordering are optimal ones.

APPENDIX III

HOMOGENEOUS MULTIPROCESSOR SYSTEM WITH INDEPENDENT MEMORIES - WORST-CASE BOUNDS OF P.D. SCHEDULING ALGORITHMS - COMPLETION TIME PERFORMANCE CRITERION

Consider a task system $(\mathcal{J},\{m_j\},\{t_j\})$ with n independent jobs to be scheduled on a homogeneous multiprocessor system with independent memories and a fixed number of processors. Let ω_{PD} be the completion time of the schedule constructed by the P.D. algorithm, when the priority list is formed by a heuristic ordering procedure, and ω_{OPT} be the length of the corresponding optimal schedule.

Although such a computation model was examined by Kafura and Shen in [Kaf],[KS1] and [KS2], they established worst-case bounds for the P.D. algorithm which correspond to pathological situations and hence, they are not as informative as might be desired. In this appendix, we establish generalised worst-case performance bounds for the same algorithm when the priority list is in a RAND, STF, LMF, LMST, LMLT or LTF ordering.

Theorem III.1: If the priority list is in a RAND or STF ordering, then

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq r + 2 + \frac{u}{2^r(\ell+1)} \leq 2 + \log_2(\frac{m}{\ell+1})$$

for $1 \leq \ell \leq (m-1)$, and

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 2 - \frac{1}{m} \quad \text{for } \ell=m.$$

This bound can be obtained by following a similar analysis as the one used in Theorem 6.2.1. Further, similar task systems with the ones used in Examples 6.2 and 6.3 can cause the algorithm to deviate from the optimal performance by the values given in Theorem III.1. Thus, the bounds are best possible ones. However, notice that for $\ell=1$, we obtain the corresponding worst-case bound presented in [Kaf] and [KS2].

Now, if the priority list is in a LMF or LMST ordering then, following a similar analysis the one given in Theorem 6.2.3 we can establish the following theorem.

Theroem III.2: Let the priority list be in a LMF or LMST ordering. Then,

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 2 - \frac{1}{\ell}, \quad 1 \leq \ell \leq m.$$

Moreover, this bound is a best possible one and can be realised by considering a task system similar to the ones presented in Examples 6.4 and 6.5. For $\ell=m$, $(\omega_{PD}/\omega_{OPT}) \leq 2-\frac{1}{m}$ which is the bound proved by Kafura and Shen for this case.

Furthermore, if the priority list is in LMF ordering and the jobs with $\ell_j=i$, $1 \leq i \leq m$, are arranged in LTF ordering, then the following theorem can be proved.

Theorem III.3: Let the priority list be in a LMLT ordering. Then,

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 1 + \frac{1}{k'+1} - \frac{1}{(k'+1)\ell} \quad , \quad \text{for } \ell=1,2 \quad \text{and}$$

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq \max\left\{\left(1 + \frac{1}{k'} - \frac{1}{(k'(\ell-1)}\right) \ , \ \left(1 + \frac{1}{k'+1} - \frac{1}{(k'+1)\ell}\right)\right\} \ ,$$

for $3 \leq \ell \leq m$.

The bound is a best possible one. A similar task system with the one presented in Example 6.6 can be used to show that the first factor of the bound is attained. On the other hand, the following example shows that the second factor of the bound can also be reached.

Example III.1: Consider the task system $(J,\{m_j\},\{t_j\})$, where the jobs [illegible] with at most $\ell_j=\ell$ are defined by:

J_1 : $(|P_1| \ , \ \frac{1}{(k'+1)\ell})$

J_j : $(|P_j| \ , \ \frac{j}{(k'+1)\ell})$, $j=2(1)(\ell-1)$

J_ℓ : $(|P_\ell| \ , \ \frac{(k'+1)\ell-1}{(k'+1)\ell})$

J_j : $(|P_\ell| \ , \ (\frac{1}{k'+1} + \frac{i}{(k'+1)\ell}))$, $j=(\ell+1)(1)(2\ell-2)$, $i=(\ell-2)(-1)1$

and J_j : $(|P_\ell| \ , \ \frac{1}{k'+1})$, $j=(2\ell-1)(1)((k'+1)(\ell-1)+2)$.

