



Fifty years of scheduling: a survey of milestones

CN Potts^{1*} and VA Strusevich²

¹University of Southampton, Southampton, UK; and ²University of Greenwich, London, UK

Scheduling has become a major field within operational research with several hundred publications appearing each year. This paper explores the historical development of the subject since the mid-1950s when the landmark publications started to appear. A discussion of the main topics of scheduling research for the past five decades is provided, highlighting the key contributions that helped shape the subject. The main topics covered in the respective decades are combinatorial analysis, branch and bound, computational complexity and classification, approximate solution algorithms and enhanced scheduling models.

Journal of the Operational Research Society (2009) **60**, S41–S68. doi:10.1057/jors.2009.2

Keywords: scheduling; history; milestones

1. Introduction

Scheduling is one of the most widely researched areas of operational research, which is largely due to the rich variety of different problem types within the field. A search on the Web of Science for publications with ‘scheduling’ and ‘machine’ as topics yields over 200 publications for every year since 1996, and 300 publications in 2005, 2006 and 2007. Arguably, the field of scheduling traces back to the early 20th century with Gantt (1916) explicitly discussing a scheduling problem. However, it was about 40 years later that a sustained collection of publications on scheduling started to appear. Nevertheless, scheduling has a long history relative to the lifetime of the main operational research journals, with several landmark publications appearing in the mid-1950s.

In this paper, we survey some of the key milestones in deterministic scheduling research. We focus on the types of scheduling problems that arise in production industries; topics such as vehicle scheduling, timetabling and personnel scheduling lie outside the scope of this survey. These milestones correspond to papers that have typically initiated a new thread of scheduling research. To provide some insight into the development of the field over time, we have indicated the topical research themes for each of five decades, where the first decade starts in the mid-1950s, the second in the mid-1960s, and so on. However, the boundaries between decades should be regarded as a rough guide only.

Each of the five following sections presents key research themes and milestones for the relevant decade. We also mention activities relating to scheduling such as the publication of influential books, the launch of a scheduling journal and the running of major conferences or the initiation of a

new conference series. To conclude the paper, we discuss possible future research topics for the coming decade and possibly beyond.

2. Decade 1: combinatorial analysis

Contributions to scheduling in the first decade were largely on the introduction of new models and the use of combinatorial analysis for developing solution algorithms. Our focus is on machine scheduling in which a set of jobs is to be processed on one or more machines. The scheduling challenge arises because each machine can process at most one job at a time and each job can be processed on at most one machine at a time.

2.1. Johnson (1954) and the flow shop

It is generally believed that the seminal paper by Johnson (1954) provided the starting point to scheduling being regarded as an independent area within operational research. Johnson considers the production model that is now called the *flow shop*. In the general flow shop, we are given a set $N = \{1, \dots, n\}$ of jobs and m successive machines $1, \dots, m$, where $m \geq 2$. Each job j consists of m operations, O_{1j}, \dots, O_{mj} , where each operation O_{ij} is to be processed on machine i for p_{ij} time units. Each job j has the same *processing route*; the order of the processing of its operations is given by the sequence (O_{1j}, \dots, O_{mj}) , or, equivalently, each job j visits the machines in the order $(1, \dots, m)$.

The main result obtained by Johnson (1954) is an elegant solution procedure for solving the two-machine flow shop problem to minimize the *makespan*, or equivalently the maximum completion time by which all jobs are completed on all machines. Johnson shows that it suffices to restrict the search for an optimal solution among those schedules for which the sequence of jobs is the same on each of the two

*Correspondence: CN Potts, School of Mathematics, University of Southampton, Southampton SO17 1BJ, UK.

