Journal of the Operations Research
Society of Japan
Vol. 31, No. 2, June 1988

DETERMINISTIC SCHEDULING IN COMPUTER SYSTEMS: A SURVEY

Tsuyoshi Kawaguchi Seiki Kyan
University of the Ryukyu

(Received May 21, 1987; Revised September 26, 1987)

Abstract The progress of computer architecture has increased the necessity to design efficient scheduling algorithms according to the types of computer systems. We survey recent results of optimization and approximation algorithms for deterministic models of computer scheduling. We deal with identical, uniform and unrelated parallel processor systems, and flowshop systems. Optimality criteria considered in this paper are schedule-length and mean (weighted) flow-time. These are important measures for evaluating schedules in computer systems. Results on practical algorithms such as list scheduling are emphasized.

1. Introduction

The progress of computer architecture such as multiprocessor systems has increased the necessity to design efficient scheduling algorithms according to the types of computer systems. Two basic approaches have been considered for the evaluation of scheduling algorithms in computer systems. One is deterministic approach and the other is stochastic approach. An advantage of deterministic approach over stochastic one is that job parameter values are not constrained to fit a prescribed distribution [22].

This paper surveys recent results of optimization and approximation algorithms for deterministic models of computer scheduling, refered to as deterministic scheduling problems. We will deal with parallel processor problems and flowshop problems. Optimality criteria considered in this paper are schedule-length and mean (weighted) flow-time. We will also emphasize results of practical algorithms such as list scheduling and highest-level-first strategy, rather than elaborated algorithms. A parallel processor system corresponds to a multiprocessor computer system which can execute several jobs simultaneously in parallel. A flowshop system can be interpreted as a computer system where all jobs pass through several phases such as input, execution and output [12]. Moreover schedule-length is a processor utilization factor, and mean (weighted) flow-time corresponds to the mean (weighted) response time which is important from the user's viewpoint [6].

In this paper we will treat single processor problems as special cases of parallel processor problems, but we will not deal with openshop and jobshop problems which are often used as models of production scheduling rather than computer scheduling. The reader is refered to [7, 33, 45, 52, 53] for openshop problems, jobshop problems and parallel processor problems involving criteria other than schedule-length and mean (weighted) flow-time. Especially, we recommend [33] (or [52]) as a comprehensive survey. Moreover [7] gives detailed descriptions about problems with additoinal resources, and [38] summarizes recent results of approximation algorithms.

Section 2 introduces the basic concepts and presents a detailed problem classification. Results on parallel processor scheduling are described in section 3 for problems with no additional resources, and in section 4 for problems with additional resources. Section 5 deals with flowshop scheduling. Finally in section 6, we give concluding remarks.

2. Preliminaries

2.1 Basic concepts

First we briefly explain the basic concepts concerning the theory of the computational complexity. For formal definitions of these concepts, the reader is refered to [26].

Let $T_A(n)$ denote the largest amount of time needed by an algorithm A to solve a problem instance of size n. If there exists a constant c such that $T_A(n) \leqq c\, f(n)$ for all values of $n \geqq 0$, we say that *the time complexity* of the algorithm is $O(f(n))$. *A polynomial time algorithm* is one whose time complexity is $O(f(n))$, where f is some polynomial function.

There is a large class of problems, called *NP-complete problems*, which include classical hard problems such as the traveling salesman problem or the integer programming problem, etc. The best known algorithms for all NP-complete problems have exponential time complexity, and any NP-complete problem has a polynomial time algorithm if and only if all the others also have polynomial time algorithms. Many researchers conjecture that NP-complete problem has no polynomial time algorithm. As a consequence, the knowledge that a problem is NP-complete shows inherent intractability of the problem. NP-completeness is defined for decision problems which return "yes" or "no" answer.

Further, the notion of *NP-hardness* is used for showing that a problem is at least as hard as NP-complete problems. An optimization problem P is NP-hard if the decision problem corresponding to P is NP-complete.

Certain algorithms for NP-hard problems have the following property: they are polynomial if any upper bounds are imposed on input numbers, for example, which represent processing times of jobs; and otherwise they are not polynomial. Such algorithms are called *pseudopolynomial time algorithms*. Given a problem P, let $D(P)$ be the set of all instances of P. Moreover let P' denote subproblem which is created by restricting $D(P)$ to the instances, each of which has the maximum number bounded by a polynomial in its length. If there exists a polynomial for which P' is NP-hard, the original problem P is said to be *strongly NP-hard* or *NP-hard in the strong sense*.

Next we define a measure for evaluating goodness of approximation algorithms. Let P denote a minimization problem and let I be any instance of P. Then *the absolute performance ratio* for an algorithm A is given by

$$R_A = \inf \{ r \geqq 1 : R_A(I) \leqq r \text{ for all instances of } P \}$$

where

$$R_A(I) = A(I)/OPT(I) ,$$

where $A(I)$ is the cost obtained by algorithm A and $OPT(I)$ is the optimum cost.

Moreover *the asymptotic performance ratio* for algorithm A is given by

$$R_A^{\infty} = \inf \{ r \geqq 1 : \text{for some } K > 0,\ R_A(I) \leqq r \text{ for any instance } I \text{ with } OPT(I) \geqq K \}.$$

If $R_A \leqq r$ for some $r \geqq 1$, then $R_A^{\infty} \leqq r$. But the converse is not necessarily true. Unless stated otherwise, the term "*performance ratio*" means the absolute performance ratio in this paper.

2.2 Problem classification

The scheduling model that we consider has a set of jobs $\{J_1, \ldots, J_n\}$ and a processor system $\{P_1, \ldots, P_m\}$. Each job is to be executed on no more than one processor at a time, and each processor can process no more than one job at a time. Problems involving additional resources are discussed only in section 4, and so parameters about additional resources are given in section 4. Below we will describe various job and processor characteristics.

2.2.1 Processor environment

As stated in the preceding section, this paper deals with two types of processor systems: parallel processor systems and flowshop systems. In *a parallel processor system*, each job requires the use of exactly one processor, and each job can be performed on each processor. On the other hand, *a flowshop system* is a processor system in which each job J_i has a chain of m tasks T_{ij} and each task T_{ij} requires execution on processor P_j. Further with respect to processing speed, we will distinguish three classes of parallel processor systems. Let s_{ij} denote the processing speed of a job J_i on a processor P_j. A parallel processor system is said to be *uniform* if $s_{1j}=s_{2j}=\ldots=s_{nj}$ for all processors P_j, and otherwise the processor system is said to be *unrelated*. In uniform processors, $s_j=s_{1j}$ $(=s_{2j}=\ldots=s_{nj})$ denotes the speed of processor P_j. Specially, a uniform processor system in which all processors have the same speed is said to be *identical*.

The following notation is used for specifying the processor environment. Let ϕ denote the empty symbol.

(1) *TP* (the type of processors)

$TP=\phi$: one processor.

$TP=P$: identical processors.

$TP=Q$: uniform processors.

$TP=R$: unrelated processors.

$TP=F$: flowshop.

(2) m (the number of processors)

$m=\phi$: m is assumed to be a free parameter.

$m=k$: m is a constant equal to k (k is a positive integer).

$m\geqq k$: m is a constant greater than or equal to k.

2.2.2 Job characteristics

If a job cannot be interrupted once it has begun execution, the job is said to be *nonpreemptive*. Otherwise, the job is said to be *preemptive*.

A partial order < is defined on a given set of jobs, where $J_i < J_j$ signifies that J_i must be completed before J_j can begin. The partial order < is conveniently represented as *a dag* (*directed acyclic graph*) where each node corresponds to a job. Let us denote by "e" the number of edges in >.

Arrival time of a job denotes the time on which the job becomes available for processing.

Processing time of a job, denoted by t_i (or t_{ij}), is defined for each of three classes of parallel processor systems.

Identical processors: t_i is the time needed by a processor to complete J_i.

Uniform processors: t_i is the time needed by the slowest processor to complete J_i.

Unrelated processors: t_{ij} is the time needed by processor P_j to complete J_i.

The following notation is used for specifying job characteristics.

(1) *RULE* (rule of operation)

RULE=ϕ: each job is nonpreemptive.