The priority list $L=(J_1,J_2,\ldots,J_{(k'+1)(\ell-1)+2})$ is in a LMLT ordering. The schedules resulting from the P.D. algorithm and the corresponding optimal

scheduling process are shown in Fig.III.1 and Fig.III.2 respectively. However, the ratio of the completion times of these schedules is:

$$\frac{\omega_{PD}}{\omega_{OPT}} = 1 + \frac{1}{k'+1} - \frac{1}{(k'+1)\ell} \quad ,$$

which is the value of the second factor of the bound given in Theorem III.3

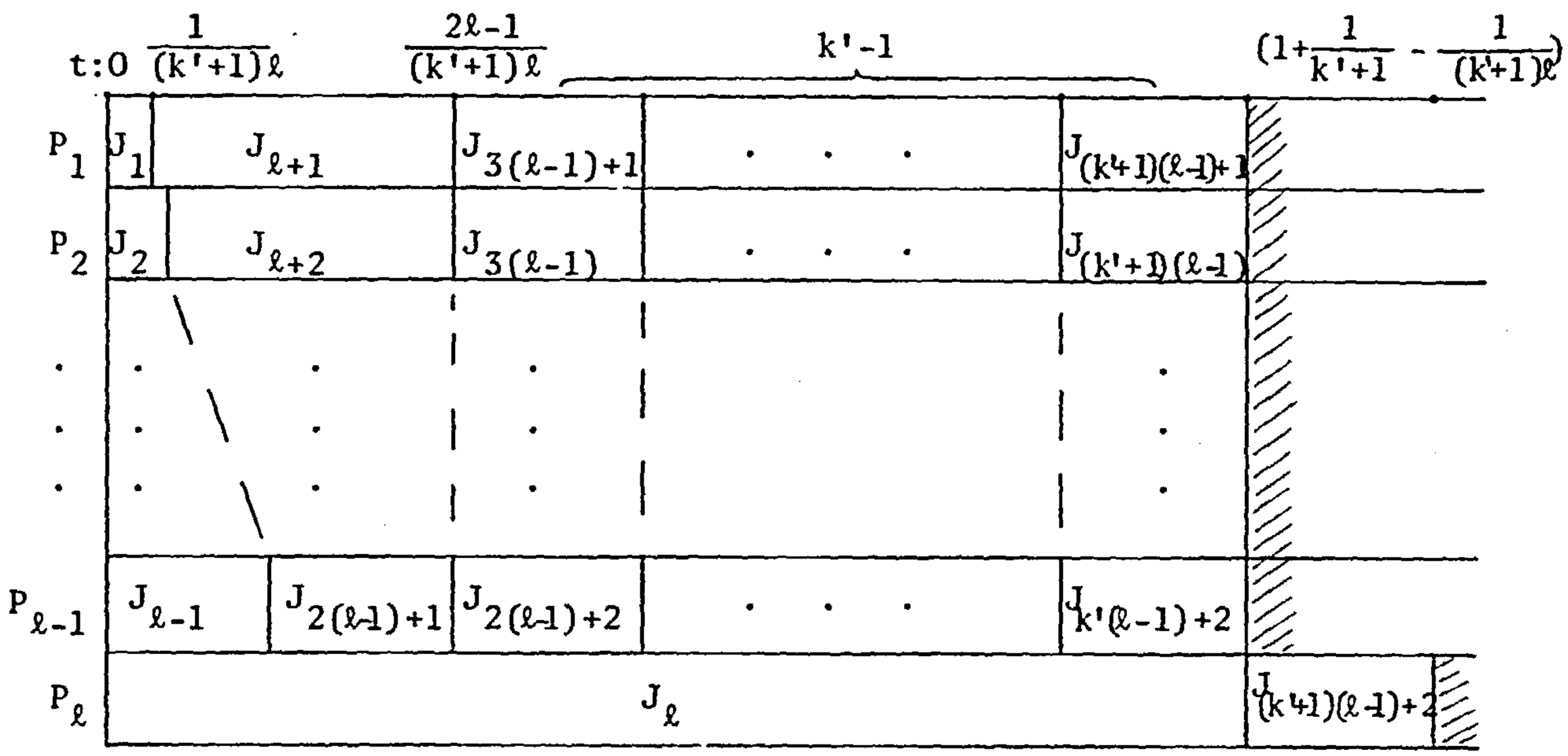


FIGURE III.1: The schedule constructed by the P.D. algorithm for the priority list given in Example III.1

