COMPUTER AND JOB-SHOP SCHEDULING THEORY

EDITED BY
E. G. COFFMAN, JR.
PENNSYLVANIA STATE UNIVERSITY

COAUTHORS:
J. L. BRUNO, E. G. COFFMAN, JR.,
R. L. GRAHAM, W. H. KOHLER, R. SETHI,
K. STEIGLITZ, AND J. D. ULLMAN

A WILEY-INTERSCIENCE PUBLICATION
JOHN WILEY & SONS
NEW YORK / LONDON / SYDNEY / TORONTO

Library of Congress Cataloging in Publication Data
Main entry under title:

Computer and job-shop scheduling theory.

"A Wiley–Interscience publication."
Bibliography: p.
Includes index.
1. Scheduling (Management)—Mathematical models.
2. Scheduling (Management)—Data processing.
I. Coffman, Edward Grady.

TS157.5.C65 658.5'1 75-19255
ISBN 0-471-16319-8

Printed in the United States of America

10 9 8 7 6 5 4 3 2

COMPUTER AND JOB-SHOP SCHEDULING THEORY

PREFACE

In the past several years interest and new results in the theory of deterministic scheduling have mounted at an increasing rate. This book is an attempt to represent the current position of the field in terms of research and the types of problems currently being investigated. As a result, most of the material covered is relatively recent and cannot be found elsewhere in textual form. Although the book consists of six coordinated contributions, it is not to be considered an edited collection of papers, but rather a multiple-author text written specifically for its purpose. Simply because of the number of coauthors, it was necessary to have an editor to coordinate the overall effort of seven people.

The book provides a theoretical treatment of sequencing problems arising in computer and job-shop environments. However, the models are simple in structure and are consequently meaningful in a very large variety of applications. Briefly, the general model studied assumes a set of tasks or jobs and a set of resources to be used in executing or servicing the tasks. In almost all cases the models are deterministic in the sense that the information describing tasks is assumed to be known in advance. This information includes task execution times, operational precedence constraints, deferral costs, and resource requirements. The principal sequencing problems examined are the minimization of schedule lengths, minimization of the mean time in system (weighted by deferral costs), and scheduling to meet due dates or deadlines. A number of closely related problems are also studied. The results presented include efficient optimal algorithms, heuristics and their performance bounds, efficient enumerative and iterative methods, and mathematical descriptions of the complexity of a wide variety of sequencing problems.

Computers arise in the subject matter in at least three ways. First, they represent an almost universal job shop for our purposes. The appearance of virtually all the problems we analyze can be observed or envisioned in the design or operation of general-purpose computer systems, although the prime importance of specific problems may exist in other applications. Second, computers must be considered in the implementation of the enumerative and iterative approaches to sequencing problems. Finally, the field of computer science is the origin of the complexity theory that we introduce and then apply to problems of sequence.

As in other areas of applied mathematics, scheduling theory spans many academic disciplines; in our case these include Industrial Engineer-

ing, Management Science, Business Administration, Operations Research, Computer Science, and Electrical Engineering. The book is directed primarily to these departments of research institutes and of the academic world in which graduate students and faculty are performing research in deterministic scheduling theory. Because of the nature of the material, substantial mathematical maturity must be assumed on the part of the reader. Thus as a text or supplementary material one must assume a course with graduate students towards the end, at least, of the first year of study.

I acknowledge the *Institut de Recherche d'Informatique et d'Automatique* (Rocquencourt, France) for the primary support in the production of this book. The initial phases and much of the work reported in Chapters 2 and 3 were supported in part by the National Science Foundation under grant NSF-28290 at the Pennsylvania State University. The work in Chapter 6 was supported by the National Science Foundation under grants NSF-GK-37400 and NSF-GK-42048, and the U.S. Army Research Office under contract DAHCO4-69-C-0012. Thanks are due to Mrs. Teddi Potter and Mrs. Hannah Kresse for typing portions of the manuscript.

Finally, special thanks are due to Drs. M. R. Garey and D. S. Johnson for their valuable efforts in examining and correcting the manuscript. They have been of particular help to me, as editor, except insofar as they have, through a constant stream of new results, attempted to antiquate the book before its appearance.

E. G. COFFMAN, JR.

University Park, Pennsylvania
May 1975

CONTENTS

CHAPTER ONE

INTRODUCTION TO DETERMINISTIC SCHEDULING THEORY

E. G. COFFMAN, JR.
INSTITUT DE RECHERCHE D'INFORMATIQUE ET D'AUTOMATIQUE, ROCQUENCOURT, FRANCE*

In this chapter we introduce and put into a common context the contents of the remainder of the book. In so doing, we present common notation for models analyzed in subsequent chapters, as well as some additional results serving as background or complementary material extending and unifying the coverage of the book. As a result, it is recommended that the reader examine the present chapter before attempting to read Chapters 2 through 6. Although each of Chapters 2 through 6 may be read without any essential recourse to material in the others, the reader will note that there are reasons for the sequence chosen for the chapters.

