European Journal of Operational Research 47 (1990) 271-292
North-Holland

Invited Review

A state-of-the-art review of parallel-machine scheduling research

T.C.E. CHENG
Department of Actuarial and Management Sciences, University of Manitoba, Winnipeg, Manitoba, Canada R3T 2N2

C.C.S. SIN
Department of Management Sciences, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

Abstract: In this paper the major research results in deterministic parallel-machine scheduling theory will pass a survey. The review reveals that there exist a lot of potential areas worthy of further research.

Keywords: Scheduling, sequencing, classification

1. Introduction

Multiple-machine scheduling theory is the study of constructing schedules of machine processing for a set of jobs in order to ensure the execution of all jobs in the set in a reasonable amount of time. Three issues are to be dealt with:

What machines to be allocated to which jobs?

How to order the jobs in an appropriate processing sequence?

How to rationalize the 'reasonableness' of the schedule?

In other words, the major concern of multiple-machine scheduling theory is how to provide a perfect match, or near perfect match, of machines to jobs and subsequently determine the processing sequence of the jobs on each machine in order to achieve some prescribed goal.

It is not difficult to obtain real life examples of multiple-machine scheduling. For applications in computer sciences, some micro-computers have a mathematical co-processor to take care of the mathematical operations of the computer programs in addition to the central processing unit. In flow-line types of production systems, there should always be more than one machine of each kind available to avoid the entire production line being held up due to the failure of a single machine. Moreover, having more than one machine of a kind reduces the possibility of a bottleneck situation occurring. Adding an appropriate number of machines to a job shop production system can increase the routing flexibility which, in turn, can reduce the throughput time of a batch of jobs (Muramatsu et al., 1985). In a petrol station with several petrol pumps which are all connected to the same storage tank, the rate of petrol dispatching will be diminished as more pumps are used (Dror et al., 1987).

A simplification of the multiple-machine system is to consider the system as one aggregated facility. Therefore, the results of single-machine scheduling theory can be applied to study multiple-machine systems. The limitation of this approach is that the model, focusing only on the

Received January 1990

macro performance of the system, loses sight of the micro performance of the individual machines. To differentiate the performance of individual machines is impossible. In spite of this simplification, the scheduling of a single aggregated facility is by no means an easy problem.

Mathematically, multiple-machine scheduling problems are even more challenging. We recall that a multiple-machine scheduling problem involves a similar level of complexity as a single-machine scheduling problem to which an additional level of complexity is added due to allocation of machines. Thus, the overall complexity of the multiple-machine scheduling problem is inflated, often exponentially, as the number of machines or jobs or both increases.

Until recently, only a few multiple-machine scheduling problems have been solved or shown to be tractable, such as the two-machine flow shop problem. Yet, the only way to find the optimal schedules of other unsolved scheduling problems is by implicit enumeration. As a result, the trend of multiple-machine scheduling research is to investigate its special cases which may be easier to solve by relaxing some of the constraints of the general scheduling problem.

Some of the commonly used constraint relaxations include allowing preemption of jobs, requiring jobs to have a common processing time requirement or relaxing some of the precedence constraints of jobs (Błazewicz et al., 1988). In a similar vein, parallel-machine scheduling problems can be viewed as a class of problems which is relaxed from the multiple-machine scheduling problems. As research on parallel-machine scheduling problem progresses, a better understanding of multiple-machine scheduling problems may be gained.

The milestone of the development of scheduling theory is the emergence of the concept of NP-completeness. Through this concept, an intractable scheduling problem can be shown to be too hard to solve. It means that a polynomial time algorithm for this scheduling problem is very unlikely to exist. This problem belongs to the class of NP-complete problems. From then, the trend of scheduling research is either to find a polynomial algorithm for a tractable problem, or to show that the problem is intractable. In the latter case, attention will be directed toward the design of an approximation algorithm to search for the sub-optimal schedule or the design of an exponential time algorithm to search for the optimal schedule.

Multiple-machine scheduling systems have two possible configurations, namely, serial and parallel. Serial configuration applies to flow-line type of production systems. The machines in series can be either identical machines (uniform machines with identical speed, i.e. essentially the same kind of machines which are used for the same purposes) or different-purpose machines. The advantage of using a series of identical machines is that it allows the machines to function as one aggregated unit. Thus the productivity of this series is increased. However, other than to increase the system reliability, identical machines in series will reduce the flexibility of the system. An arrangement of different-purpose machines in series is comparable to a continuous flow production line as used in processing industries or industries with relatively large production runs; it trims the production flexibility of the system. A parallel-machine production system is one in which a job can be processed by any one of the free machines. Each finished job will free a machine and leave the system. Note that in this paper, by a parallel-machine system we mean identical machines working in parallel, unless stated otherwise.

2. Notation and representation

We shall use the following primary notation throughout this review, but additional notation will be introduced when necessary:

J : set of n jobs, $J_1, J_2, \ldots, J_i, \ldots, J_n$,
M : set of m machines, $M_1, M_2, \ldots, M_k, \ldots, M_m$,
t_{ik} : processing time requirement for J_i on M_k,
$t_{\max}$: the largest processing time of the set of jobs,
$t_{\min}$: the smallest processing time of the set of jobs,
d_i : due-date requirement for J_i,
r_i : ready time or available time for J_i,
C_i : completion time of J_i,
F_i : flow time of J_i, equal to $C_i - r_i$,
L_i : lateness of J_i, equal to $C_i - d_i$,
E_i : earliness of J_i, $\max\{0, -L_i\}$,
T_i : tardiness of J_i, $\max\{0, L_i\}$,
$\lceil x \rceil$: least integer greater than x,

$\lfloor x \rfloor$: largest integer less than x,
$|x|$: absolute value of x,
$E[Y]$: expected value of the random variable Y.

A commonly used representation of scheduling problems has the form of $n/m/A/B$, which consists of four parameters. The first parameter, n, represents the size of the job set J. The second parameter, m, is the number of machines in the system, with $m = 1$ for a single machine and $m > 1$ for a multiple-machine system. The third parameter, A, is a system parameter which gives the system information about job characteristics and system configuration/machine layout for which the notation is as follows:

Job characteristics

Job processing time

(a) Unit processing time, also called unit job, $t_{ik} = 1$ for all i and k. For $t_{ik} = t$ for all i and k, where t is a constant, we assume $t_{ik} = 1$. This can be done by normalization.

(b) Various processing times, $t_{ik} \neq 1$ for some i and k.

Due-date requirement

(a) Without due-date requirement.

(b) Common due-date, $d_i = d$ for all i, where d is a constant.

(c) General due-date,

$d_i \neq d_k$ for some i and k.

Preemptive jobs

(a) Preemptive jobs may leave the machines unfinished and return later for further processing.

(b) Non-preemptive jobs may not leave the machines unless finished.

Precedence constraints

(a) Independent: no precedence constraint exists between any two jobs, which means that the set of partial order of jobs is empty.

(b) Dependent: precedence constraints exist between some pairs of jobs. We also assume transitive partial ordering relation, which is, if J_a precedes J_b and if J_b precedes J_c, then J_a must precede J_c.

(c) Tree precedence relation: each job has at most one immediate successor or predecessor.

In-tree : all jobs point to one ending node, for example, a production line with one final product.

Out-tree : all jobs start from one node, the root, and keep on branching, for example, a binary tree.

Chain : a series of jobs are connected as the intersections of several subsets of in-tree and out-tree jobs.

Forest : a combination of several sets of in-tree and out-tree jobs.

Arbitrary : precedence relations may occur, but they will not show a simple predictable pattern.

Job ready time

(a) Same ready times for all jobs, $r_i = r$ for all i, where r is a constant.

(b) Arbitrary ready times,

$r_i \neq r_j$ for some i and j.

Multiple-machine systems – System configuration / machine layout

(a) Identical serial machines.

(b) Different-purpose serial machines.

(c) Identical parallel machines, i.e.

$t_{i1} = t_{i2} = \cdots = t_{ik} = \cdots = t_{im} = t_i$

for $i = 1, 2, \ldots, n$.

(d) Uniform parallel machines, i.e. let s_k be the speed factor of the k-th machine; then, for different machines processing the same job with different speeds, $t_{ik} = t_i s_k$ for $i = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, m$.

(e) Unrelated parallel machines, i.e. no particular relation between the values of t_{ik} for $i = 1, 2, \ldots, n$ and $k = 1, 2, \ldots, m$.

(f) Shop scheduling problems: Flow shop: continuous manufacturing layout, all jobs follow the same routing of machines. Generally, production is set at a given rate.

Job Shop: functional manufacturing layout, no fixed routing of jobs. The jobs pass through the functional departments in batches and each batch may have a different routing.

Open shop: jobs can enter the system to any machine and exit from any machine.

If no parameter is given, it implies that the system has the following characteristics: no preemption is allowed, no precedence constraint exists between any pair of jobs, the number of machines is arbitrary and there is no relaxation on job processing times.

The fourth parameter, B, is the performance criterion. All production systems have a goal or a

set of goals to be accomplished by which the performance of the system can be evaluated. The performance criterion is an index used to select the better schedule for implementing a system when more than one feasible schedule exists.

Since all jobs have processing time requirements, usually the length of schedules is evaluated when we compare two schedules. In addition, some sets of jobs have fixed job weights. These weights can be interpreted as the processing cost, holding cost and early or late penalty cost of each job. More complicated situations arise when the weights are a function of the job's flowtime, lateness or tardiness.

It is not the purpose of this study to provide an exhaustive listing of all possible performance criteria. However, the performance criteria that will be discussed in this review are listed here:

Performance criteria

(a) *Completion time based*

(i) ΣC_i or $\Sigma C_i/n$, total completion time or mean completion time, respectively.

(ii) $\Sigma w_i C_i$, weighted completion time.

(iii) $C_{\max} = \max\{C_1, C_2, \ldots, C_n\}$, the maximum completion time of all jobs in a job-set J, also called the makespan, which means the schedule length.

(b) *Due-date based*

(i) ΣL_i or $\Sigma L_i/n$, total job lateness or mean lateness, respectively.

(ii) $\Sigma w_i L_i$, weighted lateness.

(iii) $L_{\max} = \max\{L_1, L_2, \ldots, L_n\}$, the maximum lateness of all jobs in a job-set J.

(iv) ΣT_i or $\Sigma T_i/n$, total job tardiness or mean tardiness, respectively.

(v) $\Sigma w_i T_i$, weighted tardiness.

(vi) $T_{\max} = \max\{T_1, T_2, \ldots, T_n\}$, the maximum tardiness of all jobs in a job-set J.

(c) *Flowtime based*

(i) ΣF_i or $\Sigma F_i/n$, total job flowtime or mean flowtime, respectively.

(ii) $\Sigma w_i F_i$, weighted flowtime.

3. Restrictive assumptions

The following assumptions will be used in this review (French, 1982).

Each job has only one operation.

No job can be processed on more than one machine simultaneously.

Any machine can process any job.

No machine may process more than one job at a time.

Machines will never break down and are available throughout the scheduling period.

Job processing times are independent of the schedule.

Machine setup time is negligible.

Transportation time between machines is negligible.

Work-in-process inventory is allowed and its associated costs are negligible.

Number of jobs is fixed.

Number of machines is fixed.

Processing time of J_i on M_k is given and fixed for all i and k.

Ready time of J_i for all i is known.

4. Solution methods

All deterministic scheduling problems are combinatoric optimization problems. There are four major classes of solution methods for combinatoric optimization problems, namely, complete enumeration, exponential time algorithms, polynomial time algorithms and approximation algorithms. Each class of solution method yields a different level of efficiency and accuracy.