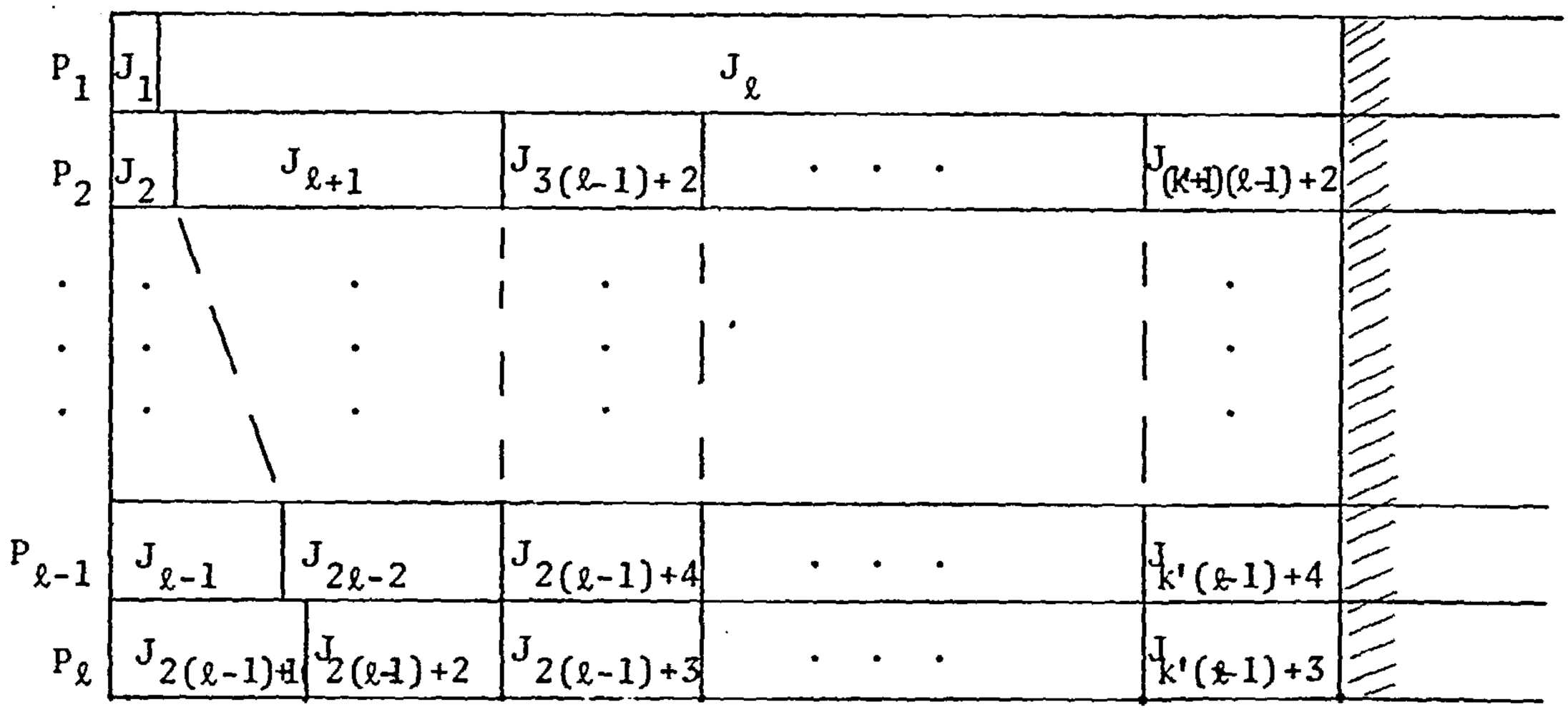


FIGURE III.2: The optimal schedule for the task system given in Example III.1

In the case, where $\ell=m\geq 3$ and $k'=1$ we obtain $(\omega_{PD}/\omega_{OPT})\leq 2-(1/(m-1))$ while for $k'=1$ and $\ell=2$ we get $(\omega_{PD}/\omega_{OPT})\leq 5/4$. These bounds have been proved as the worst-case bounds when the priority list is in a LMLT ordering in [Kaf] and [KS2].

Finally, if the priority list is in a LTF ordering then in order to obtain the lower limit of the worst-case performance for $1\leq\ell\leq m-1$ as well as prove the upper limit for $\ell=m$, a similar analysis to the one used in part (i) of Theorem 6.2.5 is followed.