The mutual independence of Chapters 2 through 6 is of course a convenience and to some extent a result of a multiple-author book in which appear several different styles of writing and approaches to notation. However, the subject matter has been partitioned in such a way that this mutual independence is not unnatural. Consequently, any discontinuities in style and presentation experienced in passing from one chapter to another should be tolerable, especially if one first makes reference to relevant material in this chapter.

In Section 1.1 the objectives and motivations of the book are discussed; the general model to be studied is presented in Section 1.2. Section 1.3 discusses background results drawn primarily from Conway, Maxwell, and Miller [CMM]. Section 1.4 reviews the results of Chapters 2 through 6, making use of tabular presentations where possible. A number of related topics, covered in Section 1.5, complement the material in Chapters 2 through 6. Finally, in Section 1.6 some comments are made on the notation used in subsequent chapters.

* On leave from The Pennsylvania State University.

1.1 OBJECTIVES AND MOTIVATIONS

In very general terms, the scheduling problems studied in this book assume a set of resources or servers and a fixed system of tasks which is to be serviced by these resources. Based on prespecified properties of and constraints on the tasks and resources, the problem is to find an efficient algorithm for sequencing the tasks to optimize or tend to optimize some desired performance measure. The primary measures studied are schedule length and the mean time spent in the system by the tasks. The models of these problems we analyze are deterministic in the sense that all information governing the scheduling decision is assumed to be known in advance. In particular, the tasks and all information describing them are assumed to be available at the outset, which we normally take as time $t = 0$.

Since one can not organize an effective work day, plan examination periods at universities, or even prepare a nontrivial meal without frequently encountering such problems, it does not seem necessary to dwell on motivations for the study of these problems. On the other hand, our interest must focus on problem formulations that reflect applications in which poor sequencing decisions incur intolerably large costs. Thus the assumptions by which the problem is formalized ought, for example, to reflect general industrial job shops. In fact, because of primary associations and interests of the authors, the original context of the models analyzed is often computer systems.

Readers who are well-informed about general-purpose computer architectures and the problems of economical computer operation will realize at once that we are sacrificing very little in generality. One seldom finds basic sequencing problems that do not have interesting and important counterparts in existing or proposed computer systems. But it is worth noting here, also as a justification for the book's title, that specific questions of application will not be discussed. For, as we shall see later, the book reflects the fact that significant theoretical results (apart from general complexity issues) are largely confined to rather simplistic models that apply over a broad spectrum of real-life scheduling problems.

Many similar terms are used abstractly in this book without being formally defined. For example, unless otherwise noted, jobs, tasks, programs (in a computer), and customers can all be regarded as equivalent for our purposes. Also, resources may be referred to as machines, storage devices, and most commonly, processors. With respect to appropriate resources jobs may be performed, run, executed, stored, or serviced. We may use rule, procedure, or an appropriate function or mapping instead of algorithm; and the terms scheduling, sequencing, allocation, and assign-

ment are used synonymously or analogously, depending on context. As customary in current literature, nontrivial algorithms are specified using an informal Algol-like notation; no special devices appear that will not be transparent to the reader.

Broadly speaking, the goal of this book is a presentation and analysis of deterministic sequencing problems which is sufficiently comprehensive to ensure that

(1) the status of recent and ongoing research in this field is adequately covered, and

(2) the significant principles that can be abstracted from current and past analyses and formal approaches to these problems are well represented.

In these days, any attempt at a book that is an unqualified success in both respects is bound to fail; it would be in a constant state of writing. Of course, the primary reason for this is the usual difficulty in remaining *au courant* in a field that has accumulated as much momentum as scheduling theory. But the problem is aggravated enormously because scheduling theory spans several disciplines, each of which is large and vigorous in its own right. We are referring to the many industrial research as well as academic departments of Operations Research, Management Science, Computer Science, Industrial Engineering, Electrical Engineering, and Applied Mathematics, in which one finds the (increasingly) many researchers currently engaged in the advancement of scheduling theory. It would be impossible to cover the great variety of specific systems that form the principal motivations in these disciplines. Thus the more modest goal is to focus on tractable, generic models of simple structure whose combinatorial complexity and analysis resembles or specializes that of the various structures encountered in the above-mentioned disciplines.