E-mail: C.N.Potts@soton.ac.uk

machines; schedules of this type are now called *permutation* schedules. Moreover, Johnson (1954) proves that there exists a optimal schedule in which a job j is sequenced earlier than another job k , if $\min\{p_{1j}, p_{2k}\} \leq \min\{p_{2j}, p_{1k}\}$. The most common algorithm that is based on this property is as follows: an optimal sequence starts with the jobs for which $p_{1j} \leq p_{2j}$ sorted in non-decreasing order of p_{1j} , followed by the remaining jobs sorted in non-increasing order of p_{2j} . Another implementation of the same approach assigns to each job j a *priority* index $\text{sgn}(p_{1j} - p_{2j})(W - \min\{p_{1j}, p_{2j}\})$ and sorts the jobs in non-decreasing order of these priorities; here, W is a large number such that $W > \min\{p_{1j}, p_{2j}\}$ for all jobs j .

It is interesting to notice that historically the first scheduling result addresses not the simplest scheduling model: the flow shop has two (or more) machines, while the simplest machine environment that one could perceive consists of just one machine.

2.2. Jackson (1955), Smith (1956) and single machine scheduling

In single machine scheduling, the processing time of job j on the machine is denoted by p_j . Since all sequences yield the same makespan, other objective functions are of interest. The first results on single machine scheduling appeared immediately after the work by Johnson, and provided algorithms that are also based on *priority rules*.

Given a schedule, the *lateness* of a job j is defined as $C_j - d_j$, where C_j is its completion time and d_j is a given *due date*. The *maximum* lateness, traditionally denoted by L_{\max} , gives a measure of how much the original due dates need be changed, so that in the resulting schedule each job will be completed by its modified due date. Jackson (1955) shows that a sequence which minimizes L_{\max} is given by the *earliest due date* priority rule (*EDD* rule) in which the jobs are sorted in non-decreasing order of their due dates d_j .

Smith (1956) addresses the single machine problem of minimizing the sum of the completion times $\sum C_j$. If divided by the number of jobs n , this criterion represents an average time spent by a job in the system, which is a widely used performance measure in queueing theory. An optimal permutation is obtained by sequencing the jobs in non-decreasing order of their processing times; this rule is known as the *shortest processing time* rule (*SPT* rule). Moreover, as shown by Smith (1956), if each job j has an associated positive *weight* w_j which represents its relative importance, then the sum of weighted completion times $\sum w_j C_j$ is minimized by sequencing the jobs in non-decreasing order of the ratios p_j/w_j ; this priority rule is often referred to as *Smith's rule* or the *SWPT rule*.

A common technique used for proving the optimality of a certain priority rule is based on a *pairwise interchange* argument. Suppose that there exists an optimal sequence in which some pair of adjacent jobs does not obey the rule under

consideration. If it can be proved that when the order of this pair of jobs is reversed, the objective function does not get worse, then ordering the jobs according to the rule is a sufficient condition for optimality of the resulting sequence. Below, we illustrate this technique by giving a brief proof of the optimality of the SWPT rule.

Suppose that for the problem of minimizing $\sum w_j C_j$ on a single machine there exists a schedule S that is optimal, but not obtained by the SWPT rule. Without loss of generality assume that the jobs are numbered in such a way that in S they are processed according to the sequence $(1, \dots, n)$. Scanning the jobs in the order of this numbering, identify a pair of adjacent jobs k and $k+1$ such that $p_k/w_k > p_{k+1}/w_{k+1}$, which is equivalent to $w_{k+1}p_k > w_k p_{k+1}$. For schedule S , the value of the objective function can be written as

$$F(S) = \sum_{j=1}^n w_j C_j = \sum_{j=1}^{k-1} w_j C_j + w_k \sum_{i=1}^k p_i + w_{k+1} \sum_{i=1}^{k+1} p_i + \sum_{j=k+2}^n w_j C_j.$$

Consider a new schedule S' that is obtained from S by swapping jobs k and $k+1$. It follows that in S' the completion times of all jobs other than k and $k+1$ remain the same as in schedule S , while jobs k and $k+1$ complete at times $\sum_{i=1}^{k+1} p_i$ and $\sum_{i=1}^{k-1} p_i + p_{k+1}$, respectively. This implies that

$$F(S) - F(S') = w_k \left(\sum_{i=1}^k p_i - \sum_{i=1}^{k+1} p_i \right) + w_{k+1} \left(\sum_{i=1}^{k+1} p_i - \left(\sum_{i=1}^{k-1} p_i + p_{k+1} \right) \right) = -w_k p_{k+1} + w_{k+1} p_k > 0,$$

which shows that schedule S is not optimal.