RULE=*pmtn*: each job is preemptive.

(2) $<$ (partial order on a given set of jobs)

$<=\phi$: no precedence relation is specified.

$<$=*prec*: $<$ is an arbitrary dag.

$<$=*intree*: each node of $<$ has at most one outdegree, that is, each node has at most one successor.

$<$=*outtree*: each node of $<$ has at most one indegree, that is, each node has at most one predecessor.

$<$=*tree*: $<$ is either an intree or an outtree.

$<$=*chain*: each node of $<$ has at most one outdegree and at most one indegree.

(3) a_i (arrival times of jobs)

$a_i=0$: arrival times are equal to 0 for all jobs.

a_i: each job has an arbitrary arrival time.

(4) t_i (processing times of jobs)

$t_i=1$: each job has unit processing time.

$t_i=\phi$: each job has an arbitrary processing time.

2.2.3 Optimality criteria

Let C_i denote completion time of the job J_i in a given schedule, and we define two criteria for evaluating schedules as follows:

schedule-length $C_{max} = \max\{C_1,\ldots,C_n\}$;

mean flow-time $(1/n)\sum_{i=1}^{n}(C_i-a_i)$ or

mean weighted flow-time $(1/n)\sum_{i=1}^{n} w_i(C_i-a_i)$

where w_i denotes weight of job J_i. Minimization of mean weighted flow-time corresponds to minimizing total weighted completion time $\sum w_iC_i$ because n and $\sum w_ia_i$ are invariant with respect to schedules. Therefore in this paper we will denote mean weighted flow-time by $\sum w_iC_i$.

2.2.4 Three-field notation

When we need to distinguish scheduling problems in a short way, we will use three-field notation defined as follows [33]:

$$\alpha \mid \beta \mid \gamma$$

where

$$\alpha = (TP,\ m),$$
$$\beta = (RULE,\ <,\ a_i,\ t_i),$$
$$\gamma = C_{max} \text{ or } \sum w_i C_i.$$

3. Parallel Processor Problems With No Additional Resources

This section concerns parallel processor problems with no additional resources. Both cases of a_i=0 and arbitrary a_i are considered for each criterion of C_{max} and $\sum w_i C_i$.

3.1 Optimization algorithms

3.1.1 Minimizing schedule-length

Let us start from problems with a_i=0 and t_i=1. Table 1 summarizes the results for complexity of C_{max} minimization problems with a_i=0 and t_i=1. Some entries in the column < have not been explained yet. *An interval order* is a dag in which each job J_x corresponds to an interval $[a_x,\ b_x]$ on the real line, and J_x precedes J_y if and only if $a_y > b_x$. Also a dag is *a level order* if each connected component is partitioned into some k levels $L_1,\ldots,L_k$ such that for every two jobs $J_x \in L_i$ and $J_y \in L_j$ where $i > j$, J_x precedes J_y. *An opposing forest* is a disjoint union of intrees and outtrees. Moreover < in problem 1-7 represents a dag whose height is smaller than or equal to a constant h. Figure 1 shows an interrelationship among partial orders <, where "$a \leftarrow b$" means that a is included in b.

prec

series-parallel

opposing forest

interval order *intree* *outtree* *level order*

chain

ϕ

Figure 1. Interrelationship Among Partial Orders <.

Table 1. Complexity of C_{max} minimization problems with a_i=0 and t_i=1.

Problem	Parameters: a_i=0, t_i=1 and TP	m	RULE	<	Complexity	References
1-1	P			prec	strongly NP-hard	[66]
1-2	P			tree	$O(n\alpha(n))^*$	[37]
1-3	P			interval order	$O(n^2)$	[58]
1-4	P	≧ 3		prec	Open	
1-5	P	≧ 3		opposing forest	$O(n^{2m-2} \log n)$	[18]
1-6	P	≧ 3		level order	$O(n^{m-1})$	[18]
1-7	P	≧ 3		the height ≦ h	$O(n^{h(m-1)+1})$	[17]
1-8	P	2		prec	$O(e+n\alpha(n))^*$	[23]

* $\alpha(n)$ is an inverse of Ackermann's function and is very slow-growing. For example, if $n \leqq 2^{65536}$ then $\alpha(n) \leqq 5$.

The problem for *m*, a free parameter, is NP-hard if < is an arbitrary dag (problem 1-1). The problem remains NP-hard even if < is an opposing forest [27]. Polynomial algorithms are developed only for < = *tree* and for < = *interval order* (problems 1-2 and 1-3).

The problem for fixed $m \geqq 3$ can be solved in polynomial time for partial orders of special forms (problems 1-5, 1-6 and 1-7). But when < is an arbitrary dag (problem 1-4), complexity of the problem is open. Problem 1-4 is the major scheduling problem remaining open.

The m=2 case can be solved in polynomial time even if < is an arbitrary dag (problem 1-8). An algorithm by Coffman and Graham [14] requires time $O(e+n\alpha(n))$ if a dag is transitive reduced or transitive closed, and otherwise it requires more $O(\min(en, n^{2.61}))$ time [23]. On the other hand, Gabow [23] presents an $O(e+n\alpha(n))$ algorithm which uses neither transitive reduction nor transitive closure.

Next we will consider problems with a_i=0 and arbitrary t_i. Table 2 indicates the results for complexity of C_{max} minimization problems with a_i=0 and arbitrary t_i. The single processor problem with a_i=0 has at least one optimal schedule without preemptions even if preemptions are allowed. Thus an $O(n^2)$ algorithm for the nonpreemptive case of problem 2-1 can also solve the preemptive case. A pseudopolynomial time algorithm by Sahni [63] solves problem 2-2 in time $O(n(\sum t_i)^{m-1})$. But problem 2-3 is strongly NP-hard and has probably no pseudoplynomial time algorithm. Problem 2-5 which corresponds

Table 2. Complexity of C_{max} minimization problems with a_i=0 and arbitrary t_i.

Problem	Parameters: a_i=0, t_i=ϕ and				Complexity	References
	TP	*m*	*RULE**	<		
2-1		1	*pmtn*,ϕ	*prec*	$O(n^2)$	[49, 5]
2-2	*P*	≧ 2			NP-hard	[26]
2-3	*P*				strongly NP-hard	[26]
2-4		≧ 2		*interval order*	strongly NP-hard	[58]
2-5	*P*		*pmtn*		$O(n)$	[55]
2-6	*Q*		*pmtn*		$O(n+m \log m)$	[30]
2-7	*R*		*pmtn*		LP**	[51]
2-8	*P*		*pmtn*	*tree*	$O(n \log m)$	[31]
2-9	*P*	2	*pmtn*	*prec*	$O(n^2)$	[56, 12]
2-10	*Q*	2	*pmtn*	*prec*	$O(n^2)$	[36, 33]
2-11	*P*	≧ 3	*pmtn*	*prec*	Open	
2-12	*P*		*pmtn*	*prec*	strongly NP-hard	[12]

* An entry in the column *RULE* given as "*pmtn*,ϕ" means that the associated problem has at least one optimal schedule without preemptions even if preemptions are allowed.

** We denote by "LP" that the associated aproblem can be formulated into a linear programming, which is polynomially solvable.

to the preemptive case of problem 2-3 can be solved in time $O(n)$, and the more general problems (2-6, 2-7 and 2-8) can also be solved in polynomial time. The $O(n)$ algorithm for problem 2-5 is used as a subroutine in many of preemptive scheduling algorithms [39, 57]. The algorithm is as follows.

Algorithm McNaughton

Step 1. Set $D = \max\ (\sum_{i=1}^{n} t_i/m,\ \max\{t_i\})$.

Step 2. When scheduling job J_i, let j be the smallest index of a processor with idle time and let (s, D) be the idle interval on P_j. If $s+t_i \leqq D$, schedule job J_i in the interval $(s, s+t_i)$. Otherwise, schedule J_i in the interval (s, D) on P_j and $(0, t_i-(D-s))$ on P_{j+1}.