In order to facilitate the study of scheduling problems, the concept of performance ratio is introduced. Performance ratio, or relative performance, is an index which indicates the deviation of the criterion value yielded by an algorithm from the optimal value for the given problem, or, in some cases, from the value for the best known solution.

An instance, I, of the problem is a specification of the variables such as n, m, t_i, w_i, d_i and a set of partial orders of a set of jobs. We denote $A(I)$ as the criterion value of that instance I produced by an algorithm A and OPT(I) the optimal criterion value of that instance I. We then define the performance ratio of instance I, denoted by $R_A(I)$,

as follows:

$$R_A(I) = \frac{A(I)}{\mathrm{OPT}(I)}.$$

Note that, for a minimization problem, $R_A(I) \geqslant 1$ for all A and for all I. Since $R_A(I)$ is only the performance ratio of a particular instance I of the problem, we define the absolute peformance ratio, R_A, for all instances of the problem as

$$R_A \equiv \inf\{r \geqslant 1:\ R_A(I) \leqslant r \text{ for all } I\}.$$

The absolute performance ratio represents the maximum discrepancy between the criterion value produced by algorithm A and the optimal solution. We define the asymptotic performance ratio, R_A^∞, as the asymptotic discrepancy of the criterion value yielded by an algorithm A to the optimal solution when very large values of n and m are used. That is,

$$R_A^\infty \equiv \inf\{r \geqslant 1: \text{ for some } N > 0,\ R_A(I) \leqslant r \text{ for all } I \text{ with } \mathrm{OPT}(I) \geqslant N\}.$$

We define the performance ratio of the worst case of a problem, denoted as R, as the larger of the absolute performance ratio and the asymptotic performance ratio. In terms of worst case analysis, R represents the bound of the maximum discrepancy, both absolutely and asymptotically.

Rather than looking at the performance ratio for one algorithm at a time, Sahni (1976, 1977) introduced an ε-approximation scheme to compare a set of related algorithms. Let $\varepsilon \geqslant 1$ be a constant, F_A the criterion value of the solution yielded by an algorithm A, and F^* the criterion value of the optimal solution. We then require that

$$|(F^* - F_A)/F^*| \leqslant \varepsilon.$$

Thus $F_A(\varepsilon)$ represents the set of ε-approximation schemes of any algorithm A for any given value of $\varepsilon \geqslant 1$.

In addition to the performance ratios of algorithms, the rates of growth of their running time are useful. This rate is an indication of computational complexity. The objective of comparing the rates of growth of the running time is to distinguish algorithms with a higher rate of growth from those with a lower rate of growth. This comparison becomes significant for problems of very large size. It is always desirable to have an algorithm with a low rate of growth of running time.

5. Theory of NP-completeness

The theory of NP-completeness provides a criterion to classify the problems into classes of P or NP (P is the abbreviation of a deterministic polynomial time algorithm and NP denotes a non-deterministic polynomial time algorithm. Completeness means that the algorithm that solves a particular problem will also solve other problems in the same class; in this sense, completeness means the possibility of sharing the same algorithm to solve all problems of a given class).

The fact that a given problem belongs to the class of NP-complete problems means that it is a relative of the 'satisfiability' problem (SAT). Any algorithm that can solve the SAT problem can also solve the given problem, as well as any other problem in the NP-complete class. Cook (1971) has provided a proof that SAT is NP-complete. Karp (1972) has asserted that all NP-complete problems can be 'polynomially-reduced' to SAT. If two problems are in the class of NP-complete problems, they can be transformed into each other in polynomial time. The tool with which this is done is called polynomial transformation.

The NP-complete problems are regarded as hard problems. P problems are relatively easy problems. The development of this theory provides a guideline on what type of result can be expected on what type of scheduling problem. For the study of NP-complete problems, the direction of the study should be focused on developing exponential algorithms and heuristic rules. Many scheduling problems have been shown to be NP-complete (Bruno et al., 1974; Garey and Johnson, 1979; Karp, 1972, 1975a,b; Lenstra and Rinnooy Kan, 1979, 1984a; Lenstra et al., 1977; Papadimitriou and Steiglitz, 1982; Ullman, 1975, 1976).

6. Literature review on parallel-machine scheduling research

In the following we cite over eighty papers drawing from the literature in the disciplines of applied mathematics, computer sciences, industrial engineering, management sciences, oper-

ations research and production management, which are from over forty different journals, books and monographs. This review brings up-to-date parallel-machine scheduling results since publication of McNaughton's paper in 1959.

The class of parallel-machine scheduling problems has been a subject of extensive study by computer scientists for a long time because scheduling incoming jobs on parallel processors presents a major operational problem for running a time-sharing computer system (Coffman, 1976; Horvath et al., 1977; Gonzalez, 1977; Sahni, 1979). The same problem is also encountered in a machine shop where job orders are to be scheduled on groups of identical production facilities. An extensive treatment of the problems of machine scheduling can be found in the works of Conway et al. (1967), Baker (1974), Coffman et al. (1976), Rinnooy Kan (1976), Lenstra (1977), Bellman et al. (1982), Dempster et al. (1982) and French (1982). Gupta and Kyparisis (1987) have provided the most recent survey paper on single-machine scheduling research. Past survey papers on parallel-machine scheduling can be found in Graham et al. (1979), Lawler et al. (1982), and Lenstra and Rinnooy Kan (1984a,b).

The first paper on parallel-machine scheduling is by McNaughton (1959). He introduced three performance criteria for parallel-identical-machine scheduling. For a set of preemptive jobs, the minimum makespan is given by $\max\{t_1, t_2, \ldots, t_n, \sum_{1 \leqslant i \leqslant n} t_i/m\}$ which is one type of completion time based (CTB) performance measures. This quantity is called the McNaughton lower bound. The available time of each machine is initially set as the bound McNaughton's algorithm arbitrarily picks a job and processes it on the lowest index unfilled machine until all jobs are assigned. This method is called the wrap-around algorithm. Its rate of growth of the running time is $O(n)$ and it produces a maximum of $m-1$ job preemptions.

Another performance criterion is total weighted tardiness, which is one type of due-bate based (DDB) performance measures. McNaughton has not developed an algorithm for this problem. However, if all due-dates are equal to zero, the problem becomes the weighted flowtime problem, which is a flowtime based (FTB) performance measure. He also proves that for performance criteria with linear weighted costs, there exists an optimal schedule without any job preemption. It implies that for the weighted flowtime problem, the optimal schedule for non-preemptive jobs is identical to that for preemptive jobs.

In the following, we will present the major results of these three performance criteria.

7. Parallel-machine completion time based performance measures

7.1. Optimization algorithms

Hu (1961) introduced a label scheduling algorithm, also called the critical path method, to solve a set of non-preemptive in-tree unit jobs, in $O(n)$ time. Hu's algorithm first labels the level of the terminal job as zero which implies that the terminal job should be the last job to be processed. Then, according to the set of partial orders of other jobs, these jobs are labelled with the highest level that they may have. Highest level jobs have the highest priority to be processed on the first available machine. The level of a job is defined as the length of the longest path from that job to the terminal job. In other words, this algorithm identifies the critical path first, then schedules the jobs accordingly. Hu's approach is called "highest level first". The original proof of optimality of the algorithm as given by Hu is somewhat lengthy. A simpler proof was subsequently presented by McHugh (1984). A set of preemptive jobs of various processing time requirements can be partitioned into a number of unit sub-jobs with an appropriate precedence relation. Thus, Hu's algorithm is applicable for sets of unequal jobs. This approach will nevertheless increase the problem size exponentially (Gonzalez and Johnson, 1980).

Chen and Liu (1975) generalized Hu's algorithm to deal with non-preemptive partial order unit jobs on an arbitrary number of machines. Their algorithm, called simple level algorithm, requires the partial order of the jobs to satisfy the condition that the number of independent jobs in the sets of immediate successors is always less than the number of machines; it yields an optimal schedule in $O(n)$ time.

The basic idea of Hu's algorithm is first to construct a priority list for the set of jobs according to the 'highest level first' rule. Then jobs to be processed on the first available machine are ordered according to this priority list. This approach to scheduling is called list scheduling. We

will show that this approach has wide applications in relation to the design of optimization and approximation algorithms.

Fujii et al. (1969, 1971) investigated the problem of a set of unit jobs with arbitrary precedence relations to be processed on two machines. Their algorithm orders a set of disjoint compatible job pairs by a matching technique. Thus, the number of pairwise disjoint compatible job pairs is maximized. Any pair of jobs i and j is defined as compatible if both jobs are free from any precedence relation so that they can be processed on two different machines simultaneously. The algorithm runs in $O(n^3)$ time and was subsequently reduced to $O(n^{5/2})$ by Kariv (1976).

Coffman and Graham (1972) applied the highest level first algorithm for arbitrary precedence relation problems for a set of non-preemptive unit jobs. The algorithm breaks ties with jobs in the same level in a way entirely different from Hu's algorithm. Rather, if a job contains a lower lexicographical order of the labels of the immediate successors, it has a higher processing priority. For the two-machine case, the algorithm produces an optimal makespan in $O(n^2)$ time. Sethi (1976b) improved the labelling process which runs in $O(e + n\alpha(n))$ where e is the number of precedence relations and $\alpha(n)$ is an inverse of Ackermann's function which has a very slow growth rate of a constant function of n (Tarjan, 1975). Garey and Johnson (1976) developed another algorithm to examine whether there exists a feasible schedule for two processors for a set of arbitrary precedence relation unit jobs with general due-date requirements. Their algorithm is also applicable for finding the minimum makespan when all job due-dates are zero. The approach of their algorithm is to combine the set of partial orders of jobs and the modified due-dates of each job. This algorithm runs in $O(n^2)$ time.

Gabow (1982), based on the idea of highest level first scheduling, developed a new algorithm for a set of partial order jobs for the two-machine problem. He considered the scheduling problem as a directed acyclic graph (*dag*) with jobs represented by nodes and precedence relations by edges. Hence, a *dag* can be partitioned into levels. The algorithm runs in $O(e + n\alpha(n))$ time where e is the number of edges of the set of jobs. The algorithm first identifies the critical path; it then moves the job on non-critical paths to higher or lower sequence of processing so that the machine idle time can be reduced. He shows that the algorithm is not optimal for any fixed number of machines which is larger than or equal to three, but the upper bound performance ratio of this algorithm is less than or equal to $2 - 2/m$.

Davida and Linton (1976) showed that the highest level first algorithm developed by Hu is equally applicable to out-tree scheduling problems with sets of non-preemptive unit jobs. Hence, Hu's algorithm is found to be applicable to produce optimal schedules for both in-tree and out-tree scheduling problems.

From a theoretical point of view, preemptive scheduling has an advantage over non-preemptive scheduling in that it reduces the complexity of finding an optimal schedule. French (1982) pointed out that scheduling with preemptions allows the scheduler the flexibility of choosing any job (or a portion of it) and inserting it into the machine with idle time between two scheduled jobs. He argued that this flexibility in processing jobs greatly reduces the complexity of the process of constructing a solution to the scheduling problem. Another advantage is that the decision variables of a linear programming formulation of a preemptive scheduling problem can have a value between zero and one instead of being only zero–one integers, hence reducing the computational requirements. This is due to the fact that in preemptive scheduling a fraction of a job can be processed on one machine and then preempted to another machine for further processing (Lawler and Labetoulle, 1978).

In addition, from the practical point of view, the makespan of a preemptive job schedule should be less than that of a non-preemptive jobs schedule. An example is the case of three jobs, all sharing a common processing time, which are to be processed on two machines. If no job preemption is allowed, the makespan is equal to two. However, if job preemption is allowed, the makespan of the schedule is 1.5. It seems that the preemptive schedule will generate a better makespan (Muntz and Coffman, 1969; Sethi, 1976a,b). Nevertheless, if preemption is allowed, more memory is required to transfer between the main memory and auxiliary storage in computer applications. In manufacturing applications, preemptions will incur a cost for transferring preempted jobs.