2.3. McNaughton (1959) and parallel machine scheduling

McNaughton (1959) studies problems of scheduling jobs on m identical parallel machines, where $m \geq 2$. Such problems require p_j units of processing of job j to be assigned to the machines. In the case of minimizing the makespan, a schedule is defined by the assignment of each job's processing to the machines, while their order on each machine is immaterial. Since no job can be processed by several machines at a time, it follows that the longest job duration serves as a *job-based* lower bound on the optimal makespan. Further, for any assignment of the jobs to the machines, there is a machine that is loaded for at least $\sum_{j \in N} p_j / m$ time units; this value, the average machine load, is a *machine-based* lower bound on the makespan. Let LB denote the larger of these lower bounds, so that

$$LB = \max \left\{ \frac{1}{m} \sum_{j \in N} p_j, \max_{j \in N} p_j \right\}. \tag{1}$$

If each job is processed without any interruption, then the optimal makespan can sometimes be larger than LB, that is, the bound (1) is not tight. However, if preemption is allowed, the optimal makespan is equal to LB. In a preemptive schedule, the processing of each job can be interrupted at any time and resumed later, possibly on a different machine, under the condition that the total duration of all partial processing is equal to the original processing time of the job. McNaughton (1959) gives a simple algorithm that finds an optimal preemptive schedule: imagine that all jobs are processed by a single auxiliary machine with no idle time at the start or between jobs and cut that schedule into at most m segments of length LB (except possibly the last one, which may be shorter) and interpret each of these segments as a preemptive assignment of the jobs to the m original parallel machines.

2.4. Characteristics of machine scheduling problems

By the end of the 1950s, it had become clear that the problem area of deterministic machine scheduling had formed an independent branch of operational research, with its own range of problems and solution approaches. The processing systems had been classified into *single-stage* with one machine or several machines operating in parallel, and *multi-stage* with machines operating in series.

The jobs processed in single-stage systems consist of one operation only that is either performed on a single machine or on any of the m available *parallel* machines. In turn, parallel machines can be *identical*, *uniform* or *unrelated*. Let p_{ij} denote the processing time of any job j if it is processed on a machine i , where $p_{ij} = p_j$ if the parallel machines are identical, and $p_{ij} = p_j/s_i$, where s_i denotes the speed of machine i if the machines are uniform. In the case of unrelated parallel machines, the values of p_{ij} are arbitrary.

By contrast, in multi-stage systems, each job consists of several operations, each performed by a specified machine. A distinction was made between the *flow shop* systems, in which the jobs have identical processing routes, and the *job shop* systems, in which each job has an individual fixed route.

Typically, the objective function to be minimized depends on the completion times of the jobs, and the function is regular, that is, non-decreasing with respect to each of its arguments. The most popular objective functions include the makespan, the sum of (weighted) completion times, the maximum lateness, the (weighted) number of late jobs, etc.

2.5. Other results

Jackson (1956) extends the results of Johnson (1954) to the two-machine job shop problem to minimize the makespan, provided that each job consists of at most two operations. There are several lower bounds on the optimal makespan: the total workload of each machine and the makespan of the optimal flow shop schedule for the jobs with the same processing route. Jackson (1956) describes an algorithm that

finds a schedule with makespan that is equal to the largest of these lower bounds.

Akers and Friedman (1955) study the job shop problem with two jobs, which is in some sense dual to the version of the problem considered by Jackson (1956). Here, each of the two jobs has its own processing route with possible repeated visits to the same machine. An elegant graphical interpretation is given that reduces the problem of minimizing the makespan to finding the shortest path in a rectangle with rectangular obstacles.