Further let us give a concise description of the algorithm for problem 2-6. If $X_j = \sum_{i=1}^{j} t_i$ and $Y_j = \sum_{i=1}^{j} s_i$ where $t_1 \geqq \dots \geqq t_n$ and $s_1 \geqq \dots \geqq s_m$, then the optimal value of C_{max} for the problem is given by $D = \max\ \{X_1/Y_1, X_2/Y_2,\dots,X_{m-1}/Y_{m-1}, X_n/Y_m\}$. In the algorithm, processors are viewed to constitute a composite processor system. Each processor P_j initially has a speed function $s_j(t)$ given by $s_j(t) = s_j$ for $0 \leqq t \leqq D$. Depending on its processing requirement, a job is scheduled in an interval (T,D) on a processor P_j, or the job is scheduled in an interval (T,D) on P_{j-1} and $(0,T)$ on P_j. In the former case, the speed function of P_j is newly given by $s_j(t) = s_j$ for $0 \leqq t \leqq T$, and $s_j(t) = 0$ for $T < t \leqq D$; and specially if $T=0$, P_j is deleted from the composite processor system. In the latter case, two processors P_{j-1} and P_j are replaced by a composite processor P_{j-1} with speed $s_{j-1}(t) = s_{j-1}$ for $0 \leqq t \leqq T$, and $s_{j-1}(t) = s_j$ for $T < t \leqq D$. For convenience sake, let

$$S_i(a,b) = \int_a^b s_i(t)dt \quad \text{and} \quad S_i = S_i(0,D).$$

Algorithm GS (Gonzalez and Sahni)

Step 1. Sort processors so that $s_1 \geqq \ldots \geqq s_m$. Set D to the optimal value of C_{max}, and $s_i(t) = s_i$ for $0 \leqq t \leqq D$.

Step 2. (We may schedule jobs in any order.) When scheduling job J_a, let k be the least integer satisfying $t_a \geqq S_k$.

<u>Case 1.</u> $t_a = S_k$. Schedule J_a in the interval $(0,D)$ on P_k, and subtract one from the index of P_j, $k+1 \leqq j \leqq m$. //comment: We now have a composite processor system with $m-1$ processors and have $n-1$ jobs to schedule. //

<u>Case 2.</u> $t_a > S_k$. Let T be the time satisfying $S_{k-1}(T,D) + S_k(0,T) = t_a$. Schedule J_a in the interval (T,D) on P_{k-1} and $(0,T)$ on P_k; and then replace P_{k-1} and P_k by a single processor (newly called P_{k-1}) with speed $s_{k-1}(t)$ for $0 \leqq t \leqq T$, and $s_k(t)$ for $T < t \leqq D$. Subtract one from the index of P_j, $k+1 \leqq j \leqq m$. //comment: Again, we are left with a composite processor system with $m-1$ processors and $n-1$ jobs to schedule.//

<u>Case 3.</u> $S_m > t_a$. Let T be the time satisfying $S_m(T,D) = t_a$. Schedule J_a in the interval (T,D) on P_m and let $s_m(t) = 0$ for $T \leqq t \leqq D$.

An algorithm by Muntz and Coffman solves problems 2-8 [57], 2-9 [56] and 2-10 [36] in $O(n^2)$ time, and solves problem 2-6 [36] in $O(mn^2)$ time. However as shown in table 2, more efficient algorithms are presented for problems 2-6 and 2-8. For arbitrary $<$, no polynomial time algorithm has been found yet even if $m=3$ and preemptions are allowed (see problem 2-11).

Table 3 gives the results for complexity of C_{max} minimization problems with arbitrary arrival times. It should be noted that minimization of C_{max}

for jobs $\{J_1,\ldots,J_n\}$ with arrival times a_i and a partial order < is equivalent to minimizing maximum lateness $L_{\max} = \max(C_i - d_i)$ for the jobs $\{J_1',\ldots,J_n'\}$ signified as follows: due dates, d_i, of J_i' are given by $d_i = -a_i$, respectively, and arrival times are equal for all J_i'; and moreover J_i' precedes J_j' in the partial order on $\{J_1',\ldots,J_n'\}$ if and only if J_j precedes J_i in <. For example, problem 3-1 (P | *outtree*, a_i, t_i=1 | $C_{\max}$) is equivalent to P | *intree*, t_i=1 | $L_{\max}$. Some of the results indicated in table 3 are presented, in the original papers, as results for $L_{\max}$ minimization problems.

Table 3. Complexity of $C_{\max}$ minimization problems with arbitrary a_i.

Problem	Parameters other than a_i *	Complexity	References
3-1	problem 1-2 (<=*outtree*)	$O(n)$	[9, 50]
3-2	problem 1-2 (<=*intree*)	strongly NP-hard	[9]
3-3	problem 1-8	$O(n^3)$	[25]
3-4	problem 2-1 (*RULE*=ϕ)	$O(n^2)$	[49, 5]
3-5	problem 2-1 (*RULE=pmtn*)	$O(n^2)$	[5]
3-6	problem 2-5	$O(mn)$	[31]
3-7	problem 2-6	$O(n \log n+mn)$	[64]
3-8	problem 2-7	LP	[51]
3-9	problem 2-8 (<=*outtree*)	$O(n^2)$	[50]
3-10	problem 2-8 (<=*intree*)	strongly NP-hard	[50]
3-11	problem 2-9	$O(n^2)$	[50]
3-12	problem 2-10	$O(n^2)$	[50]

* Each problem is the same as one entered in this column, except that a_i is arbitrary.

3.1.2 Minimizing mean weighted flow-time

Table 4 indicates the results for complexity of $\sum w_i C_i$ minimization problems with a_i=0. All of these results are indicated in [33]. The $\sum w_i C_i$ minimization problem with <=ϕ and a_i=0 has at least one optimal schedule without preemptions even if preemptions are allowed [55]. Thus an $O(n \log n)$ algorithm for the nonpreemptive case of problem 4-1 is also an optimization algorithm for the preemptive case. The same statement is true for problems 4-3 and 4-8. Moreover problems 4-10 and 4-11 are NP-hard and strongly NP-hard, respectively, whether preemptions are allowed or not. Although problem 4-10 is NP-hard, a pseudopolynomial time algorithm by Sahni [63] solves the problem in time $O(n(\sum t_i)^{m-1})$. If < is an arbitrary dag, even the single processor problem is strongly NP-hard (see problem 4-2). Moreover the two-processor problem is NP-hard even for the $\sum C_i$ criterion and <=*tree* (problem 4-7), and even for the $\sum w_i C_i$ criterion and <=ϕ (problem 4-10). It is an interesting open question whether the two-processor problem with the $\sum C_i$ criterion remains to be polynomial for <=*chain* (compare 4-3 with 4-7). An $O(n^3)$ algorithm by Adolphson [3] solves the single processor $\sum w_i C_i$ minimization problem for a slightly more general < than "*series-parallel*". Whether the problem is polynomial for a more general < is a subject for a future study (compare 4-2 with 4-9).

Results for arbitrary a_i are given in table 5. All of these results are indicated in [33].

Table 4. Complexity of $\sum w_i C_i$ minimization problems with a_i=0.

Problem	Parameters: a_i=0, t_i=ϕ and					Complexity
	w_i	*TP*	*m*	*RULE*	<	
4-1	1		1	*pmtn*, ϕ		$O(n \log n)$
4-2	1		1		*prec*	strongly NP-hard
4-3	1	*P*		*pmtn*, ϕ		$O(n \log n)$
4-4	1	*Q*				$O(n \log n)$
4-5	1	*Q*		*pmtn*		$O(n \log n+mn)$
4-6	1	*R*				$O(n^3)$
4-7	1	*P*	≧ 2		*tree*	strongly NP-hard
4-8			1	*pmtn*, ϕ		$O(n \log n)$
4-9			1		*series-parallel*	$O(n \log n)$
4-10		*P*	≧ 2	*pmtn*, ϕ		NP-hard
4-11		*P*		*pmtn*, ϕ		strongly NP-hard

Table 5. Complexity of $\sum w_i C_i$ minimization problems with arbitrary a_i.