Any set of non-preemptive jobs can be viewed as a special case of a set of preemptive partial order jobs. Hence, if a non-preemptive scheduling problem is shown to be NP-complete, then, by restriction, the corresponding preemptive scheduling problem is also NP-complete (Garey and Johnson, 1979; Ullman, 1975). Along this line, Muntz and Coffman (1969) applied the level algorithm to solve the two-machine preemptive scheduling problem. A restriction is that it must be possible to partition the set of jobs into several subsets so that all jobs within the same subset are mutually independent. In other words, the precedence relations of jobs, if at all, can exist only between two jobs from different subsets. Thus the precedence relations of the set of jobs form a bipartite graph. The algorithm assigns the subsets of jobs level by level with higher level job first. The algorithm runs in $O(n^2)$ time and yields mn job preemptions. For cases with three or more machines, the algorithm nevertheless gives good schedules but does not guarantee optimality.

Another algorithm called "fast-schedule-by-weight" was introduced by Gonzalez and Johnson (1980) to compare with Muntz and Coffman's algorithm for the same two-machine preemptive scheduling problem. The new algorithm runs in $O(n \log m)$ time with no more than $n-2$ preemptions. However, the algorithm is only applicable to forest precedence relations. A slightly different version of the algorithm called critical-weight algorithm runs in $O(nm)$ time and produces $2nm-2n-m+2$ job preemptions for two or more machines (note that for the two-machine case, the number of job preemptions is equal to one).

Not all minimum makespan problems have a polynomial time algorithm. Thus, some problems must rely on exponential time algorithms to find the optimal schedules. A mixed integer programming formulation of the minimum makespan problem is given by Baker (1974). A general dynamic programming scheme has wide applications for most scheduling problems (Rothkopf, 1966;

Table 1
NP-completeness results of parallel-machine minimum makespan scheduling problems

Preemptive independent jobs
McNaughton (1959), $n/m/\text{ptmn}/C_{\max}$, $O(n)$, P.

Preemptive partial order jobs
Muntz and Coffman (1969), $n/2/\text{ptmn}$, prec, $t_i=1/C_{\max}$, $O(n^2)$, P.
Gonzalez and Johnson, (1980), $n/m/\text{ptmn}$, $\text{prec}/C_{\max}$, two possible algorithms.
Algorithm I: $O(n \log m)$ with $n-2$ preemptions, P.
Algorithm II: $O(nm)$ with $2nm-2n-m+2$ preemptions, P.
Ullman (1975), $n/m/\text{ptmn}$, $\text{prec}/C_{\max}$; $n/m/\text{ptmn}$, prec, $t_i=1, 2/C_{\max}$;
$n/m/\text{ptmn}$, prec, $t_i=1/C_{\max}$, all instances are NP.
Open problem, $n/m \geqslant 3/\text{ptmn}$, $\text{prec}/C_{\max}$.

Non-preemptive independent jobs
Ullman (1975), $n/m/t_i=1/C_{\max}/n/\text{fixed}$ $m/t_i=1/C_{\max}$, all instances are NPC.
Lenstra and Rinnooy Kan (1980), n/fixed $m/C_{\max}$, NPC as in m-partition.

Non-preemptive partial order jobs
Hu (1961), $n/m/\text{in-tree}$, $t_i=1/C_{\max}$, $O(n)$, P.
Chen and Liu (1975), $n/m/\text{restricted order}$, $t_i=1/C_{\max}$, $O(n)$, P.
Davida and Linton (1976), Bruno (1982), $n/m/\text{out-tree}$, $t_i=1/C_{\max}$, $O(n)$, P.
The following papers deal with the same problem which is $n/2/t_i=1/C_{\max}$, P.
(Assuming the problem is either transitively closed or reduced *dag*.)
Fujii et al. (1969, 1971), $O(n^{2.5})$.
Coffman and Graham (1972), $O(e+n\alpha(n))$.
Garey and Johnson (1976), $O(n^2)$.
(Assuming the problem is non-transitive closure of *dag*.)
Gabow (1982), $O(e+n\alpha(n))$.
Ullman (1975), arb t_i, arb m; $t_i=1$ or 2, arb m; $t_i=1$, arb m; all instances are NPC.
Ullman (1975), $B=\frac{1}{2}[\Sigma_{1\leqslant i\leqslant n} t_i]$, $n=Bm$, NPC.
Open problem, fixed $m \geqslant 3$.

Lawler and Moore, 1969). The scheme for the minimum makespan problem is as follows (Graham et al., 1979): Let $C_i(\tau_k)$ be the minimum cost of a schedule without idle time for $J_1, \ldots, J_n$, subject to the constraint that the last job on the k-th machine is completed at time τ_k for $k = 1, \ldots, m$. Then the recursion relation is given by

$$C^*_{\max}(\tau_1, \ldots, \tau_m) = \min_{1 \leqslant k \leqslant m} \{\max(C_i(\tau_k), C_{i-1}(\tau_1, \ldots, \tau_k - t_i, \ldots, \tau_m))\}$$

with the initial condition

$$C(\tau_1, \ldots, \tau_m) = \begin{cases} 0 & \text{if } \tau_k = 0 \text{ for } k = 1, \ldots, m, \\ \infty & \text{otherwise.} \end{cases}$$

The above procedure solves the minimum makespan problem in $O[n(C^*_{\max})^m]$, where $C^*_{\max}$ denotes the optimal makespan, with the maximum number of n^m intermediate solutions which require a large amount of memory storage. For large values of $C^*_{\max}$, this optimization method requires prohibitive times in the rare case. A similar dynamic programming scheme is given in Lenstra and Rinnooy Kan (1979).

Another dynamic programming scheme with a limited number, k, of different processing times is given by Leung (1982). The algorithm is implemented in $O(n^{2(k-1)} \log(t_{\max} + m))$ time and requires $O(\log mn^{k-1})$ memory space.

Table 1 shows the known NP-complete (in short NPC) cases of the minimum makespan problem.

7.2. Approximation algorithms

The scheduling problem to find the minimum makespan of a set of independent preemptive jobs has been well-solved by the wrap-around algorithm due to McNaughton (1959). However, other types of scheduling problems for finding the minimum makespan, namely sets of dependent preemptive jobs, preemptive and non-preemptive partial order jobs, remain unsolved. Some of these problems have been shown to belong to the class of NP-complete problems. Hence, the use of approximation algorithms is justified.

There are essentially two approaches to the design of approximations, namely, list scheduling and pin packing, the discussion of each of which is to follow.

7.2.1. List scheduling – (i) Worst case analysis of performance bounds

List scheduling has been the subject of extensive research and involves three dominated algorithms:

(1) random list scheduling,
(2) highest level first, and
(3) longest processing time.

Graham (1966, 1969) introduced the list scheduling algorithm in light of Hu's algorithm. The naive method of list scheduling is to order the set of jobs randomly. A more systematic method is to order jobs in ascending or descending processing times. The highest level first algorithm due to Hu is a type of list scheduling. We will leave the discussion of the highest level first algorithm for later.

All three list scheduling algorithms share a similar pattern of job dispatching: A priority list for the set of jobs is constructed. Jobs on the top of the list are picked and loaded on the first available machine. The loading procedure continues until the list is exhausted. The idea of picking a job and loading it to the first available machine is based on the intuitive principle of making the loads on machines as even as possible (De and Morton, 1980). The implementation of these algorithms is very easy. However, these algorithms work very well especially with a fixed number of machines. Indeed, there are more than one hundred dispatching rules to construct the priority list. We refer the reader to the paper of Panwalkar and Islander (1977) for an extensive list of dispatching rules.

7.2.1.1. Minimum makespan problems – independent non-preemptive jobs. Graham showed that for any arbitrary priority list the bound performance ratio is $R_{\text{RDM}} \leqslant 2 - 1/m$. For single machine problems, this bound is one and agrees with intuition. As the number of machines increases, the bound approaches the value two asymptotically (it means that the performance ratio in the worst case is still less than two). Graham et al. (1979) asserted that, if the random list scheduling algorithm is used, a pairwise interchange (PI) of any two jobs can be used. This process repeats until the objective function stops improving. However, the bound is still $R_{\text{PI}} \leqslant 2 - 2/(m+1)$ which is not much of an improvement for even a small value of m.

Achugbue and Chin (1981) gave a detailed account of all cases of random list scheduling problems. They showed that the behaviour of bounds on list scheduling varies, by narrowing

down the ratio ρ, defined as t_{max}/t_{min}, which represents the degree of similarity between the processing times of all jobs (hence the job processing times may be considered similar but not necessarily identical). For $\rho \leqslant 2$, the bound is $\frac{5}{3} - \frac{1}{3}\lfloor \frac{1}{2}m \rfloor$ for $m \geqslant 4$ and $\frac{3}{2}$ for $m = 2$ or 3. For $\rho \leqslant 3$, the bound of the performance ratio R_{RDM} is $2 - \frac{1}{3}\lfloor \frac{1}{3}m \rfloor$ for $m \geqslant 6$, $\frac{17}{10}$ for $m = 5$ and $\frac{5}{3}$ for $m = 3$ or 4. For any real value of ρ the problem remains unsolved if $1 \leqslant \rho \leqslant 4$. This is probably due to the discrete nature of the problems which makes it difficult to solve them.

Intuitively, the way to construct the priority list of a list scheduling algorithm should be systematic rather than random. Indeed, the rationale of the design of approximation algorithms for minimum makespan problems should be to avoid lumpiness in the loads near the end, in other words, to avoid late processing of excessively long jobs (De and Morton, 1980). Graham (1969) proved that for the priority list of list scheduling constructed according to descending order of processing times (LPT), the bound of the performance ratio is $R_{LPT} \leqslant \frac{4}{3} - \frac{1}{3}m$ in the worst case. The bound is also tight.

However, for RDM and LPT list scheduling, the maximum desirable error level is uncontrollable. Graham (1969) designed an algorithm to trade-off between the error level of the algorithm and computational complexity. For the two-machine case, the first k jobs are picked and loaded to the two machine optimally. Then the rest of the $n - k$ jobs will be loaded at random. This k-algorithm yields a bound on the performance ratio

$$R_k \leqslant 1 + 1/\left(2\left\lfloor 1 + \tfrac{1}{2}k \right\rfloor\right).$$

For any fixed number of machines, the bound is

$$R_k \leqslant 1 + (1 - 1/m)/(1 + \lceil k/m \rceil).$$

As observed, the lower value of the bound occurs if the value of k is an m-multiple. Thus, for k equal to zero, all jobs will be scheduled at random, and the bound is no greater than $2 - 1/m$. For k equal to two times the number of machines, the bound is no greater than $\frac{4}{3} - \frac{1}{3}m$, which is as good as LPT. For k equal to three times the number of machines, the bound is no greater than $\frac{5}{4} - \frac{1}{4}m$. For k equal to mod(m), with k tending to infinity, the bound is one and loose.