In the classical flow shop model, the main processing restriction is that a job can start on a machine only after it is completed on the previous machine. In numerous applications of chemical industry and metallurgy, it is required that the next operation of a jobs starts exactly when the previous operation is finished. This restriction is known as ‘no-wait in process’. Piehler (1960) was the first to demonstrate that the m -machine flow shop problem with no-wait in process to minimize the makespan reduces to the travelling salesman problem (TSP). Recall that in the TSP it is required to find the shortest closed route (a Hamiltonian tour) that visits each of the given cities exactly once. The result by Piehler (1960) was rediscovered later on several occasions. Gilmore and Gomory (1964) showed that the two-machine no-wait flow shop problem reduces to a TSP with a special structure and therefore admits a fast algorithm.

Conway *et al* (1967) consider the problem of minimizing the total completion time $\sum C_j$ on identical parallel machines. They show that an optimal schedule can be found by extending the SPT rule. Specifically, their procedure is based on the idea of what is now known as *list scheduling*: form a list of jobs (by the SPT rule for the problem under consideration) and at any time that a machine becomes available remove the first job from the current list and assign it to that machine.

Despite an understandable desire to identify a general technique that is capable of handling a wide range of scheduling problems, it was recognized that many individual problems by nature require purpose-built algorithms. Consider, for instance, a single machine problem to minimize the number of late jobs. It appears natural to apply the EDD priority rule, but the resulting sequence is not necessarily optimal. The following approach is due to Moore (1968). Renumber the jobs in the EDD order, and try to schedule them one by one until some job k cannot be completed by its due date. This implies that in any schedule at least one job of the first k jobs must be late. Since each job carries the same penalty for missing a due date, it makes sense to remove from the current partial schedule a job j , where $1 \leq j \leq k$, that has the largest processing time. The removed job is then scheduled late. The removal of a long job creates more capacity for scheduling the remaining jobs by their due dates. The process repeats until all jobs are sequenced. The outlined algorithm is often referred to as Moore’s algorithm, although in the paper of Moore (1968) this particular implementation is described as an ‘Author’s Supplement’ with a footnote that indicates this

version of the algorithm is more computationally efficient than the one in the main body of the paper and is due to T.J. Hodgson. As observed by Sturm (1970), the optimality proof given by Moore (1968) does not cover Hodgson's implementation, and a natural inductive proof is provided. Interestingly, Hodgson's algorithm, with an optimality proof by Sturm, remains known as Moore's algorithm.

2.6. Scheduling at the end of Decade 1

A considerable influence on the development of scheduling can be attributed to two books that summarize early achievements in the area: an edited collection by Muth and Thompson (1963) and a research monograph by Conway *et al* (1967). It is important to stress that both books were translated into other languages, which had the effect of extending dramatically the global community of scheduling researchers.

The papers included in the book of Muth and Thompson (1963) are mainly devoted to the issues of finding solutions to practical problems; the methods discussed include priority rules, integer programming, Monte-Carlo, stochastic analysis, algorithms with learning, enumerative algorithms, etc. The seminal paper of Johnson (1954) is also reproduced there. A contribution by G. Fisher and G.L. Thompson presents a famous benchmark job shop problem $10 \times 10 \times 10$ (10 jobs, 10 machines, 10 operations per job) which helped to motivate the search for methods of guided enumeration for more than the next 20 years.

The book by Conway *et al* (1967) combines under the same cover a collection of material on both deterministic machine scheduling theory and queueing theory. The book makes an attempt to introduce a short-hand notation for scheduling problems, by appearance similar to that widely accepted in queueing theory. For example, the flow shop problem with n jobs, m machines to minimize the total (average) completion time is denoted by Conway *et al* as $n|m|F|\bar{F}$. Some authors still follow that notation, but in the 1970s it was replaced by a three-field classification scheme.