After a more formal discussion of the results of subsequent chapters we return to the question of objectives vis-à-vis the types of problems and approaches not treated in detail. We conclude this section with some remarks on the literature as it bears on our point of departure. The emphasis of subsequent chapters is on results which have appeared within the last 7 or 8 years; in fact, most of these have appeared within the last 4 or 5 years. Thus in terms of textual material, the initial chapters of the book by Conway, Maxwell, and Miller [CMM], which was published in 1967, form a natural basis for the present work. We especially recommend this book to the reader desiring engineering motivation beyond that provided here. For again, we emphasize that our treatment is almost wholly mathematical, with very little recourse to discussions of pragmatics. The applicability (and, of course, inapplicability) of our work will be quite evident in virtually all cases, owing primarily to the simplicity of the models. But in [CMM] one obtains good insight into the

general problems in practice and the many features of such problems that extend the models analyzed here but for which comparable results are not known (see also [Bak]).

This book presents virtually all the theoretical results in [CMM] that fall within the boundaries of deterministic scheduling theory. They form the background for the new results. Indeed, this background material occupies only a couple of brief sections in Chapters 1 and 2. Additional background material concerned with computer sequencing problems and relevant to parts of Chapters 2 and 5 can be found in a more recent text [CD] on operating-systems theory. In fact, the third chapter of the latter book forms a small subset whose limitations motivated the conception of the present work. Thus the present book is largely new material not appearing elsewhere in textual form. For recent survey papers dealing with many of the subjects of this book, the reader is referred to [G3], [Co], [B1], and [BLR].

1.2 A GENERAL MODEL

The scheduling model, from which subsequent problems are drawn, is described by considering in sequence the resources, task systems, sequencing constraints, and performance measures. (At the end of the chapter we describe briefly the notational questions relevant to this and subsequent chapters).

1.2.1 Resources

In the majority of the models studied, the resources consist simply of a set $P = \{P_1, \ldots, P_m\}$ of processors. Depending on the specific problem, they are either identical, identical in functional capability but different in speed, or different in both function and speed.

In the most general model there is also a set of additional resource types $\mathcal{R} = \{R_1, \ldots, R_s\}$, some (possibly empty) subset of which is required during the entire execution of a task on some processor. The total amount of resource of type R_j is given by the positive number m_j. In the computer application, for example, such resources may represent primary or secondary storage, input/output devices, or subroutine libraries.* Although it is possible to include the processors in $\mathcal{R}$, it is more convenient to treat them separately because

* In this chapter, unless noted otherwise, and in Chapter 4 a resource type R_i is to be regarded as a set of m_i identical resources. In the somewhat more general model of Chapter 5, m_i is the *amount* of resource R_i usually normalized to 1, any fraction of which may be required for executing a given task.

(1) they will constitute a resource type necessarily in common with all tasks (although two different tasks need not require the same processors), and
(2) they are discretized with the restriction that a task can execute on at most one processor at a time.
These constraints need not apply to the resource types R_i, $1 \le i \le s$.

1.2.2 Task Systems

A general task system for a given set of resources can be defined as the system $(\mathcal{T}, \prec, [\tau_{ij}], \{\mathcal{R}_j\}, \{w_j\})$ as follows:

1. $\mathcal{T} = \{T_1, \ldots, T_n\}$ is a set of tasks to be executed.
2. $\prec$ is an (irreflexive) partial order defined on $\mathcal{T}$ which specifies operational precedence constraints. That is, $T_i \prec T_j$ signifies that T_i must be completed before T_j can begin.
3. $[\tau_{ij}]$ is an $m \times n$ matrix of execution times, where $\tau_{ij} > 0$ is the time required to execute T_j, $1 \le j \le n$, on processor P_i, $1 \le i \le m$. We suppose that $\tau_{ij} = \infty$ signifies that T_j cannot be executed on P_i and that for each j there exists at least one i such that $\tau_{ij} < \infty$. When all processors are identical we let τ_j denote the execution time of T_j common to each processor.
4. $\mathcal{R}_j = [R_1(T_j), \ldots, R_s(T_j)]$, $1 \le j \le n$, specifies in the ith component, the amount of resource type R_i required throughout the execution of T_j. We always assume $R_i(T_j) \le m_i$ for all i and j.
5. The weights w_i, $1 \le i \le n$, are interpreted as deferral costs (or more exactly cost rates), which in general may be arbitrary functions of schedule properties influencing T_i. However, the w_i are taken as constants in the models we analyze. Thus the "cost" of finishing T_i at time t is simply $w_i t$.

This formulation contains far more generality than we intend to embrace in subsequent chapters, but each problem studied can be represented as a special case of the model. One particular restriction worth noting is the limitation on operational precedence. We cannot, for example, represent loops in computer programs modeled as task systems. Note that the partial order $\prec$ is conveniently represented as a directed, acyclic graph (or *dag*) with no (redundant) transitive arcs. Unless stated otherwise, we assume $\prec$ is given as a list of arcs in such a graph. In general, however, the way in which a partial order is specified in a given problem may influence the complexity of its solution. (We return to this point later.)