Problem	Parameters other than a_i	Complexity
5-1	problem 4-1 (*RULE*=ϕ)	strongly NP-hard
5-2	problem 4-1 (*RULE*=*pmtn*)	$O(n \log n)$
5-3	problem 4-8 (*RULE*=ϕ)	strongly NP-hard
5-4	problem 4-8 (*RULE*-*pmtn*)	strongly NP-hard

3.2 Approximation algorithms

3.2.1 Minimizing schedule-length

Table 6 summarizies the results of performance and complexity of approximation algorithms for C_{max} minimization problems. First we give the following remarks to help interpret the table.

Table 6. Performance and complexity of approximation algorithms for C_{max} minimization problems.

Problem	R_A	Complexity	Algorithm	References
$P \mid prec \mid C_{max}$	$2-1/m$	$O(n \log m+e)$	LIST	[32,]
$P \mid prec, t_i=1 \mid C_{max}$	$2-1/(m-1)$	$O(e+n\alpha(n))$	HLF	[47, 23]
(problem 1-1)	$2-2/m$	$O(e+n\alpha(n))$	CG	[48, 23]
	$4/3-1/3m$	$O(n \log n)$	HLF	[32, 38]
$P \mid\mid C_{max}$	$(1.182, 1.2+2^{-k})$	$O(n \log n + kn \log m)$	MF	[20, 19]
(problem 2-3)	$\leq 7/6+2^{-k}$	$O(n \log n + km^4 n)$	DUAL	[34, 34]
$Q \mid\mid C_{max}$	$(1.512, 1.583)$	$O(n \log n + n \log m)$	modified HLF	[16, 19]
	$(1.341, 1.4+2^{-k})$	$O(n \log n + kn \log m)$	modified MF	[19, 19]
$R \mid\mid C_{max}$	$\leq 2.5\sqrt{m}$	$O(mn \log n)$		[15, 15]
	2	LP$+O(m^{m-1})$		[60, 60]
$P \mid intree \mid C_{max}$	$2-2/(m+1)$	$O(n \log n)$	HLF	[47,]
$P \mid chain \mid C_{max}$	$5/3$	$O(n \log n)$	HLF	[47,]
$P \mid pmtn, prec \mid C_{max}$ (problem 2-12)	$2-2/m$	$O(n^2)$	MC	[48, 12]
$Q \mid pmtn, prec \mid C_{max}$	$(0.35\sqrt{m}, 1.23\sqrt{m})$	$O(mn^2)$	MC	[36, 33]

(1) Each row in the table is associated with an algorithm.

(2) For an algorithm whose performance ratio is not exactly known, we give upper and lower bounds of the ratio in the column R_A. These bounds are best bounds currently known.

(3) Each complexity indicated in the table is the minimal one that the authors know.

(4) Each entry in the column on algorithm denotes the name by which the associated algorithm is called in this paper.

(5) The first entry in references gives the source for the performance of the associated algorithm, and the second entry indicates the source for complexity of the algorithm. If the second entry is missing, complexity of the associated algorithm is obtained by the authors.

Algorithms LIST and HLF can be used for the most general case, that is, for $RULE=\phi$, $<=prec$ and $t_i=\phi$.

Algorithm LIST (List Scheduling)

Step 1. An ordered list of jobs is constructed.

Step 2. Whenever a processor becomes available, the list is scanned from left to right; and the first unexecuted job that is ready for execution is assigned to the processor.

Before describing algorithm HLF, *the level of a job* on a partial order $<$ is defined as follows: (i) the level of a job with no successor is equal to the processing time of the job; and (ii) the level of a job with one or more successors is equal to the processing time of the job plus the maximal value of the levels of the successors of the job.

Algorithm HLF (Highest Level First)

Step 1. A priority list is constructed, where jobs are arranged in nonincreasing order of their levels.

Step 2. Do step 2 in algorithm LIST.

Specially, algorithm HLF for $<=\phi$ case is sometimes called *LPT* (*Largest Processing Time First*) strategy. The performance ratio of algorithm LIST is given by $2-1/m$ for $P \mid prec \mid C_{\max}$, and the performance ratio is the minimal one currently known for this problem. However, if $t_i=1$ (problem $P \mid prec, t_i=1 \mid C_{\max}$) or if $<$ has a special form (problems $P \mid\mid C_{\max}$, $P \mid intree \mid C_{\max}$ and $P \mid chain \mid C_{\max}$), then algorithm HLF gives a smaller ratio than $2-1/m$. Further, another algorithm which is abbreviated "CG" in the table gives a smaller ratio than HLF for $P \mid prec, t_i=1 \mid C_{\max}$. The algorithm has been previously described as an optimization algorithm for problem 1-8, and requires $O(\min(en, n^{2.61}))$ time if < is neither transitive reduced nor transitive closed. For $P \,||\, C_{\max}$, a number of approximation algorithms are presented in addition to HLF. Algorithms MF and DUAL shown in the table use approximation algorithms for *the bin-packing problem*. The problem can be viewed as one of packing n pieces with sizes t_i into a minimum number of bins of capacity C. *Algorithm FFD (First-Fit Decreasing)*, which is used in MF, generates a bin-packing by placing successively a piece with the largest size into the lowest indexed bin in which it fits. Let $FFD(C)$ be the function which returns 1 if algorithm FFD can pack all t_i into m bins of capacity C, and otherwise returns 0.

Algorithm MF (Multi-Fit)

Step 1. Initially set $L = \sum_{i=1}^{n} t_i/m$ and $U=2L$.

Step 2. Repeat the following operation k times: after substituting $(L+U)/2$ for C, set L to C if $FFD(C)=0$, and otherwise set U to C.

On the other hand, *an ε-dual approximation algorithm* for the bin-packing problem is defined as a polynomial time algorithm that can pack all pieces into $OPT(C)$ bins of capacity $(1+\varepsilon)C$, where $OPT(C)$ denotes the number of bins used by an optimal packing. Let $DUAL(\varepsilon,C)$ be the function which returns 1 if an ε-dual approximation algorithm can pack all t_i into m bins of capacity $(1+\varepsilon)C$, and otherwise returns 0. Algorithm DUAL consists of the same steps as MF, except that the function $DUAL(\varepsilon,C)$ is used instead of $FFD(C)$. For the bin-packing problem, an 1/5-dual approximation algorithm and an 1/6-dual approximation algorithm are presented in [34]. Using these algorithms, algorithm DUAL guarantees $R_{DUAL} \leqq 6/5+2^{-K}$ in $O(kn+n \log n)$ time, and guarantees $R_{DUAL} \leqq 7/6+2^{-k}$ in $O(knm^4+n \log n)$ time [34].

Besides the three algorithms indicated in table 6, Sahni [63] presents an algorithm which requires time $O(n(n^2/\varepsilon)^{m-1}$ in order to guarantee $R_A \leqq 1+\varepsilon$ for any $\varepsilon > 0$. Moreover an algorithm by Friesen and Langston [21] has $R_A=1.180$ and can be executed in the same time-complexity as algorithm MF. However, the algorithm contains the process of scheduling 32 jobs optimally on 5 processors. (Note that this process can be executed in $O(1)$.) Algorithm LIST has an advantage that it is available even though processing times of jobs are unknown. However, it is considerably inferior to algorithms HLF and MF in regard to the performance. R_{LIST} for $P \,||\, C_{\max}$ is given by $2-1/m$ [32], and this value is little improved even for jobs with similar processing times. It is shown in [1] that if $\max\{t_i\}/\min\{t_i\} \leqq 3$ then R_{LIST} is given by 17/10 for $m=5$, and is given by $2-1/\{3\lfloor m/3\rfloor\}$ for other $m \geq 3$.

Algorithms HLF and MF are modified for $Q \mid\mid C_{max}$. The modified HLF generates a schedule by successively placing a job with the largest processing time on the processor which would complete the job first. For the modified MF, the reader is refered to [19]. Problem $R \mid\mid C_{max}$ has an $O(m^m + mn \log n)$ algorithm with $R_A \leq 1.5\sqrt{m}$ in addition to the algorithms indicated in the table [15].

An algorithm by Muntz and Coffman, abbreviated "MC" in the table, can be used for $P \mid pmtn, prec \mid C_{max}$ and $Q \mid pmtn, prec \mid C_{max}$. The algorithm can be viewed as a highest-level-first strategy for the preemptive case. As stated in 3.1.1, the algorithm generates an optimal schedule for each of problems 2-6, 2-8, 2-9 and 2-10.