This technique of first constructing a partial set of jobs optimally and then ordering the rest of jobs randomly, introduced by Graham (1969), was further elaborated by Johnson (1972). Sahni (1976, 1977) found that the k-algorithm is not truly running in polynomial time. As the value of k increases, the rate of growth of the running time of the k algorithm increases exponentially. He introduced an algorithm called rounded dynamic programming (RDYN). Thus the running time of RDYN is no longer exponentially growing with an increase in the value of k but depends on the level of desirable accuracy ε. For each value of ε, a different algorithm is generated. Hence, Sahni defined this series of algorithms as a fully polynomial scheme. The bound of the performance ratio is

$$R_{RDYN} \leqslant 1 + 2nk/\sum t_i \quad \text{with } k = \varepsilon \sum t_i/2n.$$

Ibarra and Kim (1976) provided a tighter bound of performance ratio for LPT which is $R_{LPT} \leqslant 1 + (2m - 1)/n$ if $n \geqslant 2\rho(m - 1)$. However, this bound of the LPT algorithm does not perform very well for the two-machine case with two unit jobs. The bound is $\frac{5}{2}$ whereas Graham's bound is $\frac{7}{6}$, and the optimal solution is one. Asymptotically, the boundary value of Ibarra and Kim's algorithm and that of Graham's are similar. However, a close look at the Ibarra and Kim's performance bound reveals that

$$R_{LPT} \leqslant 1 + \frac{2m - 1}{\rho(2m - 2)}.$$

As the values of m and n tend to infinity, the bound becomes

$$R_{LPT} \leqslant 1 + 1/\rho.$$

The worst case occurs when $t_{min} = t_{max}$ (i.e. $\rho = 1$). Then $t_i = t$ for all i, and the worst case bound is two. For Graham's LPT the asymptotic performance ratio is $\frac{4}{3}$. Thus we conclude that Ibarra and Kim's bound is not superior to Graham's bound asymptotically.

Dobson (1984) proved that if the set of jobs is restricted to having relatively small processing times, the bound using the LPT algorithm was still $\frac{4}{3}$ which is no improvement over what Graham (1976) showed. Hence, he proposed a general expression for the range of the processing times. For $t_i \in [0, (1/k)C^*_{max}]$ for some k, the LPT algorithm produces the bound in the range $R \leqslant (k + 3)/(k + 2)$. For k equal to one, the bound is $\frac{4}{3}$ and

for k equal to two, the bound is $\frac{5}{4}$. Kao and Elsayed (1988) proved that if $t_i \leqslant mt_{i-1}/(m-1)$ for $i = 2, 3, \ldots, n$, the performance bound to LPT is $1 + (m-1)/n$.

7.2.1.2. Minimum makespan problems – Preemptive partial order jobs. Muntz and Coffman (1969) applied highest level first algorithm on a set of preemptive partial order jobs. The set of jobs is restricted so that it can be partitioned into several subsets of jobs. All jobs in the same subset must be mutually independent. The rationale of this algorithm is that each job is partitioned into several sub-jobs of identical processing time requirement. Precedence relations of these sub-jobs are added to ensure that each partition is feasible. Thus the set of jobs is converted to a set of sub-unit-jobs. The bound of performance ratio is $R_{MC} \leqslant 2 - 2/m$ for two or more machines. The algorithm produces optimal schedules for the two-machine case. In addition, the asymptotic performance ratio is two (Lam and Sethi, 1977).

7.2.1.3. Minimum makespan problems – Non-preemptive partial order jobs. The result of the highest level first algorithm due to Muntz and Coffman (1969) can be applied to a set of non-preemptive partial order unit jobs. The bound of performance ratio is $R_{MC} \leqslant 2 - 2/m$ for the $m \geqslant 2$ machine cases. Furthermore, for the two-machine case, the algorithm yields an optimal schedule.

Coffman and Graham (1972) applied the highest level first algorithm to solve a set of non-preemptive partial order jobs. The algorithm breaks ties between jobs of the same level by counting the lexicographical order of the labels of their immediate successors which is different from Muntz and Coffman's highest level first algorithm. Coincidentally, the bound is also $R_{CG} < 2 - 2/m$, which is identical to the Muntz–Coffman's bound (Lam and Sethi, 1977). Goyal (1977) proposed a generalized version of the Coffman–Graham algorithm for the two-machine system with non-preemptive precedence relation jobs. If the range of processing time is between one and k, then the bound can be improved to $R_{G\{1,k\}} \leqslant \frac{3}{2} - 1/(2k)$ for $k = 3$. This bound is in agreement with that of Coffman and Graham (1972).

Kaufman (1974) applied the highest level first algorithm to solve a set of non-preemptive in-tree jobs. His algorithm breaks ties between jobs at the same level arbitrarily, which produces the bound $R_{KF} \leqslant 1 + (m-1)t_{max}/\Sigma t_i$. He proved that the difference between R_{KF} and the worst case bound of performance ratio of a set of preemptive partial order jobs is the value of $t_{max}(1 - 1/m)$. If this is a set of unit jobs, the value will become $1 - 1/m$. This value agrees with the observation of the single-machine case, i.e. the optimal schedule is identical whether job preemption is allowed or not. Chen and Liu (1975) applied the simple level algorithm to a set of arbitrary partial order unit jobs. Their algorithm is similar to the highest level first algorithm except that the assignment of priorities to jobs at the same level is completely arbitrary. The bound is $\frac{4}{3}$ for $m = 2$ and $2 - [1/(m-1)]$ for $m \geqslant 3$.

The random list scheduling due to Graham (1966) is applicable for a set of non-preemptive partial order jobs. The bound of the performance ratio is $R_{RDM} \leqslant 2 - 1/m$ for $m \geqslant 2$.

Graham (1976) gave three bounds of the performance ratio for three related sets of non-preemptive partial order jobs scheduling problems by using the LPT algorithm. He proved that the LPT algorithm produces the bound $2 - 1/m$. This bound is improved slightly by restricting the set of jobs to unit jobs; it becomes $2 - 1/(m-1)$ for two or more machines. The LPT algorithm yields an optimal schedule for the two-machine case. Furthermore, for a set of non-preemptive in-tree jobs, the LPT algorithm produces a performance bound $2 + \varepsilon/[(m+1)(1+\varepsilon)]$ for any constant value of $\varepsilon \geqslant 0$ (Graham, 1976). The asymptotic bound approaches two as $\varepsilon \to 0$ and $m \to \infty$. In conclusion, Graham showed that, although the scheduling problem to find the sub-optimal schedule for a set of non-preemptive partial order jobs was restricted to either a set of unit jobs or a set of in-tree jobs, the asymptotic bound of the performance ratio using the LPT algorithm is still two, which is not much improvement over the unrestricted problem.

Kunde (1981) gave a full account of the LPT algorithm for sets of in-tree, out-tree and chain jobs. For the set of in-tree jobs, the performance bound is $2 - 2/(m-1)$, which was given by Graham (1976). For the set of out-tree jobs, no single bound was given but Kunde proved that the bound is worse than the bound of a set of in-tree jobs, $2 - 2/(m-1)$, but converging to 2 as the

number of machines increases. However, if $m = 2^k - 1$, with k an integer and $k \geqslant 2$, the bound of the set of out-tree jobs is $R_{\text{out}} \geqslant 2 - 2(m+1)$. Scheduling problems with a set of chain jobs can be viewed as the general case of opposing forests (Garey et al., 1983). The bound for this problem has two results since there are different numbers of machines. If the number of machines is 2, 3 or 4, the bound is equal to $1 + (m-1)/(2m-1)$. If $m \geqslant 5$, let $A_m = \{\lfloor x_m \rfloor, \lceil x_m \rceil\}$, where

$$x_m = \left(m + \frac{1}{m}\right)\{\sqrt{1+2m} - 1\};$$

then

$$R_{\text{chain}} = 1 + \max_{k \in A_m}\left\{\frac{(m-2k+1)(2k-1)}{(m-2k+1)(3k-1)+k^2}\right\}.$$

The asymptotic bounds on the performance ratio are: R(independent LPT) $= \frac{4}{3}$, R(in-tree LPT) $= 2$, R(out-tree LPT lower bound) $= 2$ and R(chain LPT) $= \frac{5}{3}$.

This concludes our discussion of the worst case analysis of approximation algorithms.

7.2.2. List scheduling – (ii) Probability analysis of performance bounds

Instead of only considering the worst case analysis of approximation algorithms to solve the minimum makespan problem, Coffman et al. (1982a) considered a probabilistic analysis of the list scheduling algorithm.

7.2.2.1. Minimum makespan problems – Independent non-preemptive jobs. Throughout the research done by Coffman et al. (1982, 1984a) and Coffman and Gilbert (1985), Loulou (1984) and Bruno and Downey (1986), all of their discussions assume that the processing times of the set of jobs are independent and identically distributed random variables with a uniform distribution. The exception is in the second part of Coffman and Gilbert's paper (1985) which considered the processing times of the set of jobs to be exponentially distributed random variables.

Coffman et al. (1982) considered the bound of the LPT algorithm for $t_i \in (0, 1]$. The performance bound is no greater than $1 + O(m^2/n^2)$. In a subsequent paper, Coffman et al. (1984a) introduced the restricted longest first (RLF) rule in which jobs in the set were assigned in pairs, one job to one processor and the other job to another processor. The job with the longer processing time of the pair is the first job to be scheduled. If the number of jobs of the set is odd, the last unassigned job is then assigned to the first available machine. The RLF rule yields $E[R_{\text{RLF}}] = \frac{1}{4}n + \frac{1}{2}(n+1)$. Thus we have

$$E[R_{\text{RLF}}]/E[\text{OPT}] \leqslant 1 + 2/n(n+1),$$

since it is obvious that

$$C^*_{\max} \geqslant \tfrac{1}{2} \sum_{1 \leqslant i \leqslant n} E[t_i] = \tfrac{1}{4}n.$$

For the LPT algorithm, the upper and lower bounds of $E[C_{\max,\text{LPT}}]$ are equal to $\frac{1}{4}n + \frac{1}{4}(n+1)$ and $\frac{1}{4}n + \frac{1}{2}\varepsilon(n+1)$, respectively. Thus, we have

$$1 + \frac{1}{n(n+1)} \leqslant \frac{E[C_{\max,\text{LPT}}]}{E[C^*_{\max}]} \leqslant 1 + \frac{2\varepsilon}{n(n+1)}.$$

When we compare the running times of LPT and RLF, both of them have a rate of growth $O(1/n^2)$. However, when we compare their bounds, LPT has $1 + O(1/m)$ and RLF has $1 + O(1/n)$. Nevertheless, this paper is limited to the two-machine case. For an arbitrary number of machines, the problem is still open.

In the same paper, the performance of random list (RDM) scheduling was given. The bound is $E[C_{\max,\text{LPT}}]/E[C^*_{\max}] \leqslant 1 + \frac{2}{3}n$ for any number of machines. It happens that RDM performs very well if the size of the set of jobs is small. Loulou (1984) proved the results of RDM and LPT that the absolute errors of $R_{\text{RDM}} - R_{\text{OPT}}$ and $R_{\text{LPT}} - R_{\text{OPT}}$ converge in probability to a finite limit.

Using statistical methods, Bruno and Downey (1986) conducted a probabilistic analysis of list scheduling. Two bounds are given. The first one is

$$\Pr\left\{\frac{C_{\max}}{C^*_{\max}} < \frac{4(m-1)}{n}\right\} > 1 - \varepsilon,$$

where $1 - \varepsilon$ is the degree of desirable confidence, L denotes any list scheduling algorithm and $C_{\max,\text{L}}$ is the makespan produced by L. And the second one is

$$\frac{C_{\max}}{C^*_{\max}} \leqslant 1 + \frac{m-1}{nA},$$

where $t_i \in [A, B]$ and $A, B \in \mathbb{Z}^+$, $1 \leqslant i \leqslant n$, and is obtained by first determining n and then getting $C_{\max,\text{L}}$ close to $C^*_{\max}$.

Coffman and Gilbert (1985) followed the work of Coffman et al. (1982a, 1984a) and developed the probabilistic bound for the RDM algorithm for the case that the processing times were assumed to be uniformly distributed which gave the probabilistic bound as follows:

$$E\left[\frac{C_{\max,\mathrm{RDM}}}{C^*_{\max}}\right] \leqslant 1 + \frac{2(m-1)}{n-2} \quad \text{for } n > 2.$$

Furthermore, if the processing times were assumed to be exponentially distributed, then the probabilistic bound became

$$E\left[\frac{C_{\max,\mathrm{RDM}}}{C^*_{\max}}\right] \leqslant 1 + \frac{(m-1)H_{m-1}}{n-m} \quad \text{for } n > m,$$

where H_i denotes the i-th harmonic number.