It is interesting to notice that during the first decade there was no full understanding among researchers regarding the concept of computational complexity; we will address the relevant issues below, in Section 4. Also worthy of mention is the fact that all of the algorithms discussed thus far are what we now call polynomial-time algorithms. Typically, such algorithms were originally developed for problems of small dimension (single machine, two machines, two operations per job, or two jobs in a multi-machine job shop). As established later (after 1970), many combinatorial optimization problems become essentially more complex if a numerical parameter grows beyond two. It seems that researchers during the early days of scheduling were intuitively aware of that phenomenon.

3. Decade 2: branch and bound

Research on combinatorial analysis continued into the next decade, but with an emphasis on more challenging problems.

Typical results that were derived are of the form 'some job j precedes another job k if certain conditions are satisfied'. While not directly leading to an algorithm for solving the corresponding problem, such results are useful as dominance rules in enumerative algorithms such as dynamic programming and more particularly branch and bound.

Branch and bound is a enumerative search technique for solving combinatorial optimization problems to optimality. The branch and bound concept was developed independently in the late 1950s by Land and Doig (1960) for integer programming and Eastman (1958a, b) for the TSP. A branch and bound algorithm (for a minimization problem) is characterized by

- (a) a *branching rule* that indicates how solutions are partitioned into subsets;
- (b) a *lower bounding rule* that computes a lower bound on the cost of any solution within the subset of solutions under consideration;
- (c) optional features such as an *upper bounding rule* that constructs feasible solutions that can be evaluated to produce an upper bound, and *dominance rules* that can eliminate some subsets of solutions from further consideration.

The evolution of a branch and bound search is often represented as a *search tree*, where each node represents a subset of solutions, and the branches correspond to partitioning the solutions according to the branching rule.

The first applications of branch and bound to scheduling occurred in the mid-1960s. Many of these studies were for problems requiring a sequence of the jobs to be constructed. For such sequencing problems, a *forward sequencing branching rule* was typically used. Thus, the first branching within the search tree considers all possible jobs that can be sequenced in the first position, the second branching considers all possible jobs that can occupy the second position in the sequence, and so on. Although rather simplistic, such a branching rule has advantages, the most important being that it often allows effective dominance rules to be designed. Lower bounding schemes were usually based on solving a (highly) relaxed version of the original problem.

3.1. Flow shop

The first applications of branch and bound for scheduling involved the flow shop problem in which there are m machines and n jobs, with each job j having a processing time of p_{ij} on each machine i . Most studies considered the permutation version of the problem in which the same processing order of jobs is used on each machine (although there is no loss of generality for the case of two machines, or for three machines if the objective is to minimize the makespan). The first studies were those of Lomnicki (1965), and Ignall and Schrage (1965) who independently developed branch and bound algorithms

for minimizing the makespan in the three-machine flow shop problem (with Ignall and Schrage also providing an algorithm for minimizing the sum of completion times of the jobs in a two-machine flow shop problem). Let σ denote a partial sequence of jobs that are sequenced in the initial positions at some node of the search tree, U denote the set of unsequenced jobs and $C(\sigma, i)$ denote the completion time of the last job of σ on each machine i . Both Lomnicki (1965), and Ignall and Schrage (1965) propose (for $m = 3$) a *machine-based bound* $\max\{LB_1, \dots, LB_m\}$, where

$$LB_i = C(\sigma, i) + \sum_{j \in U} p_{ij} + \min_{j \in U} \sum_{h=i+1}^m p_{hj}.$$

McMahon and Burton (1967) introduce (for $m=3$) a *job-based bound* $\max\{LB'_1, \dots, LB'_{m-1}\}$ to be used in combination with the machine-based bound, where

$$LB'_i = C(\sigma, i) + \max_{k \in U} \left\{ \sum_{j \in U \setminus \{k\}} \min\{p_{ij}, p_{mj}\} + \sum_{h=i}^m p_{hk} \right\}.$$

Nabeshima (1967) improves the machine-based bound by including any idle time resulting from processing the operations on the preceding machine. The idle time is evaluated by solving a two-machine subproblem using the algorithm of Johnson (1954). Potts (1974) subsequently generalizes Nabeshima's lower bound by allowing idle time created from the operations on any of the previous machines to be taken into account. This more general two-machine bound was discovered independently by Lageweg *et al* (1978); they also provide a framework explaining the derivation of various lower bounds that have been proposed in the literature.