3.2.2 Minimizing mean weighted flow-time

Few results are known on the performance of approximation algorithms for $\sum w_i C_i$ minimization problems. For problem 4-10, Sahni [63] presents an approximation algorithm which requires $O(n(n^2/\varepsilon)^{m-1})$ time in order to guarantee $R_A \leq 1+\varepsilon$ for any $\varepsilon > 0$. This algorithm is efficient only for small m. Kawaguchi and Kyan [42] investigate the performance of the following algorithm for problems 4-10 and 4-11.

Algorithm LRF (Largest Ratio First)

Step 1. A priority list is constructed, where job J_i precedes job J_j if $w_i/t_i > w_j/t_j$, and precedence relations among jobs with the same w_i/t_i are arbitrary.

Step 2. Do step 2 in algorithm LIST.

Algorithm LRF can be executed in $O(n \log n)$ time. It is shown in [42] that R_{LRF} is given by $(\sqrt{2}+1)/2$ and R_{LRF} never takes a smaller value than $(\sqrt{2}+1)/2$ even if any priority rule is imposed among jobs with the same w_i/t_i. Moreover the problem of minimizing $\sum f_j^2$ is studied in [10] and [41], where f_j, $1 \leq j \leq m$, denote finishing times of processors P_j. For this problem, Chandra and Wong [10] present an $O(n \log n)$ algorithm with $R_A \leq 25/24$, and Kawaguchi [41] gives an $O(n \log m + m \log n)$ algorithm with $R_A=9/8$. These results are applicable to a special case of problem 4-11 where jobs have the same w_i/t_i.

An algorithm for $Q2 \mid\mid \sum w_i C_i$ [35] requires $O(n^2/\varepsilon)$ time in order to guarantee $R_A \leq 1+\varepsilon$ for any $\varepsilon > 0$. A modified LRF algorithm for $Q \mid\mid \sum w_i C_i$ [43] guarantees $R_A \leq \min\{\sum_{j=1}^{m} s_j/s_1, (\sqrt{2}+1)s_1/2s_m\}$ where $s_1 \geq \ldots \geq s_m$.

4. Parallel Processor Problems with Additional Resources

In this section we will deal with parallel processor problems in which each job requires the use of a resource in addition to a processor during their execution. This additional resource is called "memory" in this section. Let r_i be a fixed memory requirement of job J_i. Two types of memories are considered.

MEM=loc (local memories): each processor P_j has a private memory of capacity M_j, and job J_i can be processed on P_j if and only if $M_j \geqq r_i$.

MEM=com (common memory): a single memory of capacity M is shared among all processors, and we require that the total usage of the memory never exceeds the memory capacity M at a time.

4.1 Optimization algorithms

Table 7 summarizies the results for complexity of $C_{\max}$ minimization problems with memory constraints. The problem with local memories can be solved polynomially for preemptive cases (problems 7-1 and 7-3). Problem 7-1 has the same parameters as problem 2-5, except for the existence of memory constraints. The algorithm for problem 7-1 is as follows.

Algorithm KS (Kafura and Shen)

Step 1. Sort processors so that $M_1 \geqq M_2 \geqq \dots \geqq M_m$, and arrange jobs in non-increasing order of their memory requirements. We define the sets G_i as follows: $G_i = \{J_k \mid M_i \geqq r_k > M_{i+1}\}$ for $1 \leqq i < m$, and $G_m = \{J_k \mid M_m \geqq r_k\}$. Moreover for each i, $1 \leqq i \leqq m$, let $F_i = \bigcup_{k=1}^{i} G_k$ and let X_i be the sum of processing times of jobs in F_i.

Step 2. Set $D = \max(X_1, X_2/2, \dots, X_m/m, \max\{t_i\})$ and apply step 2 in algorithm McNaughton.

The algorithm for problem 7-3 uses algorithm GS for problem 2-6 as a subroutine. Further problem 7-2 is strongly NP-hard because problem 2-3 is so.

The most of problems with MEM=com are NP-hard even if $t_i=1$. Problem 7-7 remains to be strongly NP-hard even if the memory capacity is 1 and memory requirements are 0 or 1 for all jobs [8]. On the other hand, problem 7-5 remains to have a polynomial time algorithm even if each job requires $s \geqq 1$ memories during their execution. This algorithm uses the maximum cardinality matching, and requires time $O(sn^2+n^{5/2})$ [8].

If the C_{max} criterion is replaced by the mean flow-time $\sum C_i$, the problem corresponding to 7-4 is also strongly NP-hard and the problem corresponding to 7-5 can be solved in $O(n \log n)$ time using the algorithm for problem 7-5 [7].
corresponding to 7-4 is also strongly NP-hard and the problem corresponding to 7-5 can be solved in $O(n \log n)$ time using the algorithm for problem 7-5 [7].

Table 7. Complexity of C_{max} minimization problems with memory constraints.

Problem	Parameters: a_i=0 and						Complexity	References
	MEM	*TP*	*m*	*RULE*	<	t_i		
7-1	*loc*	*P*		*pmtn*			$O(n \log m)$	[39, 54]
7-2	*loc*	*P*					strongly NP-hard	
7-3	*loc*	*Q*		*pmtn*			$O(nm(\log m)^2)$	[54]
7-4	*com*	*P*	3			1	strongly NP-hard	[26]
7-5	*com*	*P*	2			1	$O(n \log n)$	[8]
7-6	*com*	*Q*	2			1	$O(n \log n)$	[8]
7-7	*com*	*P*	2		*chain*	1	strongly NP-hard	[8]
7-8	*com*	*P*	$m \geqq n$			1	strongly NP-hard	[26]

Table 8. Absolute and asymptotic performance of approximation algorithms for C_{max} minimization problems with memory constraints.

Problem	Parameters: a_i=0 and						R_A	R_A^∞	References*
	MEM	*TP*	*m*	*RULE*	<	t_i			
7-2	*loc*	*P*					$2-2/m$		[39,]
8-1	*com*	*P*				1		$2-2/m$	[, 46]
8-2	*com*	*P*		*pmtn*			$3-3/m$		[46,]
7-8	*com*	*P*	$m \geqq n$			1	$\leqq 2$	1	[13, 13]
8-3	*com*	*P*	$m \geqq n$				2.5	5/4	[65, 4]
8-4	*com*	*P*	$m \geqq n$		*prec*	1	$\leqq 2.7$		[24,]

* The first entry gives the source for R_A and the second entry indicates the source for R_A^∞.

4.2 Approximation algorithms

Besides the absolute performance ratio R_A, the asymptotic performance ratio R_A^∞ is often used for evaluating approximation algorithms for problems with memory constraints. Table 8 summarizies the results of the absolute and the asymptotic performances of approximation algorithms for C_{max} minimization problems with memory constraints. Each entry in the columns R_A and R_A^{∞} is the minimum one currently known. The reader should notice that even if a problem has no entry in the column R_A, an algorithm with the absolute performance ratio R_A guarantees $R_A^{\infty} \leq R_A$ for the problem. The algorithm for problem 8-1 is as follows.

Algorithm LMF (Largest Memory First)

Step 1. Sort jobs so that $r_1 \geqq \dots \geqq r_n$, and set t=0.

Step 2. If no processor is idle at time t, set t=t+1 and go to step 3. Otherwise, scan for the first unexecuted job for which sufficient units of memory are available at time t. If such job exists, the job is assigned to an idle processor in the interval (t, t+1); and otherwise set t=t+1.

Step 3. If there are uncompleted jobs, repeat step 2.

The algorithm for problem 8-2 is also the above one. Moreover the ratio 3-3/m is guaranteed for problem 8-2 even if the ordering of jobs is arbitrary in step 1 of LMF. The algorithm for problem 7-2 is similar to LMF. In this algorithm, whenever a processor P_j becomes available, a job with the largest r_i satisfying $M_j \geqq r_i$ is assigned to the processor. Fuchs and Kafura [22] study the C_{max} minimization problem in a parallel computer system where each computer consists of dual processors and a single memory is shared between these processors. This model can be viewed as a bridge between the extreme models (problems 7-2 and 8-2) and the more general model where $r \geqq 1$ processors share a single memory. A largest-memory-first strategy for this model guarantees the absolute performance ratio $(3p-2)/p$ where p denotes the number of computers and is equal to half the number of processors. Comparing the results for 7-2, 8-2 and their model, they conclude that the memory should be shared among a small number of processors. This is an important suggestion about the topology of computer system.