Frenk and Rinnoy Kan (1986, 1987) presented similar results for systems of parallel uniform machines, which are also applicable to systems of parallel identical machines. If the expected value of the job processing time is finite, the LPT is 'asymptotically absolutely optimal almost surely'; if the variance of the job processing time is finite, the LPT is 'asymptotically absolutely optimal in expectation'. They concluded, with computational and statistical results, that LPT is a good heuristic asymptotically, from the points of view of both worst and probabilistic case analyses.

7.2.3. List scheduling – (iii) Other approaches

Kim (1987) introduced a backward approach to list scheduling. This approach schedules the last processed job first. Kim randomly generated 30–40 instances to test the performances of the backward and forward approaches in relation to different scheduling heuristic rules for in-tree, out-tree and general precedence relations. Kim concluded that the backward method works better for sets of jobs with in-tree and general precedence relations, and that the forward method was better for set of jobs with out-tree precedence relations.

Table 2 concludes our discussion on list scheduling algorithms, which includes an overview of the highest level first algorithm, for the minimum makespan scheduling problem.

7.2.3.1. Bin packing: Minimum makespan problems – Independent non-preemptive jobs. The parallel-machine scheduling problem can be regarded as the dual of the bin packing problem. The general scheduling problem can be mathematically formulated as a one-dimensional bin packing problem. For a positive capacity C and a set of jobs

Table 2
Approximation algorithms for parallel-machine scheduling problems – List scheduling approach

Minimum makespan problems – Non-preemptive independent jobs
Random:
Graham (1966), $R \leqslant 2-1/m$.
Graham (1969), k-jobs optimally, $(n-k)$-jobs randomly, $R \leqslant 1+(1-1/m)/(1+\lfloor k/m \rfloor)$.
Graham (1979), local interchange, $R_{\mathrm{LI}} \leqslant 2-2/(m+1)$.
Achugbue and Chin (1981), for $\rho \leqslant 3$, $R_{\mathrm{RDM}} \leqslant 2-\frac{1}{3}\lfloor \frac{1}{3}m \rfloor$ for $m \geqslant 6$, $\frac{17}{10}$ for $m=5$, $\frac{5}{3}$ for $m=3$ or 4; for $\rho \leqslant 2$, $R_{\mathrm{RDM}} \leqslant \frac{5}{3}-\frac{1}{3}\lfloor \frac{1}{2}m \rfloor$ for $m \geqslant 4$; $\frac{3}{2}$ for $m=2$ or 3.
Longest processing time:
Graham (1969), unit jobs, $R \leqslant 2-2/m$.
Ibarra and Kim (1977), $R \leqslant 1+\rho(2m-1)/(2m-20)$.
Dobson (1984), $R \leqslant (k+3)/(k+2)$ if $t_i \in [0, (1/k)C^*_{\max}]$.

Minimum makespan problems – Preemptive partial order jobs
Random:
Graham (1966), $R \leqslant 2-1/m$.
Highest level first:
Muntz and Coffman (1969), unit jobs, $R \leqslant 2-2/m$.
Coffman and Graham (1972), unit jobs, $R \leqslant 2-2/m$.
Kaufman (1974), $R \leqslant 1+[(m-1)\max t_i]/\Sigma t_i$.
Chen and Liu (1975), unit jobs, $R \leqslant 2-1/(m-1)$ for $m \geqslant 3$, $R=\frac{4}{3}$ for $m=2$.
Longest processing time:
Graham (1976), $R \leqslant 2-1/m$; unit jobs, $R \leqslant 2-1/(m-1)$; in-tree, $R \leqslant 2-2/(m+1)$.
Kunde (1981), in-tree, $R^\infty = 2$; out-tree, $R^\infty = 2$; chain, $R^\infty = \frac{5}{3}$.

$J = (J_1, J_2, \ldots, J_n)$, each of the jobs has a processing time requirement t_i, with $0 \leqslant t_i \leqslant C$. The objective is to find the minimum value of m such that there is a partition of J into sub-sets of $J_1', J_2', \ldots, J_m'$ satisfying

$$\sum_{J_i \in J_k'} t_i \leqslant C \quad \text{for } k = 1, \ldots, m.$$

Since both the bin packing and parallel-machine problems share the same decision problem, which is whether all jobs can be processed with a fixed number of m and a fixed value of C, it is easy to show that this problem is in the class of NP-complete as documented by Garey and Johnson (1979). The survey papers by Coffman (1982) and Coffman et al. (1984b) give detailed reviews of the bin packing problem.

Krause et al. (1975) are probably the first to apply the results of one-dimensional bin packing problems to parallel-machine scheduling problems. They asserted that the bin packing problem could be restricted to the problem of finding the minimum number of bins (i.e. machines) to pack all objects (i.e. jobs). Thus, they introduced an algorithm named first-fit-increase (FFI). The set of machines is indexed ascendingly. Then, the first-fit-increase algorithm orders the set of jobs in ascending order of processing times, and assigns the job on the top of list to the lowest index machine which may fit that job. The algorithm has a bound of

$$\tfrac{27}{10} - \left\lceil \tfrac{37}{10}/n \right\rceil \leqslant R_{\mathrm{FFI}}^{\infty} \leqslant \tfrac{27}{10} - \tfrac{24}{10}/n$$

as the minimum number of machines to process the set of jobs. The bound approaches $\frac{27}{10}$ when the number of jobs tends to infinity. The first-fit-decrease algorithm (FFD) is identical to the first-fit-increase algorithm, except that it orders the set of jobs in descending order of processing times. The bound of minimum number of machines is $R_{\mathrm{FFD}} = 2 - 2/n$.

Another algorithm is called the iterated-lowest-fit-decrease algorithm (ILFD) which orders the jobs in descending order of processing times, as does FFD, but then picks an obvious lower bound as the number of machines and proceeds the FFD until no job can be assigned. Then the algorithm increases the lower bound by one unit and does the problem all over again. This procedure of adding one machine at a time keeps on repeating and halts when all jobs are assigned. The running time of ILFD is $O(n^2 \log n)$, however if the search for the minimum number of machines to fill all jobs can be implemented with a binary search method, then the running time becomes $O(n \cdot \log^2 n)$ (Coffman et al., 1978). The bound is less than two for an infinite number of jobs.

Nevertheless, the application of bin packing is not limited to parallel-machine scheduling problems. An application of bin packing can be found in computer storage allocation which is another variant of the parallel-machine scheduling problems (Coffman and Leung, 1979).

The best-fit algorithm assigns the job on the top of the list to the machine which has minimum occupied time but which fits the job. The worst-fit algorithm assigns the job to the machine with the largest available machine time. Graham's LPT algorithm can be regarded as lowest-fit-decrease. The result of this algorithm has been analyzed in the discussion of list scheduling. Coffman and Sethi (1976) found that if the ratio of the number of objects (jobs) per bin (machine) was less than or equal to three, the bound of the LPT algorithm is $P_{\mathrm{LPT}} \leqslant (k+1)/k - 1/(km)$, where k is the number of iterations chosen. Langston (1982) further provided an iterative technique to avoid the worst case of LFD. He showed that this technique can yield a bound of $\frac{5}{4} + 1/[12(2^k)]$.

Coffman et al. (1978) introduced a new bin packing algorithm called multi-fit-decrease (MFD). This algorithm is similar to ILFD but is implemented with a binary search method. Multi-fit-decrease restricts the search of the optimal schedule using the McNaughton lower bound. The upper bound is double the value of the lower bound. Then, through the binary search method with up to k iterations, the sub-optimal solution is found. The algorithm runs in $O(n \log n + knm)$ time.

Finn and Horowitz (1979) showed that MFD runs in linear time for a large number of jobs. They initially constructed the closely related problem of minimizing the difference of the maximum and the minimum bin levels. The problem is equivalent to the minimum makespan problem for a two-machine system. The algorithm is executed by iteratively exchanging jobs in two different machines in order to reduce the deviations between the levels of any pair of bins.

Table 3
A list of parallel-machine scheduling papers dealing with completion time based performance measures

Independent CTB performance measures	
Preemptive jobs:	McNaughton (1959)
Non-preemptive jobs:	Graham (1966, 1969) Johnson (1973) Krause et al. (1975) Ullman (1975) Coffman and Sethi (1976) Ibarra and Kim (1976, 1977) Sahni (1976, 1977) Coffman et al. (1978, 1982, 1984a,b) Finn and Horowitz (1979) Graham et al. (1979) Lenstra and Rinnooy Ken (1980) Achugbue and Chin (1981) Karmarkar and Karp (1982) Langston (1982) Friesen and Langston (1983) Friesen (1984) Loulou (1984) Coffman and Gilbert (1985) Bruno and Downey (1986) Hochbaum and Shmoys (1987) Kao and Elsayed (1988)
Dependent CTB performance measures	
Arbitrary:	
Preemptive jobs:	Muntz and Coffman (1969) Ullman (1975) Lam and Sethi (1977) Gonzalez and Johnson (1980)
Non-preemptive jobs:	Graham (1966) Fujii et al. (1969, 1971) Muntz and Coffman (1969) Chen and Liu (1975) Ullman (1975) Garey and Johnson (1976) Graham (1976) Sethi (1976b) Goyal (1977) Lam and Sethi (1977) Garey and Johnson (1979) Gabow (1982) Kim (1987)
In-tree/Out-tree:	
Preemptive jobs:	Nil
Non-preemptive jobs:	Hu (1961) Kaufman (1974) Davida and Linton (1976) Kunde (1981) Bruno (1982) McHugh (1984) Kim (1987)
Chain:	
Preemptive jobs:	Nil
Non-preemptive jobs:	Kunde (1981) Garey et al. (1983)

Friesen (1984) showed that for different numbers of iterations, k, and machines, the performance bound varied. For more than two machines and more than four iterations, the bound of performance ratio of MFD is $R_{\mathrm{MFD}} \leqslant \frac{1}{5}k + \frac{1}{2}^{k}$. For systems of more than 12 machines, the bound is $\frac{13}{11}$. This indeed is the best bound ever known for any bin packing algorithm. Friesen and Langston (1983) provided the bound of MFD of uniform machines.

A new approach for minimizing the difference of maximum and minimum bin levels based on an operation called "set difference" was introduced by Karmarkar and Karp (1982). This algorithm runs in $O(n^{-\log n})$ for an arbitrary number of machines. The algorithm starts with finding the difference of the two largest numbers, repeats this procedure until no comparison can be made. The number which has not been compared is the minimum difference of maximum to minimum bin levels.

Hochbaum and Shmoys (1987) developed an entirely different approach for the bin packing problem. The rationale of their dual-approximation scheme is based upon the concept of duality theory in linear programming. Instead of searching for an optimum solution from the region of feasible but suboptimal solutions, the dual approximation scheme searches for the optimum from the region of infeasible but 'super-optimal' solutions. The bound of this algorithm is measured by the degree of infeasibility allowed. This scheme has two relative error levels, $\varepsilon = \frac{1}{5}$ and $\varepsilon = \frac{1}{6}$, in which the scheme is efficient. The algorithm runs in $O((n/\varepsilon)^{1/\varepsilon})$ time. With $\varepsilon = \frac{1}{5}$ and a maximum number of iterations k, the bound is equal to $\frac{6}{5} + 1/(2k)$ with running time $O(n(k + \log n))$. For $\varepsilon = \frac{1}{6}$, the performance bound is equal to $\frac{7}{6} + 2^{-k}$ with running time $O(n(m^4 + \log n))$.