Theorem III.4: Let the priority list be in a LTF ordering. Then, for $1\leq\ell\leq m-1$,

$$\max\left\{\frac{\omega_{PD}}{\omega_{OPT}}\right\} \geq 1 + H_m - H_\ell \simeq 1 + \ell n\left(\frac{m}{\ell}\right)$$

and for $\ell=m$,

$$\frac{\omega_{PD}}{\omega_{OPT}} = 1 \quad , \qquad k^*=1,2, \quad \text{and}$$

$$\frac{\omega_{PD}}{\omega_{OPT}} \leq 1 + \frac{1}{k^*} - \frac{1}{k^*m} \quad , \quad k^*\geq 3.$$

For $\ell=1$ we have $\max\left\{\frac{\omega_{PD}}{\omega_{OPT}}\right\}\geq H_m \simeq \ell n(m)$, which is the lower limit provided in [Kaf] and [KS2]. On the other hand, for $\ell=m$ the bound is exactly the same with the one derived in [CS2] for the same algorithm, when the classical homogeneous multiprocessor model is examined. However, our analysis is much simpler than the one used in [CS2] to prove the bound. In addition, this bound is a best possible one. This is shown in the following example, which is an extension of the examples given in [CS2].

Example III.2: Consider the task system $(J,\{m_j\},\{t_j\})$, where the jobs are defined by:

$$J_1 : (|P_m| \ , \ (4m-2))$$

$$J_j : (|P_{m-j+1}|, \ (4m-(j+1))), \qquad j=2(1)m$$

$$J_j : (|P_{i+1}| \ , \ (3m-i)), \qquad j=(m+1)(1)(2m-1), i=1(1)(m-1)$$

and $J_j : (|P_m|, 2m), \qquad j=2m(1)((k^*-1)m+1)$

where $k^* \geq 4$.

Clearly, the priority list $L=(J_1,J_2,\ldots,J_{(k^*-1)m+1})$ is in a LTF ordering. Fig.III.3 and Fig.III.4 show the schedules resulting from the P.D. algorithm and the optimal scheduling process respectively.