In Fig. 1.1 for example, the notation T_i/τ_i is introduced for labeling

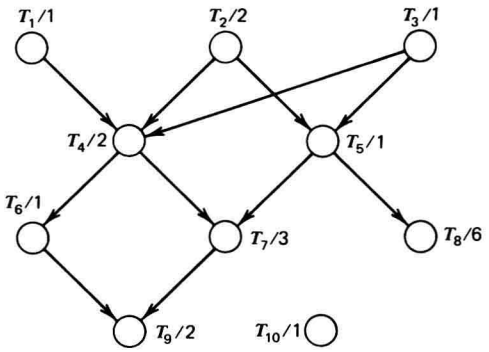

Figure 1.1 A dag representation of $(\mathcal{T}, <, \{\tau_i\})$.

Notation and properties:
1. Acyclic.
2. No transitive edges: (T_1, T_6) would be such an edge.
3. T_1, T_2, T_3, T_{10} are initial vertices; T_8, T_9, T_{10} are terminal vertices.
4. For example, T_7 is a successor of T_1, T_2, T_3, T_4, T_5 but an immediate successor of only T_4, T_5; T_5 is a predecessor of T_7, T_8, T_9 but an immediate predecessor of only T_7, T_8.
5. Levels:

8	9	8	7	7	3	5	6	2	1
T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}

6. Critical paths: T_2, T_5, T_8 and T_2, T_4, T_7, T_9.

vertices. Tasks are occasionally referred to simply by their indices (e.g., "task 1" may be used rather than T_1, especially when graph vertices are more conveniently labeled with integers).

As the reader might expect, we make heavy use of graphical methods for defining task systems, rather than defining them as appropriate five-tuples. In so doing we choose a number of more or less commonly used terms concerning dags. In particular, a *path* of length k from T to T' in a given graph G is a sequence of vertices* (tasks) $T_{i_1}, \ldots, T_{i_k}$ such that $T = T_{i_1}$, $T' = T_{i_k}$ $(k \geq 1)$ and $(T_{i_j}, T_{i_{j+1}})$ is an arc in G for all $1 \leq j \leq k-1$. Moreover, if such a path exists, T will be called a *predecessor* of T' and T' a *successor* of T. 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 predecessors, or in the second case, exactly

*The term *node* is used synonymously with vertex.

one vertex with no successors, it is also called a *tree.* In either case, the terms *root* and *leaf* have the usual meaning. The *level* of a vertex T is the sum of the execution times associated with the vertices in a path from T to a terminal vertex such that this sum is maximal. Such a path is called a *critical path* if the vertex T is at the highest level in the graph.

1.2.3 Sequencing Constraints

By "constraint" we mean here a restriction of scheduling algorithms to specific (though broad) classes. Two main restrictions are considered.

1. *Nonpreemptive* scheduling: with this restriction a task cannot be interrupted once it has begun execution; that is, it must be allowed to run to completion. In general, *preemptive* scheduling permits a task to be interrupted and removed from the processor under the assumption that it will eventually receive all its required execution time, and there is no loss of execution time due to preemptions (i.e., preempted tasks resume execution from the point at which they were last preempted).

2. *List* scheduling: in this type of scheduling an ordered list of the tasks in $\mathcal{T}$ is assumed or constructed beforehand. This list is often called the *priority list.* The sequence by which tasks are assigned to processors is then decided by a repeated scan of the list. Specifically, when a processor becomes free for assignment, the list is scanned until the first unexecuted task T is found which is ready to be executed; that is, the task can be executed on the given processor, all predecessors of T have been completed, and sufficient resources exist to satisfy $R_i(T)$ for each $1 \leq i \leq s$. This task is then assigned to execute on the available processor. We assume the scan takes place instantaneously. Also, if more than one processor is ready for assignment at the same time, we assume they are assigned available tasks in the order P_1 before P_2 before P_3, etc. As a matter of notation we assume that the list is ordered (and scanned) from left to right and written in the form $L = (T_{i_1}, \ldots, T_{i_n})$. Preemptions are not considered; thus list schedules form a subset of nonpreemptive schedules.

Before discussing performance measures, let us illustrate the means by which schedules are to be represented graphically, assuming $s = 0$. We use the type of timing diagram illustrated in Fig. 1.2 for the task system shown in Fig. 1.1. In the obvious way the number of processors determines the number of horizontal lines which denote time axes. The hatching shown in the figure represents periods during which processors are idle. When the need arises to refer to idle periods, the symbol $\emptyset$, appropriately subscripted when necessary, is used. Also, the symbol D denotes these diagrams, when such notation is desired. The symbols s_i