This concludes our discussion on bin packing algorithms for the minimum makespan problem.

Overall speaking, since the minimum makespan scheduling problem has wide applications in scheduling of computer usage, it is the most intensively studied scheduling problem. We have reviewed this problem in relation to its optimization algorithms, NP-completeness, and approximation algorithms which have two major approaches, list scheduling and bin packing. Further study of minimum makespan scheduling should go in the direction of the advancement of the bin packing algorithm or the dual approximation approach as

proposed by Hochbaum and Shmoys (1987). We conclude the discussion of completion time based performance measures with a list of papers that we have discussed (Table 3).

8. Parallel-machine due-date based performance measures

The single-machine minimum weighted tardiness scheduling problem has been shown to be NP-complete (Graham et al., 1979). Hence, parallel-machine minimum weighted tardiness is obviously an NP-complete problem (Graham et al., 1979). Even if job preemption is allowed, the problem is still NP-complete. Furthermore, if we restrict all weights of jobs to be equal, the weighted tardiness problem is reduced to Σt_i, called tardiness problem. This problem remains NP-complete for the two-machine case. For more than two machines, the problem remains open (Lenstra et al., 1977). In other words, there is no known polynomial time algorithm to solve all weighted tardiness problems, either with single- or parallel-machine, or sets of preemptive or non-preemptive jobs with equal job weights. Any other variation of weight tardiness problem is either in the class of NP-complete or remains open.

The structure of the tardiness problem makes it inapplicable to scheduling problems in computer sciences. However, the study of the tardiness problem remains the main interest in the discipline of industrial engineering and production management. Indeed, a wide variety of applications exists for the tardiness problem in manufacturing systems.

Dogramaci and Surkis (1979) presented a zero–one integer programming formulation of the weighted tardiness problem. Rajaraman (1975, 1977) introduced the use of dynamic programming to find the minimum tardiness penalty, processing cost and waiting cost. The idea of this algorithm is to first allocate m jobs at a time with one job to one machine. Then the rest of the $n - m$ unscheduled jobs are added to the m machines one at a time. The optimal schedule is found after all jobs are scheduled.

A list scheduling approach is applicable to weighted tardiness problems. Dogramaci and Surkis (1979) tested three list scheduling algorithms including shortest processing time (SPT), earliest due-date (EDD) and slack (SLK), which is defined as $d_i - t_i$ for all i. After jobs are loaded on the machines, the jobs are re-sequenced using EDD. All three algorithms are compared with the lower bound yielded by the corresponding zero–one integer program which allows the job assignment indicator variable to take on a non-negative fractional number no greater than one.

Instead of minimizing the weighted tardiness of a set of jobs, Roots (1965) established necessary and sufficient conditions to identify the optimal schedule for minimizing the maximum tardiness with a common due-date. Root's paper is the only published study of the minimum maximum tardiness problem.

For tardiness problems, it is assumed that a job completed before its due-date will not incur any cost. This assumption should be relaxed for systems with a limited capacity of finished job buffer or memory capacity. Hence, it leads to the study of lateness problems in which cost will be incurred if a job is either late or early with respect to its due-date.

There exists virtually no published literature on any attempt to solve the weighted lateness problem. The only exception is Cheng (1989) who used a heuristic approach for a problem in which due-dates, rather than given, are to be assigned and treated as a decision variable. For the maximum lateness problem, several instances have been shown to be NP-complete. Lenstra et al. (1977) have shown that finding the maximum lateness of a set of independent non-preemptive general due-date jobs for a system of two or more machines is an NP-complete problem.

Horn (1974) studied the necessary and sufficient conditions of the existence of a feasible schedule of a set of independent preemptive jobs with a common due-date d. The conditions are: (1) $t_i \leqslant d$, and (2) $\Sigma t_i \leqslant md$. In addition, he argues that the maximum lateness problem with a set of general due-dates d_i can be transformed to $d_i + M$ by a constant M in which no job is late. The problem is to find the minimum value of M.

Instead of looking at the existence of the no tardy job schedule, Brucker et al. (1977) developed an algorithm to solve the maximum lateness problem for a set of in-tree unit jobs. The algorithm is based on the highest level first algorithm. A consistent list is constructed in which the set of partial orders of the set of jobs is included with the

modified due-dates. This avoids the burden of considering the due-date of a job and its precedence relations with other jobs. The priority rule is based on the relative urgencies in the set of jobs which are represented by their modified due-dates. The set of jobs is ordered by their earliest modified due-dates and loaded to the first available machine. The running time of this algorithm is $O(n \log n)$ due to the sorting procedure. This algorithm only works for in-tree but not out-tree problems. Having completed the partial schedule of time $T_1, T_2, \ldots, T_p$, we proceed with the scheduling for time T_p and $T_p + 1$. The number of jobs to be scheduled at time T_p may be less than or equal to $T_p + 1$ for the in-tree problem, and vice versa for the out-tree problem. This algorithm is designed to schedule reducing numbers of jobs as time passes. Thus, the algorithm is limited to in-tree problems. Moreover, Brucker et al. (1977) proved that the out-tree problem is NP-complete. Monma (1982) found an algorithm that runs in $O(n)$ time. He argued that instead of sorting the entire set of jobs according to their modified due-dates, his algorithm sorted only a partial set of jobs with the modified due-dates.

In addition, it is easy to show that a set of in-tree preemptive jobs can be partitioned into in-tree non-preemptive unit jobs. Thus, either the algorithm by Brucker et al. (1977) or Monma (1982) can be applied to find the optimal schedule. This nevertheless increases the complexity of computing the optimal schedule. Instead, Lawler (1982) constructed an algorithm using a different approach. The set of jobs is ordered in ascending order of slack. Having scheduled the jobs at times $\tau_1, \tau_2, \ldots, \tau_p$, we proceed to the scheduling for time τ_p and $\tau_p + 1$; the algorithm dynamically updates the priority list. The job on the top of the list at time τ_p is picked and loaded to the first available machine. For a set of unit jobs, the algorithm is static since the priority list is constant over time.

It is easy to show that if the common due-date is zero, any time the job spends in waiting and processing would incur a penalty. Thus, the problem becomes the flowtime scheduling problem, which we are to discuss in the next section.

To conclude the discussion on due date based performance measures we list the papers that we have discussed in Table 4.

Table 4
A list of parallel-machine scheduling papers dealing with due-date based, performance measures

McNaughton (1959)
Roots (1965)
Horn (1974)
Rajaraman (1975, 1977)
Brucker et al. (1977)
Lenstra et al. (1977)
Dogramaci and Surkis (1979)
Graham et al. (1979)
Lawler (1982)
Monma (1982)
Cheng (1989)

9. Parallel-machine flowtime based performance measures

McNaughton (1959) has shown that the flowtime problem is a restricted form of the tardiness problem. For the single machine, ordering the jobs with weighted SPT without any preemption can yield the optimal sequence. SPT is a list schedule algorithm for which the priority list is ordered in ascending order of job processing times and the job on the top of the list is picked and loaded on the first available machine. If the weights in the set of jobs are all one, the optimal schedule can be found by SPT in $O(n \log n)$ (Conway et al., 1967). Merten (1970) provided a modified SPT by ordering the set of independent non-preemptive unit jobs according to the ascending order of the product of weights and processing times of a job. The highest level first algorithm by Coffman and Graham (1972) yields the optimal schedule for the two-machine system with a set of precedence unit jobs if equal weights are allowed. Although, with an arbitrary number of machines, this problem is fixed, it is still NP-complete (Lenstra et al., 1977). Bruno et al. (1974) proved that even a two-machine system for finding the weighted sum of flowtime with an unequally weighted set of jobs is NP-complete.

For an in-tree or out-tree set of jobs to be loaded to systems with more than two machines, the problem is NP-complete (Sethi, 1977). Similarly, Sethi showed that for an arbitrary precedence relation, for any number of machines, the problem is also NP-complete.

Table 5
NP-completeness results of parallel-machine weighted flowtime scheduling problems

Independent non-preemptive jobs
Conway et al. (1967), $n/m/w_i = 1/\Sigma F_i$, P, SPT.
Merten (1970), $n/m/t_i = 1/\Sigma F_{w_i}$, P, shortest $w_i t_i$.
Bruno et al. (1974), $n/2//\Sigma F_{w_i}$, NPC.
Baker and Merten (1973), $n/m//\Sigma F_{w_i}$, NPC.

Independent preemptive jobs
Graham et al. (1979), $n/2/\text{pmtn}/\Sigma F_{w_i}$, NPC.

Non-preemptive partial order jobs
Coffman and Graham (1972), $n/2/\text{prec}$, $t_i = 1/\Sigma F_i$, P, highest level first.
Lenstra et al. (1977), $n/m/\text{prec}$, $t_i = 1$, $w_i = 1/\Sigma F_i$, NPC.
Sethi (1977), $n/m \geq 2/\text{in-tree or out-tree}/\Sigma F_{w_i}$, NPC.

As expected, if the weights are not all equal, the weighted flowtime problem can be shown to be NP-complete (Baker and Merten, 1973). We conclude the discussion of NP-completeness results of weighted flowtime problems by Table 5.

Lawler (1964) proposed to use the transportation algorithm to solve the minimum weighted flowtime problems for a set of unit jobs, given that the weighted function is monotonically increasing with time, even though the function is non-linear. Eastman et al. (1964) developed the bounds of the minimum value of the weighted flowtime. If we let F_{w_1} denote the minimal weighted flowtime if there were only one machine, F_{w_n} the minimal weighted flowtime if there were n machines available and F_{w_m} the desired result, the lower bound is

$$L_n = \max\left\{ F_{w_1}, \frac{(m-1)F_{w_1}}{2m} + \frac{F_{w_n}}{m} \right\}.$$

Elmaghraby and Park (1974) developed a branch and bound method for the weighted flowtime problem for the case that the due-date of a job is equal to its processing time. Their method was modified by Branes and Brennan (1977). Sarin et al. (1988) proposed a new branching scheme which is superior to that of Branes and Brennan. They noted that the due-date allowance for each job is constant over time; hence, whether or not the due-date of a job is equal to its processing time does not affect the optimal schedule.

McNaughton (1959) proved that, if there is a schedule σ with at least one job preemption for a system of m machines, then there exists at least one other schedule σ' which has no job preemption, provided $F_w(\sigma') \leq F_w(\sigma)$. Another fact was provided by Eastman et al. (1964) which stated that any optimal schedule must have WSPT job orderings at each machine. Thus, to minimize the sum of weighted flowtimes over all machines, the sum of weighted flowtimes must be minimized for each machine (Baker and Merten, 1973). Another fact is that an optimal schedule cannot contain an empty machine unless $m > n$. A better schedule can be constructed by moving any job from any filled machine, if the number of jobs on that machine is more than one, to the empty machine (Baker and Merten, 1973). Elmaghraby and Park (1974) have shown that, in an optimal schedule, of two jobs J_i and J_j with $t_i \leq t_j$ and $w_i \geq w_j$, job i must precede job J_j. Another necessary conditions given by Elmaghraby and Park (1974) for any optimal schedule is that $f_k \geq s_l$ for every pair of machines k, l, where f_k is the finished time of last job on machine M_k and s_l is the starting time of last job on machine M_l.