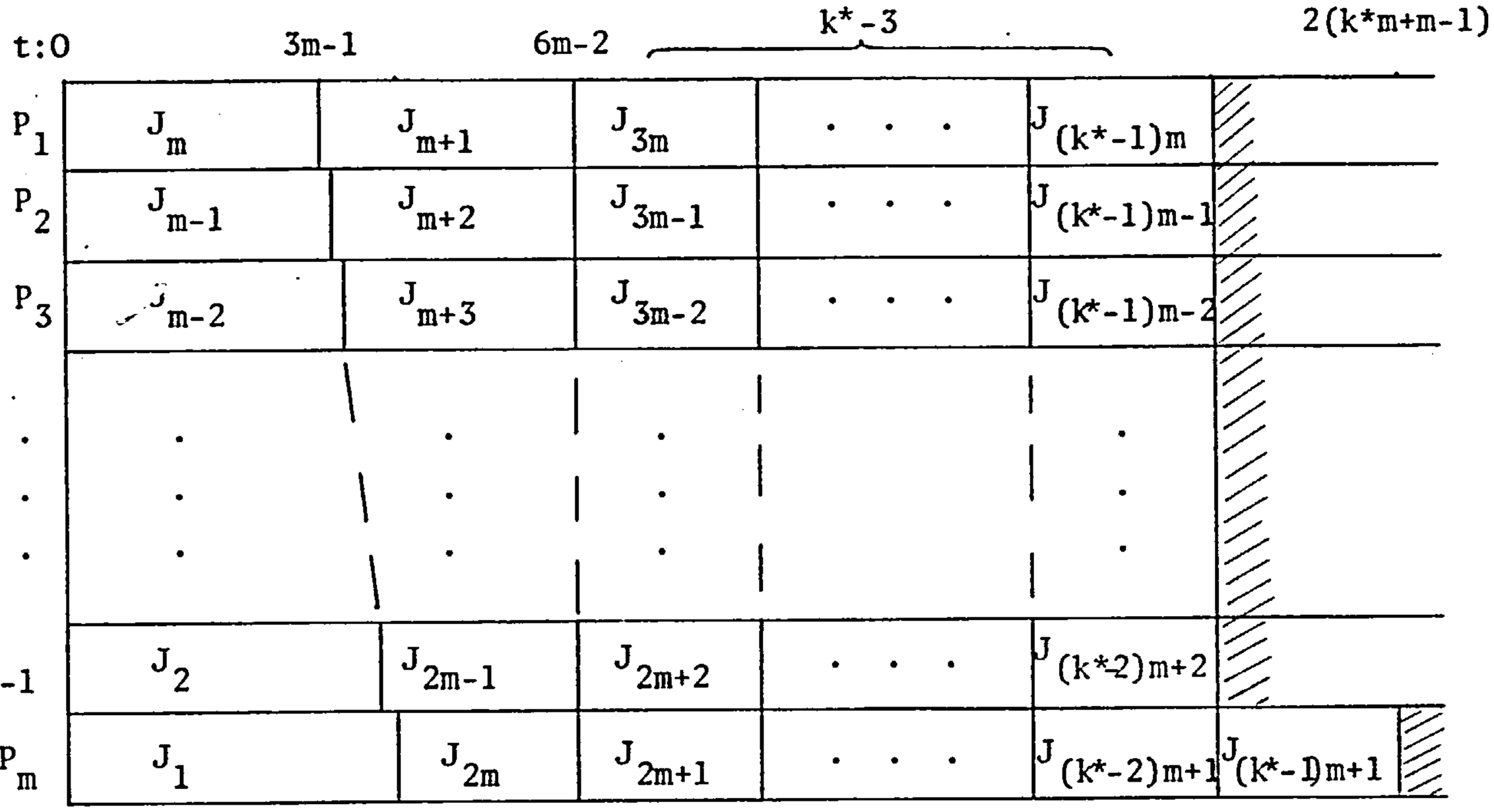


FIGURE III.3: Schedule resulting from the P.D. algorithm for the given priority list in Example III.2