In parallel with the development of the machine-based and job-based bounds and their extensions, work on the derivation of dominance rules was carried out by numerous researchers. This was a natural extension of the earlier combinatorial analysis for the flow shop problem that was discussed in the previous section. The main idea of a dominance rule is to eliminate a search tree node in which job j is appended to the partial sequence σ to form the partial sequence σj by showing that another partial sequence $\sigma k j$ always leads to a better solution or a solution with the same value. Let $\Delta_i = C(\sigma k j, i) - C(\sigma j, i)$ for each machine i . Then $\sigma k j$ dominates σj if one of the following conditions holds:

- (a) $\Delta_{i-1} \leq p_{ij}$ and $C(\sigma k, i-1) \leq C(\sigma j, i-1)$ for $i=2, \dots, m$ (Smith and Dudek, 1969).
- (b) $\max\{\Delta_{i-1}, \Delta_i\} \leq p_{ij}$ for $i = 2, \dots, m$ (McMahon and Burton, 1967; Szwarc, 1973).
- (c) $\Delta_{i-1} \leq \Delta_i \leq p_{ij}$ for $i = 2, \dots, m$ (Szwarc, 1971).
- (d) $\max_{h=1, \dots, i} \Delta_h \leq p_{ij}$ for $i = 2, \dots, m$ (Szwarc, 1973).
- (e) $\Delta_i \leq \min_{h=i, \dots, m} p_{hj}$ for $i = 2, \dots, m$ (Gupta, 1971).

Several of these conditions are equivalent to one another. Unfortunately, unless m is small, it is difficult to satisfy the conditions that allow a search tree node to be eliminated.

By the end of the decade, the better performing branch and bound algorithms could typically solve instances routinely with up to 10 jobs. However, with more than 10 jobs the general pattern emerged that many instances were solved quickly while others remained unsolved when reasonable limits on computation time were applied. It was subsequently discovered by Potts (1980) in his experiments with simultaneous forward and backward branching that all components of a branch and bound algorithm including the branching rule must be chosen appropriately if more challenging instances are to be solved to optimality.

3.2. Single machine total weighted tardiness problem

Another problem which attracted significant research effort in the context of branch and bound algorithms and dominance rules is that of scheduling jobs on a single machine to minimize the total tardiness or the total weighted tardiness. In this problem, each job j has a given due date d_j by which time it should ideally be completed, and possibly a weight w_j indicating the importance of the job. Given a schedule in which each job j is completed at time C_j , the tardiness of job j is defined by $T_j = \max\{C_j - d_j, 0\}$.

For the single machine total tardiness and total weighted tardiness problems, dominance rules play a strong role in restricting the search in branch and bound algorithms. Such rules are particularly useful in view of the difficulty caused by the non-linearity of the tardiness function in obtaining tight lower bounds that are effective in pruning the search tree.

Most of the branch and bound algorithms use a *backward sequencing branching rule* in which the first branching fixes the last job in the sequence, the second branching fixes the penultimate job, and so on. This rule has two major advantages. First, the jobs completing later typically have a larger (weighted) tardiness, and fixing their positions early allows a tighter lower bound to be computed. Second, Elmaghraby (1968) showed that if there exists a job that has zero tardiness wherever it is sequenced, then there exists an optimal solution in which this job is sequenced in the last position. Thus, under a backward sequencing branching rule, a single branch is created whenever Elmaghraby's result is applicable to the subproblem under consideration. The subproblems associated with any node of the search tree are of the same form as the original problem, but defined over a set U of unsequenced jobs.