Baker and Merten (1973) studied the sum of equal weighted flowtimes problems. If J is partitioned into $J_1', J_2', \ldots, J_j', \ldots, J_m'$, they showed that $||J_j'| - |J_k'|| \leq 1$ for any two subsets of jobs J_j' and J_k'. Therefore, interchanging jobs which have the same order of processing in any pair of machines will not change the ΣF_i, so that it is sufficient to consider the heuristic strategy of m-jobs-at-a-time rather than one-job-at-a-time, because $||J_j'| - |J_k'|| \leq 1$ (m-jobs-at-a-time means to as-

Table 6
A list of parallel-machine scheduling papers dealing with flowtime based performance measures

McNaughton (1959)
Eastman et al. (1964)
Lawler (1964)
Conway et al. (1967)
Merten (1970)
Coffman and Graham (1972)
Baker and Merten (1973)
Bruno et al. (1974)
Elmaghraby and Park (1974)
Branes and Brennan (1977)
Lenstra et al. (1977)
Sethi (1977)
Graham et al. (1979)
Sarin et al. (1988)

sign m jobs to m machines at a time, thus exactly one job will be processed by one machine). Baker and Merten (1973) then performed a statistical analysis of simulation tests for the bounds on performance ratios of a set of unequal weighted jobs using several list scheduling heuristic strategies, including WSPT, WLPT, SPT and LPT. They conclude that WSPT is out-performed if either the heuristic strategy of assigning m-jobs-at-a-time or one-job-at-a-time is used.

This concludes the discussion on flowtime based performance measure. We include in Table 6 the list of papers that we have discussed.

10. Research potential and future directions

The following are some of the future research directions. In several instances of scheduling problems, a search of the optimal solution in polynomial time for an arbitrary number of machines seems to be intractable. It is not always easy to find a single algorithm to accommodate all instances for different numbers of machines. Some attempts have been made to solve the problem for a fixed number of machines. Thus, for each (or some) fixed number of m, a particular polynomial algorithm is found. For instance, Muntz and Coffman (1969, 1970) developed an algorithm which yielded optimal schedules for two-machine systems, but for systems with more than three machines the algorithm would yield a sub-optimal schedule. Garey et al. (1983) split the opposing forest problem to the cases of two-, four- and five-machine systems and provided different polynomial algorithms for each of these cases.

Kim (1987) studied the backward approach of list scheduling heuristic rules. However, he has only surveyed the statistical performance of these rules. Instead, the worst cases of these backward rules should be studied. Since the existing research results on the worst cases analysis of these forward heuristic rules are plentiful, it is interesting and appropriate to study the worst case analysis of the backward approach applying to these results to compare the superiority of two approaches.

It is a reasonable guess that there must be some relationship between the optimal schedules of two related instances where on allows preemptive jobs and where one does not. Indeed, job preemption should not only be looked on as the relaxation of the non-preemptive constraint which is usually used to find the lower bound of the objective function value. Hence, one should investigate the similarities of any pair of related instances and try to find the commonalities between their optimality conditions.

Bicriterion scheduling has aroused interest among researchers of the single machine problem (Dileepan and Sen, 1988; Sen et al., 1988; Van Wassenhove and Baker, 1982; Van Wassenhove and Gelders, 1980). Gonzalez and Johnson (1980) mentioned that their algorithm can minimize the makespan with a limited number of job preemptions. Sin (1989) has proposed to include minimizing the number of job preemptions as a secondary measure in addition to minimizing the makespan with due-dates. There are two major approaches to solving bicriterion scheduling problems. One approach is to first optimize the major criterion, usually the soft criterion, by considering the other criterion, usually the hard criterion, as system constraints. Hence, priority of resource allocation is given to the major criterion. Following this, a trade-off between the two criteria can be used, by linear combinations such as addition, multiplication or division. Then the question is transformed to find the optimal schedule so as to satisfy the new criterion.

Friesen and Langston (1989) discussed the possibility to merge two different algorithms into one, which they called the hybrid algorithm to yield a better schedule for a given problem.

After more than two decades of research on the worst case analysis of minimum makespan problems, the attention of researchers has turned to the study of the distribution of the performance bound. However, the results of this topic are, so far, very limited by assuming the data as independent and identically distributed random variables such as uniformly distributed job processing times. Coffman and Gilbert (1985) studied this problem by assuming that the distribution of the job processing time follows the exponential distribution. The probability analysis of minimum makespan problems is very useful, since the worst case analysis sometimes does not reveal the true behaviour of some scheduling instances.

Among some instances of variants of bin packing problems, maximizing the minimum level of bin capacity is a new research topic which shares a

situation similar to that of the minimum makespan problem (Deuermeyer et al., 1982).

In this paper we have reported the state-of-the-art research results in deterministic parallel-machine scheduling theory. The review reveals that there is an abundance of potential research areas that deserve further study.

Acknowledgements

We would like to thank Professor H.J. Boom for his helpful comments. This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant OPG0036424.

References

Achugbue, J.O., and Chin, F.Y. (1981), "Bounds on schedules for independent tasks with similar execution time", *Journal of ACM* 28, 81–99.

Baker, K.R. (1974), *Introduction to Sequencing and Scheduling*, Wiley, New York.

Baker, K.R., and Merten, A.G. (1973), "Scheduling with parallel processors and linear delay costs", *Naval Research Logistics Quartery* 20, 793–804.

Barnes, J.W., and Brennan, J.J. (1977), "An improved algorithm for scheduling jobs on identical machines", *AIIE Transactions* 9, 25–31.

Bellman, R., Esogbue, A.O., and Nabeshima, I. (1982), *Mathematical Aspects of Scheduling and Applications*, Pergamon, Oxford.

Błazewicz, J., Finke, G., Haupt, R., and Schmidt, G. (1988), "New trends in machine scheduling", *European Journal of Operational Research* 37, 303–317.

Brucker, P., Garey, M.R., and Johnson, D.S. (1977), "Scheduling equal-length tasks under treelike precedence constraints to minimize maximum lateness", *Mathematics of Operations Research* 2, 275–284.

Bruno, L.J., Coffman, Jr., E.G., and Sethi, R. (1974), "Scheduling independent tasks to reduce mean finishing time", *ACM Communications* 17, 382–387.

Bruno, L.J., and Downey, P.J. (1985), "Probabilistic bounds for dual bin packing", *Acta Informatica* 22, 333–345.

Chen, N.F., and Liu, C.L. (1975), "On a class of scheduling algorithm for multi-processor computing systems", in: G. Goos and J. Hartmanis (ed.), *Parallel Processing, Lecture Notes in Computer Science 24*, Springer, Berlin, 1–16.

Cheng, T.C.E. (1989), "A heuristic for common due-date assignment and job scheduling on parallel machines", *Journal of the Operational Research Society* 40, 1129–1135.

Coffman, Jr., E.G. (ed.) (1976), *Computer and Job Shop Scheduling Theory*, Wiley, New York.

Coffman, Jr., E.G. (1982), "An introduction to proof techniques for bin-packing approximation algorithm", in: M.A.H. Dempster, J.K. Lenstra and A.H.G. Rinnooy Kan (eds.), *Deterministic and Stochastic Scheduling*, D. Reidel Publishing Company, Dordrecht, 245–270.

Coffman, Jr., E.G., Frederickson, G.N., and Lueker, G.S. (1982), "Probabilistic analysis of the LPT processor scheduling heuristic", in: M.A.H. Dempster, J.K. Lenstra and A.H.G. Rinnooy Kan (eds.), *Deterministic and Stochastic Scheduling*, D. Reidel Publishing Company, Dordrecht, 319–331.

Coffman, Jr., E.G., Frederickson, G.N., and Lueker, G.S. (1984), "A note on expected makespans for largest-first sequences of independent tasks on two processors", *Mathematics of Operations Research* 9, 260–266.

Coffman, Jr., E.G., Garey, M.R., and Johnson, D.S. (1978), "An application of bin-packing to multiprocessor scheduling", *SIAM Journal of Computing* 7, 1–17.

Coffman, Jr., E.G., Garey, M.R., and Johnson, D.S. (1984), "Approximate algorithm for bin packing – An updated survey", in: G. Ausiello, M. Lucertini and P. Serafini (eds.), *Algorithm Design for Computer System Design*, Springer-Verlag, Wien, 49–106.

Coffman, Jr., E.G., and Gilbert, E.N. (1985), "On the expected relative performance of list scheduling", *Operations Research* 33, 548–561.

Coffman, Jr., E.G., and Graham, R.L. (1972), "Optimal scheduling for two-processor systems", *Acta Informatica* 1, 200–213.

Coffman, Jr., E.G., and Leung, J.Y.T. (1979), "Combinatorial analysis of an efficient algorithm processor and storage allocation", *SIAM Journal of Computing* 8, 202–217.

Coffman, Jr., E.G., and Sethi, R. (1976), "A generalized bound on LPT sequencing", *RAIRO-Informatique* 10, 17–25.

Conway, R.W., Maxwell, W.L., and Miller, L.W. (1967), *Theory of Scheduling*, Addison-Wesley, Reading, MA.

Cook, S.A. (1971), "The complexity of theorem-proving procedures", *Proceedings of 3rd Annual ACM Symposium on Theory of Computing*, 151–158.

Davida, G.I., and Linton, D.J. (1976), "A new algorithm for the scheduling of tree structured tasks", *Proceedings of 1976 Conference on Information Sciences and Systems*, Johns Hopkins University, 543–548.

De, P., and Morton, T.E. (1980), "Scheduling to minimum makespan on unequal parallel processors", *Decision Science* 11, 586–602.

Dempster, M.A.H., Lenstra, J.K., and Rinnooy Kan, A.H.G. (eds.) (1982), *Deterministic and Stochastic Scheduling*, D. Reidel Publishing Company, Dordrecht.

Deuermeyer, B.L., Friesen D.K., and Langston M.A. (1982), "Maximizing the minimum processor finish in a multiprocessor system", *SIAM Journal of Algorithms and Discrete Mathematics* 3, 190–196.

Dileepan, P., and Sen, T. (1988), "Bicriterion static scheduling research for single machine", *OMEGA* 16, 53–59.

Dobson, G. (1984), "Scheduling independent tasks on uniform processors", *SIAM Journal of Computing* 13, 705–716.

Dogramaci, A., and Surkis, J. (1979), "Evaluation of a heuristic for scheduling independent jobs on parallel identical processors", *Management Science* 23, 1208–1216.

Dror, M., Stern, H.I., and Lenstra, J.K. (1987), "Parallel machine scheduling: Processing rates dependent on number of jobs in operation", *Management Science* 33, 1001–1009.

Eastman, W.L., Even, S., and Isaacs, I.M. (1964), "Bounds for the optimal scheduling of n jobs on m processors", *Management Science* 11, 268–279.

Elmaghraby, S., and Park, S.H. (1974), "Scheduling jobs on a number of identical machines", *AIIE Transactions* 6, 1–13.

Finn, G., and Horowitz, E. (1979), "A linear time approximation algorithm for multiprocessor scheduling", *BIT* 19, 312–320.

French, S. (1982), *Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop*, Ellis Horwood, Chichester.

Frenk, J.B.G., and Rinnooy Kan, A.H.G. (1986), "The rate of convergence to optimality of the LPT rule", *Discrete Applied Mathematics* 14, 187–197.

Frenk, J.B.G., and Rinnooy Kan, A.H.G. (1987), "The asymptotic optimality of the LPT rule", *Mathematics of Operations Research* 12, 241–254.

Friesen, D.K. (1984), "Tighter bounds for the MULTIFIT scheduling algorithm", *SIAM Journal of Computing* 13, 170–181.

Friesen, D.K., and Langston, M.A. (1983), "Bounds for MULTIFIT scheduling on uniform processors", *SIAM Journal of Computing* 12, 60–70.

Friesen, D.K., and Langston, M.A. (1989), "Analysis of a compound bin-packing", *SIAM Journal of Computing* 18, 82–90.

Fujii, M., Kasami, T., and Ninomiya, K. (1969), "Optimal sequencing of two equivalent processors", *SIAM Journal of Applied Mathematics* 17, 784–789.