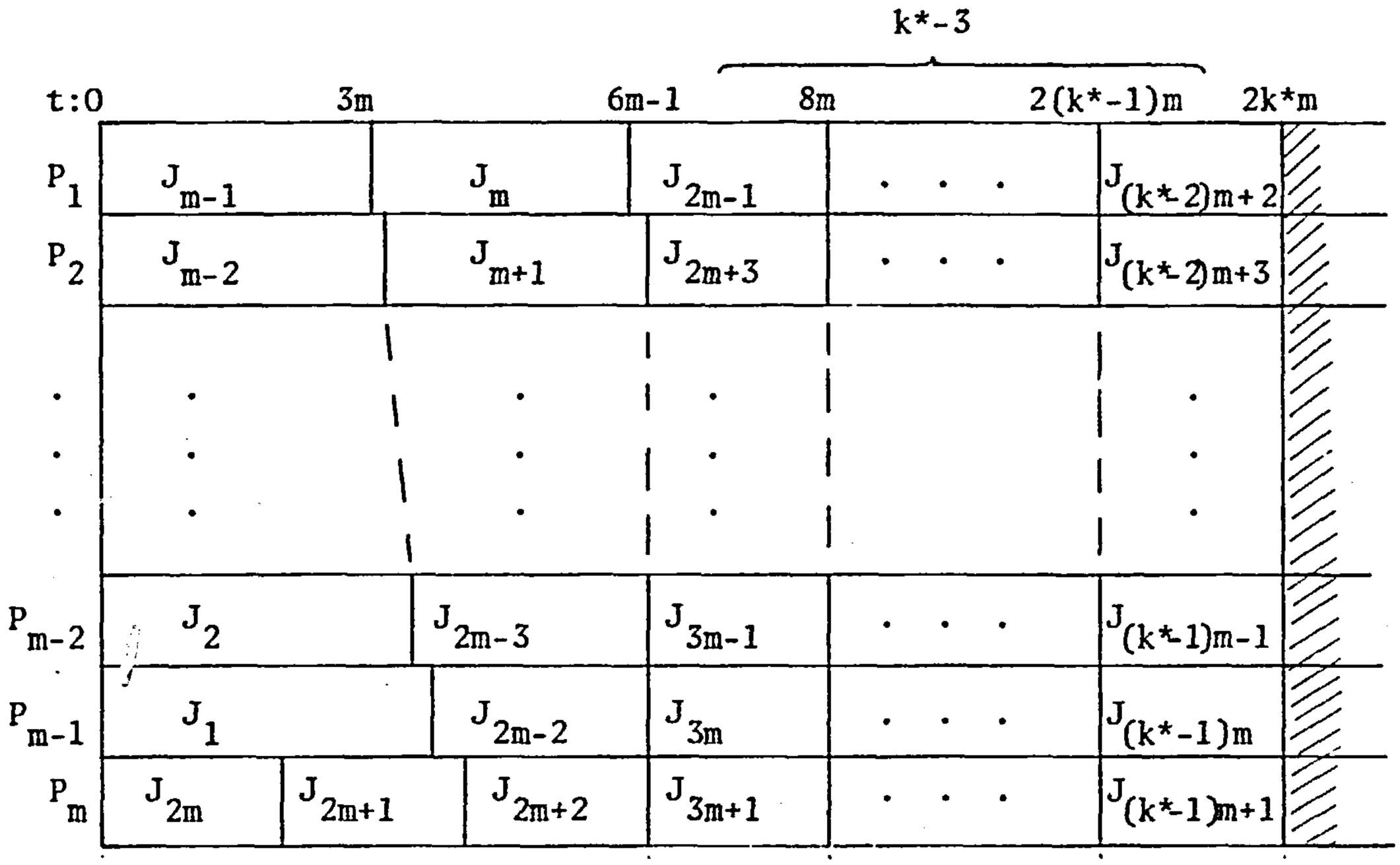


FIGURE III.4: Optimal schedule for the given task system in Example III.2

Thus, the ratio of the completion times of the above schedule is: $(\omega_{PD}/\omega_{OPT})=1+(1/k^*)-(1/(k^*m))$, which is the value of the worst-case bound of Theorem III.4 for $\ell=m$.

For $k^*=3$, a similar example with the one presented in [Gr2] can show that the bound is a best possible one for this value of k^* as well.

APPENDIX IV

HOMOGENEOUS MULTIPROCESSOR SYSTEM WITH INDEPENDENT MEMORIES - WORST-CASE BOUNDS OF Q.A.D. SCHEDULING ALGORITHMS - COMPLETION TIME PERFORMANCE CRITERION

Consider, as in the previous Appendices a task system $(J,\{m_j\},\{t_j\})$ with n independent jobs to be scheduled on a homogeneous multiprocessor system with private memories and a fixed number of processors. Let ω_{QAD} be the completion time of the schedule constructed by the Q.A.D. algorithm, when the priority list is formed by a heuristic ordering rule, and ω_{OPT} be the length of the corresponding optimal schedule.

Theorem IV.1: If the priority list is in an arbitrary ordering, then

$$\frac{\omega_{QAD}}{\omega_{OPT}} \leq \nu_x ,$$

where ν_x and x are as defined in Theorems 5.3.1 and 6.3.1, respectively.

A similar argument with the one given in Theorem 6.3.1 can prove this theorem.

Moreover, the STF, LMF, LMST and LMLT ordering rules can not provide any improvement over the worst-case performance of an arbitrary ordering. Actually, the task system in Example 6.8, when $b_i=b_{i+1}$, $1\leq i\leq m-1$, can be used to support the above statement and also to show that the bound is a best possible one.

Theorem IV.2: Let the priority list be in a LTF ordering. Then,

$$\max\left\{\frac{\omega_{QAD}}{\omega_{OPT}}\right\} \geq 1+H_m-H_{\ell^*} \simeq 1+\ell n\left(\frac{m}{\ell^*}\right), \text{ for } 1\leq \ell^*\leq m-1, \text{ and}$$

$$\frac{\omega_{QAD}}{\omega_{OPT}} \leq 1 + \frac{m-1}{\bar{k}+1} \quad , \text{ if } \bar{k}\geq m-1 \text{ and}$$

$$\frac{\omega_{QAD}}{\omega_{OPT}} \leq 1 + \frac{\bar{k}}{\bar{k}+1} \quad \text{otherwise, for } \ell^*=m.$$

A similar analysis with the one used in Theorem 6.3.3 can prove the above theorem. Moreover, when $\ell^*=m$ both bounds are best possible ones. This is realised from the following example.

Example IV.1: Consider the task system $(J,\{m_j\},\{t_j\})$, where the jobs are defined by:

$$J_1 \quad : \ (|P_m|, \frac{m}{\bar{k}+1})$$

and $$J_{1+(j-1)m+i}: \ (|P_i|, \frac{1}{\bar{k}+1}) \ , \qquad i=1(1)m, \ j=1(1)\bar{k}.$$

Clearly the priority list is $L=(J_1,J_2,\ldots,J_{\bar{k}m+1})$ is in a LTF ordering. The schedule resulting from the Q.A.D. algorithm is shown in Fig.IV.1. Moreover, an optimal schedule of length 1 it can be easily formed.