Problem 7-8 is known as the bin-packing problem, and an approximation algorithm for this problem has been described in 3.2.1. This algorithm, called FFD, gives R_{FFD}^{∞}=11/9. Note that a bin B_t, $t \geqq 1$, corresponds to a time interval (t-1, t). Moreover the following results are known for the problem [13]:

$$R_{FF}^{\infty} = 17/10 \text{ and } R_{MFFL}^{\infty} = 71/60$$

where MFFD is a modified FFD algorithm, and FF is the same as FFD except that jobs are scheduled in an arbitrary order. Further as shown in the table, the problem has a polynomial time algorithm with $R_A^{\infty}=1$. However no polynomial time algorithm that guarantees the absolute performance ratio smaller than 2 is found yet for the problem. On the other hand, even a more primitive algorithm than FF guarantees $R_A \leqq 2$. In this algorithm, called *Next-Fit*, a job is placed in the current bin B_t if it fits there, and otherwise the job is placed in the next bin B_{t+1} and t is increased by one. (Remember that FFD (or FF) places a job in the lowest indexed bin in which it fits.)

Problem 8-3 is known as the *two-dimensional bin-packing problem*. A number of results are reported on the asymptotic performance of algorithms for this problem, but few results are known on the absolute one. Moreover Garey et al. [24] presents an algorithm for the problem in which the number of memory is an arbitrary $s \geqq 1$ and the other parameters are the same as problem 8-4. The algorithm uses a highest-level-first strategy and guarantees $R_A \leqq 17s/10+1$. The bound for problem 8-4 is obtained by substituting $s=1$ in the above bound.

Very little is known about the performance of approximation algorithms for $\sum w_i C_i$ minimization problems with memory constraints.

5. Flowshop Scheduling

A flowshop consists of $m \geqq 2$ processors $\{P_1,\ldots,P_m\}$ and $n \geqq 1$ jobs $\{J_1, \ldots,J_n\}$. Each job J_i has a chain of m tasks T_{ij}, $1 \leqq j \leqq m$. Let t_i denote *the processing time of job* J_i and let t_{ij} be *the processing time of task* T_{ij}. Each task T_{ij} requires execution on processor P_j, and T_{ij} can only be executed after $T_{i,j-1}$ is finished. Specially, if T_{ij} has to start at the completion time of $T_{i,j-1}$, the resulting flowshop is said to be *a no-wait flowshop*. Before describing results of flowshop problems, we give the following notes about the notation used in this section.

(1) Whenever a three field notation $\alpha|\beta|\gamma$ is used, an information about no-wait constraints is put in the first position of β.

(2) We denote by "$t_{i1}=1$ (or $t_{i2}=1$)" when $t_{11}=t_{21}=\ldots=t_{n1}$ (or $t_{12}=t_{22}=\ldots=t_{n2}$). Moreover, the notation "$t_{ij}=1$" represents that all t_{ij}, $1 \leqq i \leqq n$ and $1 \leqq j \leqq m$, are the same.

Table 9. Complexity of flowshop problems and no-wait flowshop problems.

Problem	Parameters	Complexity
9-1	$F2 \| \| C_{max}$	$O(n \log n)$
9-2	$F2 \| pmtn \| C_{max}$	
9-3	$F2 \| a_i \| C_{max}$	strongly NP-hard
9-4	$F2 \| pmtn, a_i \| C_{max}$	
9-5	$F2 \| tree \| C_{max}$	strongly NP-hard
9-6	$F2 \| pmtn, tree \| C_{max}$	
9-7	$F2 \| \text{no-wait} \| C_{max}$	$O(n \log n)$
9-8	$F3 \| \| C_{max}$	strongly NP-hard
9-9	$F3 \| pmtn \| C_{max}$	
9-10	$F3 \| no\text{-}wait \| C_{max}$	strongly NP-hard
9-11	$F2 \| \| \sum C_i$	strongly NP-hard
9-12	$F2 \| pmtn \| \sum C_i$	Open
9-13	$F2 \| no\text{-}wait \| \sum C_i$	strongly NP-hard

5.1 Optimization algorithms

Table 9 gives the results for complexity of flowshop problems and no-wait flowshop problems. The results for problems 9-10 and 9-13 were recently obtained by Röck [61, 62]. For the sources of the other results, the reader is refered to [52]. Gonzalez and Sahni [29] show that preemptions on P_1 and P_m can be removed without increasing C_{max} in any preemptive schedule for a flowshop with m processors. Hence, the result for each of problems 9-2, 9-4, 9-6 and 9-9 is obtained from the result for the nonpreemptive case of the problem.

Achugbue and Chin [2] study j-maximal and j-minimal flowshops. A flowshop is said to be *j-maximal* (*j-minimal*) if the j-th task of each job is not smaller than (not greater than) any other task of the same job. Problem $F \mid 3 \mid C_{max}$ remains to be strongly NP-hard even if the flowshop is 1 or 3-minimal, and even if the flowshop is 1 or 3-maximal. Although problems 9-5 and 9-11 are strongly NP-hard, $F2 \mid tree, t_{ij}=1 \mid C_{max}$ and $F2 \mid tree, t_{ij}=1 \mid \sum C_i$ can be solved in $O(n)$ time and $F \mid a_i, t_{ij}=1 \mid \sum C_i$ can be solved in $O(n^3)$ time [7]. Further $F2 \mid t_{i2}=1 \mid \sum C_i$ and the 1-maximal case of $F2 \mid\mid \sum C_i$ can be solved in $O(n \log n)$ time [40].

Papadimitriou and Kanellakis [59] study an extension model of $F2 \mid\mid C_{max}$ where the flowshop has $b \geqq 0$ buffers between P_1 and P_2. Each buffer is used for storing a job whose first task has been completed on P_1, and whose second task has not been started on P_2. They show that the problem is strongly NP-hard if $1 \leqq b < n$.

5.2 Approximation algorithms

Problem $F \mid no\text{-}wait \mid C_{max}$ can be formulated as a traveling salesman problem [33]. A number of approximation algorithms are presented for this problem, and so these algorithms can be applied to $F \mid no\text{-}wait \mid C_{max}$. Further, approximation algorithms which can be viewed as list scheduling for flowshop problems are presented for $F \mid\mid C_{max}$ and $F \mid\mid \sum C_i$.

Algorithm LISTF (List Scheduling for Flowshops)

Step 1. Construct a list in which jobs are arranged in an arbitrary order.

Step 2. Let L_j, $1 \leqq j \leqq m$, denote m copies of the list obtained in step 1. Whenever a processor P_j becomes available, the first job is removed from L_j and the j-th task of the job is assigned to the processor.

Algorithm SPT (Shortest Processing time First) applies step 2 of the above algorithm to a list where jobs are arranged in nondecreasing order of their processing times. The following results are known about the performance of these algorithms [29]:

$$R_{LISTF} = m \quad \text{for } F \mid\mid C_{max},$$

$$R_{LISTF} = n \text{ and } R_{SPT} = m \quad \text{for } F \mid\mid \sum C_i.$$

The performance of algorithm LISTF is not improved for $F \mid\mid C_{max}$ even if jobs are processed in nonincreasing order of their processing times. Specially for $F2 \mid\mid \sum C_i$, the performance of SPT is given by $R_{SPT} = 2\beta/(\beta+\alpha)$ where α denotes the smallest processing time of tasks and β is the largest one [40]. Chin and Tsai [11] investigate the performance of algorithm LISTF for j-maximal and j-minimal flowshop problems, and give the following results:

$$R_{LISTF} = m-1 \quad \text{for the 1-minimal case of } F \mid\mid C_{max} \ (m \geqq 3),$$

$$R_{LISTF} \leqq 1/2+\sqrt{m} \quad \text{for the 1-maximal case of } F \mid\mid C_{max},$$

$$R_{LISTF} = 5/3 \quad \text{for the 1 or 3-maximal case of } F3 \mid\mid C_{max}.$$

Further Kawaguchi and Kyan [44] investigate the performance of an algorithm for $F2 \mid t_{i1}=1 \mid \sum C_i$ and the 1-minimal case of $F2 \mid\mid \sum C_i$. This algorithm is also a list scheduling algorithm and schedules jobs according to nondecreasing order of t_{i2}. The performance ratio of this algorithm is given by 5/3 for the 1-minimal case of $F2 \mid\mid \sum C_i$, and is given by 4/3 for $F2 \mid t_{i1}=1 \mid \sum C_i$. (As shown in 5.1, $F2 \mid t_{i2}=1 \mid \sum C_i$ and the 1-maximal case of $F2 \mid\mid \sum C_i$ can be solved optimally in time $O(n \log n)$.)