Fujii, M., Kasami, T., and Ninomiya, K. (1971), "Optimal sequencing of two equivalent processors – Erratum", *SIAM Journal of Applied Mathematics* 20, 141.

Gabow, H.N. (1982), "An almost-linear algorithm for two-processor scheduling", *Journal of ACM* 29, 766–780.

Garey, M.R., and Johnson, D.S. (1976), "Scheduling tasks with nonuniform deadlines on two processors", *Journal of ACM* 23, 461–467.

Garey, M.R., and Johnson, D.S. (1979), *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA.

Garey, M.R., Johnson, D.S., Tarjan, R.E., and Yannakais, M. (1983), "Scheduling opposing forests", *SIAM Journal of Algorithms and Discrete Mathematics* 4, 72–93.

Gonzalez, M.J. (1977), "Deterministic processor scheduling", *ACM Survey* 9, 173–204.

Gonzalez, T, and Johnson, D.B. (1980), "A new algorithm for preemptive scheduling of trees", *Journal of ACM* 27, 287–312.

Goyal, D.K. (1977), "Non-preemptive scheduling of unequal executive time tasks on two identical processors", Technical Report *CS-77-039*, Computer science Department, Washington State University, Pullman, Washington, DC.

Graham, R.L. (1966), "Bounds on certain multiprocessing anomalies", *Bell System Technical Journal* 45, 1563–1581.

Graham, R.L. (1969), "Bounds on multiprocessing timing anomalies", *SIAM Journal of Applied Mathematics* 17, 416–429.

Graham, R.L. (1976), "Bound on the performance of scheduling algorithms", in: E.G. Coffman, Jr. (ed.), *Computer and Job Shop Scheduling Theory*, Wiley, New York, 165–224.

Graham, R.L., Lawler, E.L., Lenstra, J.K., and Rinnooy Kan, A.H.G. (1979), "Optimization and approximation in deterministic sequencing and scheduling: A Survey", *Annals of Discrete Mathematics* 5, 287–326.

Gupta, S.K., and Kyparisis, J. (1987), "Single machine scheduling research", *OMEGA* 15, 207–227.

Hochbaum, D.S., and Shmoys, D.B. (1987), "Using dual approximation algorithms for scheduling problems: theoretical and practical results", *Journal of ACM* 34, 144–162.

Horn, W.A. (1973), "Minimizing average flow time with parallel machines", *Operations Research* 21, 846–847.

Horn, W.A. (1974), "Some simple scheduling algorithms", *Naval Research Logistics Quarterly* 21, 177–185.

Horvath, E.C., Lam, S., and Sethi, R. (1977), "Algorithm for preemptive scheduling", *Journal of ACM* 24, 32–43.

Hu, T.C. (1961), "Parallel sequencing and assembly line problems", *Operations Research* 9, 841–848.

Ibarra, O.H., and Kim, C.E. (1976), "On two-processor scheduling of one- or two-unit time tasks with precedence constraints", *Journal of Cybernetics* 5, 87–109.

Johnson, D.S. (1972), "Fast allocation algorithms", *Proceedings of 13th Annual IEEE Symposium on Switching and Automata Theory*, 144–154.

Johnson, D.S. (1973), "Near optimal bin-packing algorithm", Ph.D. Dissertation, Electrical Engineering Department, Massachusetts Institute of Technology.

Kao, T.Y., and Elsayed, E.A. (1988), "Performance of the LPT algorithm in multiprocessor scheduling", IE working paper 88-109, Rutgers University, NJ.

Karmarkar, N., and Karp, R.M. (1982), "The differencing method of set partitioning", Computer Sciences Div., University of California, Berkeley, CA.

Kariv, O. (1976), "An O(n2.5) algorithm for finding a maximum matching in a general graph", Ph.D. Dissertation, Weizmann Institute of Science, Rehovot.

Karp, R.M. (1972), "Reducibility among combinatorial problems", in: R.E. Miller and J.W. Thatcher (eds.), *Complexity of Computer Computations*, Plenum Press, New York, 85–103.

Karp, R.M. (1975a), "On the computational complexity of combinatorial problems", *Networks* 5, 45–68.

Karp, R.M. (1975b), "The fast approximate solution of hard combinatorial problems", *Proceedings of 6th South Eastern Conference on Combinations, Graph, Theory and Computing*, Utilitas Mathematica Publishing Co., Winnipeg, 15–34.

Kaufman, M.T. (1974), "An almost-optimal algorithm for the assembly line scheduling problem", *IEEE Transactions on Computing* c-23, 1169–1174.

Kim, Y. (1987), "On the superiority of a backward approach in list scheduling algorithms for multi-machine makespan problem", *International Journal of Production Research* 25, 1751–1759.

Krause, K.L., Shen, Y.Y., and Schwetman, H.D. (1975), "Analysis of several task-scheduling algorithms for a model of multi-programming computer systems", *Journal of ACM* 22, 522–550.

Kunde, M. (1981), "Nonpreemptive LP-scheduling on homogeneous multiprocessor system", *SIAM Journal of Computing* 10, 151–173.

Lam, S., and Sethi, R. (1977), "Worst case analysis of two scheduling algorithms", *SIAM Journal of Computing* 6, 518–536.

Langston, M.A. (1982), "Improved LPT scheduling for identical processor systems", *RAIRO Technique et Science Informatique* 1, 69–75.

Lawler, E.L. (1964), "On scheduling problems with deferral costs", *Management Science* 11, 280–288.

Lawler, E.L., and Labetoulle, J. (1978), "On preemptive scheduling on unrelated and parallel processors by linear programming", *Journal of ACM* 25, 612–619.

Lawler, E.L., Lenstra, J.K., and Rinnooy Kan, A.H.G. (1982), "Recent developments in deterministic sequencing and scheduling: A survey", in: M.A.H. Dempster, J.K. Lenstra and A.H.G. Rinnooy Kan (eds.), *Deterministic and Stochastic Scheduling*, D. Reidel Publishing Company, Dordrecht, 35–74.

Lawler, E.L., and Moore, J.M. (1969), "A functional equation and its application to resource allocation and sequencing problem", *Management Science* 16, 77–84.

Lenstra, J.K. (1977), *Sequencing by Enumeration Methods*, Mathematisch Centrum, Amsterdam.

Lenstra, J.K., and Rinnooy Kan, A.H.G. (1979), "Computational complexity of discrete optimization", in: J.K. Lenstra, A.H.G. Rinnooy Kan and P. Van Emde Boas (eds.), *Interfaces Between Computer Science and Operations Research, Proceedings of a Symposium held at the Mathematisch Centrum, Amsterdam*, Mathematisch Centrum, 64–85.

Lenstra, J.K., and Rinnooy Kan, A.H.G. (1984a), "An introduction to multiprocessor scheduling", *Consiglio Nazionale della Ricerche* 5, Roma.

Lenstra, J.K., and Rinnooy Kan, A.H.G. (1984b), "Scheduling theory since 1981: An annotated bibliography", in: M. O'Eigertaigh, J.K. Lenstra and A.H.G. Rinnooy Kan (eds.), *Combinatorial Optimization: Annotated Bibliographies*, Wiley, Chichester.

Lenstra, J.K., Rinnooy Kan, A.H.G., and Brucker, P. (1977), "Complexity of machine scheduling problems", *Annals of Discrete Mathematics* 1, 343–362.

Leung, J.Y. (1982), "On scheduling independent tasks with restricted execution times", *Operations Research* 30, 163–171.

Loulou, R. (1984), "Tight bounds and probabilistic analysis of two heuristics for parallel processor scheduling", *Mathematics of Operations Research* 9, 142–150.

McHugh, J.A.M. (1984), "Hu's precedence tree scheduling algorithm: A simple proof", *Naval Research Logistics Quarterly* 31, 409–411.

McNaughton, R. (1959), "Scheduling with deadlines and loss functions", *Management Science* 6, 1–12.

Merten, A.G. (1970), "Some quantitative techniques for file organization", Technical Report No. 15, University of Wisconsin Computer Centre.

Monma, C.L. (1982), "Linear-time algorithms for scheduling on parallel processors", *Operations Research* 30, 116–124.

Muramatsu, R., Ishii, K., and Takahashi (1985), "Some ways to increase flexibility in manufacturing systems", *International Journal of Production Research* 23, 691–703.

Muntz, R.R., and Coffman, E.G. (1969), "Optimal preemptive scheduling on two processor systems", *IEEE Transactions on Computing* 18, 1017–1020.

Panwalkar, S.S., and Iskander, W. (1977), "A survey of scheduling rules", *Operations Research* 25, 45–61.

Papadimitriou, C.H., and Steiglitz, K. (1982), *Combinatorial Optimization: Algorithm and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.

Rajaraman, M.K. (1975), "An algorithm for scheduling parallel processors", *International Journal of Production Research* 13, 479–486.

Rajaraman, M.K. (1977), "A parallel sequencing algorithm for minimizing total cost", *Naval Research Logistics Quarterly* 24, 473–481.

Rinnooy Kan, A.H.G. (1976), *Machine Scheduling Problems: Classification, Complexity and Computations*, Martinus Nijhoff, The Hague.

Root, J.G. (1965), "Scheduling with deadlines and loss functions on k parallel machines", *Management Science* 11, 460–475.

Rothkopf, M.H. (1966), "Scheduling independent tasks on parallel processors", *Management Science* 12, 437–447.

Sahni, S. (1976), "Algorithms for scheduling independent tasks", *Journal of ACM* 23, 116–127.

Sahni, S. (1977), "General techniques for combinatorial approximation", *Operations Research* 25, 920–927.

Sahni, S. (1979), "Preemptive scheduling with due dates", *Operations Research* 27, 925–934.

Sarin, S.C., Ahn, S., and Bishop, A.B. (1988), "An improved branching scheme for the branch and bound procedure of scheduling n jobs on m parallel machines to minimize total weighted flowtime", *International Journal of Production Research* 26, 1183–1191.

Sen, T., Raiszaden, F.M.E., and Dileepan, P. (1988), "A branch-an-bound approach to the bicriterion scheduling problem involving total flowtime and range of lateness", *Management Science* 34, 254–260.

Sethi, R. (1976a), "Algorithm for minimum-length schedule", in: E.G. Coffman, Jr. (ed.), *Computer and Job Shop Scheduling Theory*, Wiley, New York, 51–100.

Sethi, R. (1967b), "Scheduling graphs on two processors", *SIAM Journal of Computing* 5, 73–82.

Sethi, R. (1977), "On the complexity of mean flow time scheduling", *Mathematics of Operations Research* 2, 320–330.

Sin, C.C.S. (1989), "Some topics of parallel-machine scheduling theory", *Unpublished M.Sc.* Thesis, Department of Actuarial and Management Sciences, University of Manitoba.

Stinson, D.R. (1987), *An Introduction to the Design and Analysis of Algorithms* (2nd Edition), Charles Babbage Research Centre, Winnipeg.

Tarjan, R.E. (1975), "Efficiency of a good but not linear set union algorithm", *Journal of ACM* 22, 215–225.

Ullman, J.D. (1975), "NP-complete scheduling problem", *Journal of Computing System Science* 10, 384–393.

Ullman, J.D. (1976), "Complexity of sequencing problems", in: E.G. Coffman, Jr. (ed.), *Computer and Job Shop Scheduling Theory*, Wiley, New York, 139–164.

Van Wassenhove, L.N., and Baker, K.R. (1982), "A bicriterion approach to time/cost trade-offs in sequencing", *European Journal of Operational Research* 11, 48–54.

Van Wassenhove, L.N., and Gelders, L.F. (1980), "Solving a bicriterion scheduling problem", *European Journal of Operational Research* 4, 42–48.