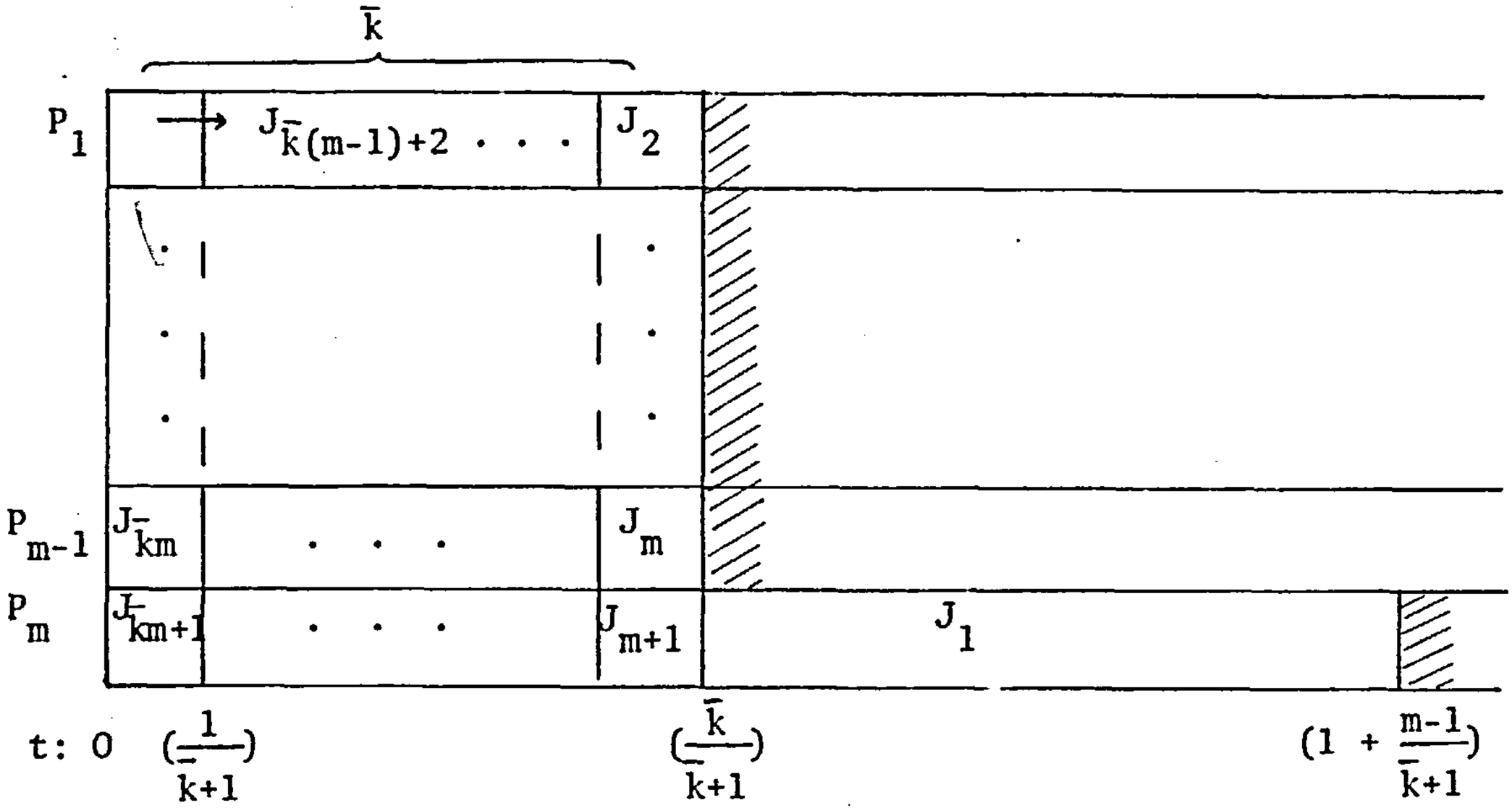


FIGURE IV.1: Schedule constructed by the Q.A.D. algorithm for the priority list given in Example IV.1

Therefore, the ratio of these two schedules is

$$\frac{\omega_{QAD}}{\omega_{OPT}} = 1 + \frac{m-1}{\bar{k}+1} \ ,$$

which is the worst value of the corresponding bound given in Theorem IV.2 for $\ell^*=m$ and $\bar{k}\geq m-1$.

When $\bar{k}<m-1$, we can show that the worst value of the corresponding bound is also attained. Actually, we need only to consider the previous example with the time requirement of job J_1 to be 1 instead of $m/(\bar{k}+1)$, in order to show that the bound is a best possible one.