Problem $F \,||\, C_{max}$ has a polynomial time algorithm which is superior to LISTF in regard to the performance. This algorithm guarantees $R_A \leqq \lceil m/2 \rceil$ in $O(mn \log n)$ time [29]. However, no polynomial time algorithm with the smaller performance than R_{SPT} is found yet for the $\sum C_i$ minimization flowshop problem, except for the special cases described above.

An optimization algorithm for $F2 \mid no\text{-}wait \mid C_{max}$ [28] generates a feasible schedule for an extension model of $F2 \,||\, C_{max}$ where the flowshop has $b \geqq 0$ buffers between P_1 and P_2 (see section 5.1). This algorithm guarantees $R_A \leqq (2b+1)/(b+1)$ in $O(n \log n)$ time [59].

6. Conclusion

We have briefly surveyed the recent results of optimization and approximation algorithms for deterministic models of computer scheduling. Many interesting models have efficient optimization algorithms, whereas others are NP-hard. A number of approximation algorithms have been presented for the latter problems, and analytical results for the performance of approximation algorithms are obtained mainly for schedule-length minimization problems with equal arrival times. However at present, many models of computer scheduling have neither polynomial time optimization algorithms nor approximation algorithms with guaranteed performance. Thus it is still important to evaluate goodness of an approximation algorithm by means of computational experiments, and such evaluation requires us to find a lower bound on the cost of an optimal schedule for a given problem instance. (Lower bounds are also used in enumerative optimization algorithms such as branch-bound-bound method.) Various bounding schemes have been reported in recent years although we have not described these schemes in this survey on account of limited space. These schemes can be further improved in the future.

Acknowledgement

The authors would like to thank Pr. Y. Nakanishi, Dr. H. Nakano and Dr. H. Ishii of Osaka University for their valuable suggestions. The first author was supported by the Grant-in-Aid for Encouragement of Young Scientists of the Ministry of Education, Science and Culture of Japan under Grant: (A) 61750343 (1986). The second author was also supported by the Grant-in-Aid for Co-operative Research of the Ministry of Education, Science and Culture of Japan under Grant: (A) 61302080 (1986).

References

[1] Achugbue, J. O. and F. Y. Chin: Bounds on Schedules for Independent Tasks with Similar Execution Times. *J. Assoc. Comput. Mach.*, Vol.28 (1981), 81-99.

[2] Achugbue, J. O. and F. Y. Chin: Complexity and Solutions of Some Three Stage Flow-Shop Scheduling Problems. *Math. Oper. Res.*, Vol.7 (1982), 532-544.

[3] Adolphson, D. L.: Single Machine Job Sequencing with Precedence Constraints. *SIAM J. Comput.*, Vol.6 (1977), 40-54.

[4] Baker, B. S., D. J. Brown and H. P. Katseff: A 5/4 Algorithm for Two Dimensional Packing. *J. Algorithms*, Vol.2 (1981), 348-368.

[5] Baker, K. R., E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan: Preemptive Scheduling of a Single Machine to Minimize Maximum Cost Subject to Release Dates and Precedence Constraints. *Oper. Res.*, Vol.31 (1983), 381-386.

[6] Blazewicz, J.: Selected Topics in Scheduling Theory. *Ann. Discrete. Math.*, Vol.31 (1987), 1-59.

[7] Blazewicz, J., W. Cellary, R. Slowinski and J. Weglarz: *Scheduling Under Resource Constraints - Deterministic Model (Ann. Oper. Res.,Vol.7)*, J. C. Baltzer AG, 1986.

[8] Blazewicz, J., J. K. Lenstra and A. H. G. Rinnooy Kan: Scheduling Subject to Resource Constraints - Classification and Complexity. *Discrete Appl. Math.*, vol.5 (1983), 11-24.

[9] Brucker, P., M. R. Garey and D. S. Johnson: Scheduling Equal-Length Tasks Under Tree-Like Precedence Constraints to Minimize Maximum Lateness. *Math. Oper. Res.*, Vol.2 (1977), 275-284.

[10] Chandra, A. K. and C. K. Wong: Worst-Case Analysis of a Placement Algorithm Related to Storage Allocation. *SIAM J. Comput.*, Vol.4 (1975), 249-263.

[11] Chin, F. Y. and L. L. Tsai: On J-maximal and J-minimal Flow-Shop Schedules. *J. Assoc. Comput. Mach.*, Vol.28 (1981), 462-476.

[12] Coffman, E. G. Jr (ed.): *Computer and Job-Shop Scheduling Theory*. John-Wiley, 1976.

[13] Coffman, E. G. Jr, M. R. Garey and D. S. Johnson: Approximation Algorithms for Bin-Packing - an Updated Survey. In *Algorithm Design for Computer System Design* (CISM Courses and Lectures, No.284), Springer-Verlag, 1984, 49-106.

[14] Coffman, E. G. Jr and R. L. Graham: Optimal Scheduling for Two Processo

Systems. *Acta Informatica*, Vol.1 (1972), 200-213.

[15] Davis, E. and J. M. Jaffe: Algorithms for Scheduling Tasks on Unrelated Processors. *J. Assoc. Comput. Mach.*, Vol.28 (1981), 721-736.

[16] Dobson, G.: Scheduling Independent Tasks on Uniform Processors. *SIAM J. Comput.*, Vol.13 (1984), 705-716.

[17] Dolev, D. and M. K. Warmuth: Scheduling Precedence Graphs of Bounded Height. *J. Algorithms*, Vol.5 (1984), 48-59.

[18] Dolev, D. and M. K. Warmuth: Profile Scheduling of Opposing Forests and Level Orders. *SIAM J. Alg. Disc. Meth.*, Vol.6 (1985), 665-687.

[19] Friesen, D. K. and M. A. Langston: Bounds for MULTIFIT Scheduling on Uniform Processors. *SIAM J. Comput.*, Vol.12 (1983), 60-70.

[20] Friesen, D. K.: Tighter Bounds for the MULTIFIT Processor Scheduling Algorithm. *SIAM J. Comput.*, Vol.13 (1984), 170-181.

[21] Friesen, D. K. and M. A. Langston: Evaluation of a MULTIFIT-Based Scheduling Algorithm. *J. Algorithms*, Vol.7 (1986), 35-59.

[22] Fuchs, K. and D. Kafura: Memory-Constrained Task Scheduling on a Network of Dual Processors. *J. Assoc. Comput. Mach.*, Vol.32 (1985), 102-129.

[23] Gabow, H. N.: An Almost Linear-Algorithm for Two-Processor Scheduling. *J. Assoc. Comput. Mach.*, Vol.29 (1982), 766-780.

[24] Garey, M. R., R. L. Graham, D. S. Johnson and A. C. C. Yao: Resource Constrained Scheduling as Generalized Bin Packing. *J. Combinatorial Theory Ser A*, Vol.21 (1976), 257-298.

[25] Garey, M. R. and D. S. Johnson: Two-Processor Scheduling with Start-Times and Deadlines. *SIAM J. Comput.*, Vol.6 (1977), 416-426.

[26] Garey, M. R. and D. S. Johnson: *Computers and Intractability - A Guide to the Theory of NP-completeness*. Freeman, 1979.