Some algorithms such as that of Emmons (1969) do not use lower bounds, but instead rely on dominance rules to restrict the search. An obvious lower bound on the total weighted tardiness is given by the minimum value of the maximum tardiness of the jobs obtained by sequencing the jobs in EDD order multiplied by the smallest job weight. Shwimer (1972) uses this lower bound in his branch and bound algorithm,

but with a lookahead mechanism in which the smallest lower bound of all potential child nodes is used as a lower bound for the current node. Branch and bound algorithms have also been proposed that use lower bounds obtained from the solution of a transportation problem (Gelders and Kleindorfer, 1974, 1975) and a linear assignment problem (Rinnooy Kan *et al*, 1975). Lawler (1964) was the first to propose the use of a transportation problem to obtain a lower bound. Assuming that all processing times are integers, the transportation problem has variables defined by

$$x_{jt} = \begin{cases} 1 & \text{if one unit of processing of job } j \\ & \text{is assigned a completion time of } t, \\ 0 & \text{otherwise,} \end{cases}$$

and is formulated as:

$$\begin{aligned} & \text{Minimize } \sum_{j=1}^n \sum_{t=1}^T c_{jt} x_{jt} \\ & \text{subject to } \sum_{t=1}^T x_{jt} = p_j, \quad j = 1, \dots, n \\ & \quad \quad \quad \sum_{j=1}^n x_{jt} = 1, \quad t = 1, \dots, T \end{aligned}$$

where T is the sum of the processing times and c_{jt} is a suitably defined cost. The transportation problem arises through relaxing the constraint that each unit of processing of each job j must be scheduled contiguously. Gelders and Kleindorfer (1974) suggest the use of $c_{jt} = w_j \max\{t - d_j, 0\}/p_j$ which provides a valid lower bound. Rinnooy Kan *et al* (1975) use a lower bound obtained from the solution of a linear assignment problem, where the element c_{jk} of the cost matrix is a lower bound on the weighted tardiness of assigning job j to position k . Assuming the job j is sequenced in position k , the value of c_{jk} is computed by summing the $k - 1$ smallest processing times of jobs in $N \setminus \{j\}$ and adding the processing time of job j . In both the transportation and assignment lower bounds, adjustments are made to take into account any precedences derived from dominance rules of the type described below.

As indicated above, dominance rules play a crucial role in helping to prune the search tree in branch and bound algorithms. For some optimal schedule, let B_j and A_j be the sets of jobs known through previous application of the dominance rules to be processed before and after job j , respectively. Also, let $P(Q)$ denote the total processing time of jobs in some set Q . Then, for any pair of jobs j and k , there exists an optimal schedule in which job j precedes job k if one of the following conditions holds:

- (a) $p_j \leq p_k$, $w_j \geq w_k$ and $d_j \leq \max\{d_k, P(B_k) + p_k\}$ (Shwimer, 1972; Emmons, 1969).
- (b) $d_j \leq d_k$, $w_j \geq w_k$ and $d_k \geq P(N \setminus A_j) - p_k$ (Emmons, 1969).
- (c) $d_k \geq P(N \setminus A_j)$ (Emmons, 1969).
- (d) $d_k \geq P(N)$ (Elmaghraby, 1968).

A precedence graph $G = (N, A)$ can be constructed, where the arc (j, k) is added to G whenever one of the above conditions is satisfied. The dominance rules can be applied prior to the branch and bound algorithm to obtain a precedence graph G , and also within the branch and bound algorithm at any search tree node to restrict the descendent nodes for that branch of the search tree.

The performance of the branch and bound algorithms introduced during this decade was dependent on the characteristics of the instances generated: instances with a small range of due dates and a small mean due date relative to the sum of processing times proved to be especially challenging. As for the permutation flow shop, instances with up to 10 jobs could be routinely solved by several of the available algorithms, but larger instances with the more challenging characteristics generally remained unsolved under reasonable computation time limits.