APPENDIX V

HETEROGENEOUS MULTIPROCESSOR SYSTEM WITH INDEPENDENT MEMORIES - A LOWER OPTIMAL BOUND FOR THE FINAL COMPLETION TIME

Consider a task system $(J,[t_{ij}])$ with n independent jobs to be scheduled on the heterogeneous multiprocessor system with independent memories, as described in section 7.2. Let $\{A_{rs}\}= \bigcup_{\ell=1}^{r} F_{\ell s}$, $1\leq r\leq m$, $1\leq s\leq nc$, where $F_{\ell s}$ is the set of jobs with $\ell_j=\ell$, $1\leq \ell\leq m$, $1\leq j\leq n$, which belong to the s^{th} class of jobs and nc is the number of classes in the task system. Also, let T_{rs} be the sum of the time requirements of all the jobs belonging to the set A_{rs}, when they are run on a standard processor. Finally, let $b_{\ell s}$ be the processing speed of the ℓ^{th} processor for the s^{th} class.

We define ω and t_{max} as:

$$\omega = \max_{1\leq r\leq m} \left\{ \sum_{s=1}^{nc} \left[\frac{T_{rs}}{r(\max_{1\leq i\leq r} \{b_{is}\})} \right] \right\} \tag{V.1}$$

and

$$t_{max} = \max_{1\leq j\leq n} \{\tau_j\} .$$

Now, we say that

$$\omega_{MIN} = \max\{t_{max},\omega\} \tag{V.2}$$

is the minimum completion time of the jobs in the task system $(J,[t_{ij}])$. The first term guarantees that the minimal length is at least as long as the maximum of the minimum time requirement of all the jobs in the task system. This is necessary since no parts of the same job can be executed simultaneously. The second term in the definition of ω_{MIN} insures that there are sufficiently large memories available for enough time to satisfy the memory requirements of the jobs. The following theorem establishes that ω_{MIN} is a lower optimal bound for the final finishing time of any task system.

Theorem: For the model under investigation,

$$\omega_{OPT} \geq \omega_{MIN} .$$

Proof: Let us assume that

$$\omega_{OPT} < \omega_{MIN} . \tag{V.3}$$

Then, because of the equation (V.2) we have

$$\omega_{OPT} < \max\{t_{max}, \omega\}$$

and since, $\omega_{OPT} \geq t_{max}$ we have

$$\omega_{OPT} < \omega,$$

or because of the equation (V.1)

$$\omega_{OPT} < \max_{1 \leq r \leq m} \left\{ \sum_{s=1}^{nc} \left\lceil \frac{T_{rs}}{r(\max_{1 \leq i \leq r} \{b_{is}\})} \right\rceil \right\} . \qquad (V.4)$$

Let k be the smallest integer such that

$$\omega = \sum_{s=1}^{nc} \left\lceil \frac{T_{ks}}{k(\max_{1 \leq i \leq k} \{b_{is}\})} \right\rceil .$$

Thus, the inequality (V.4) becomes

$$\omega_{OPT} < \sum_{s=1}^{nc} \left\lceil \frac{T_{ks}}{k(\max_{1 \leq i \leq k} \{b_{is}\})} \right\rceil . \qquad (V.5)$$

On the other hand, if ω_{OPT}^{s}, $1 \leq s \leq nc$, is the theoretical optimal completion time of the jobs in the set A_{ks}, we have

$$\left.\begin{aligned} T_{k1} &\leq k\omega_{OPT}^{1}(\max_{1 \leq i \leq k} \{b_{i1}\}) \\ T_{k2} &\leq k\omega_{OPT}^{2}(\max_{1 \leq i \leq k} \{b_{i2}\}) \\ &\vdots \\ \text{and} \quad T_{k,nc} &\leq k\omega_{OPT}^{nc}(\max_{1 \leq i \leq k} \{b_{i,nc}\}) . \end{aligned}\right\} \qquad (V.6)$$

Furthermore, the summation of the inequalities (V.6) will give us an inequality of the same direction, which can be expressed as

$$\sum_{s=1}^{nc} \left\lceil \frac{T_{ks}}{k(\max_{1 \leq i \leq k} \{b_{is}\})} \right\rceil \leq \omega_{OPT}$$

since, $\omega_{OPT} \geq \omega_{OPT}^{1} + \ldots + \omega_{OPT}^{nc}$. This inequality contradicts the inequality (V.5) and hence (V.1) as well. Thus, $\omega_{OPT} \geq \omega_{MIN}$ and consequently, ω_{MIN} is a lower optimal bound for the final finishing time of a task system.