[27] Garey, M. R., D. S. Johnson, R. E. Tarjan and M. Yannakakis: Scheduling Opposing Forests. *SIAM J. Alg. Disc. Meth.*, Vol.4 (1983), 72-93.

[28] Gilmore, P. C. and R. E. Gomory: Sequencing a One-State Variable Machine - a Solvable Case of the Traveling Salesman Problem. *Oper. Res.*, Vol.12 (1964) 655-679.

[29] Gonzalez, T. and S. Sahni: Flowshop and Jobshop Schedules - Complexity and Approximation. *Oper. Res.*, Vol.26 (1978), 36-52.

[30] Gonzalez, T. and S. Sahni: Preemptive Scheduling of Uniform Processor Systems. *J. Assoc. Comput. Mach.*, Vol.25 (1978), 92-101.

[31] Gonzalez, T. and D. B. Johnson: A New Algorithm for Preemptive Scheduling of Trees. *J. Assoc. Comput. Mach.*, Vol.27 (1980), 287-312.

[32] Graham, R. L.: Bounds on Multiprocessing Timing Anomalies. *SIAM J. Appl. Math.*, vol.17 (1969), 416-429.

[33] Graham, R. L., E. L. Lawler, J. K. Lenstra and A. H. G. Rinnooy Kan: Optimization and Approximation in Deterministic Sequencing and Scheduling - a Survey. *Ann. Discrete Math.*, Vol.5 (1979), 287-326.

[34] Hochbaum, D. S. and D. B. Shmoys: Using Dual Approximation Algorithms for Scheduling Probelms - Theoretical and Practical Results. *Proc. 26th IEEE Symp. on Foundatoins of Computer Science*, 1985, 78-89.

[35] Horowitz, E. and S. Sahni: Exact and Approximate Algorithms for Scheduling Nonidentical Processors. *J. Assoc. Comput. Mach.*, Vol.23 (1976), 317-327.

[36] Horvath, E. C., S. Lam and R. Sethi: A Level Algorithm for Preemptive Scheduling. *J. Assoc. Comput. Mach.*, Vol.24 (1977), 32-43.

[37] Hu, T. C.: Parallel Sequencing and Assembly Line Problems. *Oper. Res.*, Vol.9 (1961), 841-848.

[38] Ishii, H.: Approximation Algorithms for Scheduling Problems. *Comm. Oper. Res. Soc. Japan*, Vol.31 (1986), 26-35 (in Japanese).

[39] Kafura, D. G. and V. Y. Shen: Task Scheduling on a Multiprocessor System with Independent Memories. *SIAM J. Comput.*, Vol.6 (1977), 167-187.

[40] Kawaguchi, T.: On List Scheduling for the Mean Flow-Time Flowshop Problem. *Trans. IECE Japan*, Vol. J 66-A (1983), 516-523 (in Japanese).

[41] Kawaguchi, T.: A Linear Time Placement Algorithm Related to Storage Allocation. *Trans. IECE Japan*, Vol. J 66-A (1983), 1023-1024 (in Japanese).

[42] Kawaguchi, T. and S. Kyan: Worst Case Bound of an LRF Schedule for the Mean Weighted Flow-Time Problem. *SIAM J. Comput.*, Vol.15 (1986), 1119-1129.

[43] Kawaguchi, T. and S. Kyan: Recent Topics on Scheduling Algorithms. *Proc. 7th Mathematical Programming Symposium Japan*, 1986, 143-157 (in Japanese).

[44] Kawaguchi, T. and S. Kyan: Bounds on Permutation Schedules for the Mean Finishing Time Flowshop Problem. *Paper of Technical Group on Circuit and System, CAS 86-174, IECE Japan*, 1987, 1-8.

[45] Kise, H.: On Recent Topics of the Machine Scheduling Theory. *Proc. 2nd Mathematical Programming Symposium Japan*, 1981, 81-92 (in Japanese).

[46] Krause, K. L., V. Y. Shen and H. D. Schwetman: Analysis of Several Task Scheduling Algorithms for a model of Multiprogramming Computer Systems. *J. Assoc. Comput. Mach.*, Vol.22 (1975), 522-550.

[47] Kunde, M.: Nonpreemptive LP-Scheduling on Homogeneous Multiprocessor Systems. *SIAM J. Comput.*, Vol.10 (1981), 151-173.

[48] Lam, S. and R. Sethi: Worst Case Analysis of Two Scheduling Algorithms.

SIAM J. Comput., Vol.6 (1977), 518-536.

[49] Lawler, E. L.: Optimal Sequencing of a Single Machine Subject to Precedence Constraints. *Management Sci.*, Vol.19 (1973), 544-546.

[50] Lawler, E. L.: Preemptive Scheduling of Precedence-Constrained Jobs on Parallel Machines. In *Deterministic and Stochastic Scheduling* (ed. M. A. H. Dempster et al.), Reidel, 1982, 101-123.

[51] Lawler, E. L. and J. Labetoulle: On Preemptive Scheduling of Unrelated Parallel Processors by Linear Programming. *J. Assoc. Comput. Mach.*, Vol. 25 (1978) 612-619.

[52] Lawler, E.L., J. K. Lenstra and A. H. G. Rinnooy Kan: Recent Developments in Deterministic Sequencing and Scheduling - a Survey. In *Deterministic and Stochastic Scheduling* (ed. M. A. H. Dempster et al.), Reidel, 1982, 35-73.

[53] Lenstra, J. K., A. H. G. Rinnooy Kan and P. Brucker: Complexity of Machine Scheduling Problems., *Ann. Discrete Math.*, Vol.1 (1977), 343-362.

[54] Martel, C.: Preemptive Scheduling to Minimize Maximum Completion Time on Uniform Processors with Memory Constraints. *Oper. Res.*, Vol.33 (1985),

[55] McNaughton, R.: Scheduling with Deadlines and Loss Functions. *Management Sci.*, Vol.6 (1959), 1-12.

[56] Muntz, R. R. and E. G. Coffman, Jr: Optimal Preemptive Scheduling on Two Processor Systems. *IEEE Trans. Computer*, Vol.c-18 (1969), 1014-1020.

[57] Muntz, R. R. and E. G. Coffman, Jr: Preemptive Scheduling of Real Time Tasks on Multiprocessor Systems. *J. Assoc. Comput. Mach.*, Vol.17 (1970), 324-338.

[58] Papadimitriou, C. H. and M. Yannakakis: Scheduling Interval-Ordered Tasks. *SIAM J. Comput.*, Vol.8 (1979), 405-409.

[59] Papadimitriou, C. H. and P. C. Kanellakis: Flowshop Scheduling with Limited Temporary Storage. *J. Assoc. Comput. Mach.*, Vol.27 (1980), 533-549.

[60] Potts, C. N.: Analysis of a Linear Programming Heuristic for Scheduling Unrelated Parallel Machines. *Discrete Appl. Math.*, Vol.10 (1985), 155-164.

[61] Röck, H.: The Three-Machine No-Wait Flow Shop Is NP-Complete. *J. Assoc. Comput. Mach.*, Vol.31 (1984), 336-345.

[62] Röck, H.: Some new results in flowshop scheduling. *Zeitschrift für Operations Research*, Vol.28 (1984), pp.1-16.

[63] Sahni, S.: Algorithms for Scheduling Independent Tasks. *J. Assoc. Comput. Mach.*, Vol.23 (1976), 116-127.

[64] Sahni, S. and Y. Cho: Scheduling Independent Tasks with Due Times on a

Uniform Processor System. *J. Assoc. Comput. Mach.*, Vol.27 (1980), 550-563.

[65] Sleator, D. K. D. B.: A 2.5 Times Optimal Algorithm for Bin Packing in Two Dimensions. *Information Processing Lett.*, Vol.10 (1980),37-40.

[66] Ullman, J. D.: NP-Complete Scheduling Problems. *J. Comput. Syst. Sci.*, Vol.10 (1975), 384-393.

Tsuyoshi KAWAGUCHI and Seiki KYAN:
Department of Electronics and
Information Engineering, Faculty of
Engineering, University of the Ryukyus
Nishihara, Okinawa 903-01, Japan