3.3. Job shop problem

Unlike the flow shop and single machine total weighted tardiness problems discussed above where a schedule is defined by a single sequence of jobs, the job shop problem requires a sequence of jobs to be found for each of the machines. We discuss the problem of minimizing the makespan, which has provided the main focus of attention in job shop scheduling. Owing to the complex structure of the job shop, dominance rules do not play a significant role in restricting the search.

A useful representation of the job shop problem is through the disjunctive graph formulation as proposed by Roy and Sussman (1964). The disjunctive graph has a node for each operation plus an initial node and a final node, with various conjunctive and disjunctive arcs. There is a conjunctive arc from the initial node to each node representing the first operation of some job, from each node representing an operation of a job to the node representing the next operation of the same job, and from each node representing the last operation of some job to the final node. These are disjunctive arcs between each pair of nodes that correspond to operations requiring the same machine. A possible solution is obtained by orienting each disjunctive arc to represent the processing order of the corresponding operations on the relevant machine, where the orientations are chosen so that the resulting graph is acyclic. By associating a zero weight with the initial and final node and a weight equal to the processing time of the corresponding operation for all other nodes, the makespan can be found by computing a path of maximum weight from the initial to the final node.

Two main types of branching rule have been proposed in the literature. Németi (1964) introduced *disjunctive arc branching*, which selects some disjunctive arc connecting nodes u and v in the disjunctive graph formulation, where u and v correspond to operations requiring the same machine. Two branches are created. In the first branch, the disjunctive arc is oriented so that the operation corresponding to u

is processed before the operation corresponding to v , while in the second branch the reverse orientation is adopted. Brooks and White (1965) proposed *active schedule generation branching*, where an active schedule is one in which no operation can be started earlier without delaying another operation. Under this branching rule, an unsequenced operation with the earliest completion time is first selected, together with the machine i that it requires. For each operation that requires machine i and has an earliest start time that is strictly less than the smallest earliest completion time, a branch is created that sequences the operation in the first unfilled position on machine i .

An obvious lower bound is obtained from the disjunctive graph formulation by removing all disjunctive arcs (except those whose orientations are already fixed) and finding the maximum weight path in the resulting graph. However, this simplistic lower bound can be improved through the use of the following single machine relaxation in which a lower bound is computed for each machine i and the best of these single machine bounds is selected. The single machine relaxation is obtained from the disjunctive graph formulation by removing, as above, all disjunctive arcs except those between operations that require machine i . Any operation u requiring machine i has a processing time on this machine, has a release date r_u (sometimes referred to as a ‘head’) that defines the earliest time that u can start, and has a delivery time q_u (sometimes referred to as a ‘tail’) that defines the minimum time that must elapse after the processing of u and the completion of the schedule. The release date r_u is the value of a maximum weight path from the initial node to the node corresponding to u but ignoring p_u , the weight of node u ; while the delivery time q_u is the value of a maximum weight path from the node corresponding to u to the final node but again ignoring p_u . Let S_i denote the set of operations that require machine i , let $r_* = \min_{u \in S_i} r_u$ and $q_* = \min_{u \in S_i} q_u$. The following lower bounds based on this subproblem have been proposed:

- (a) $r_* + \sum_{u \in S_i} p_u$ (Schrage, 1970a; Charlton and Death, 1970).
- (b) $T_i + q_*$, where T_i is the optimal makespan on machine i obtained by sequencing the jobs in non-decreasing order of r_u (Brooks and White, 1965).
- (c) $r_* + U_i$, where U_i is the optimal schedule completion time for the subproblem on machine i obtained by ignoring release dates and sequencing the jobs in non-increasing order of q_u (Schrage, 1970b).
- (d) V_i , where V_i is the optimal schedule completion time for the subproblem on machine i obtained by an enumerative algorithm (Bratley *et al*, 1973; McMahon and Florian, 1975).

In spite of the more sophisticated branch and bound algorithms that were available at the end of the decade, the famous $10 \times 10 \times 10$ instance by G. Fisher and G.L. Thompson (see Section 2.6) remained unsolved, with the best known solution

value being 972 (see McMahon and Florian, 1975) compared with 930 which is now known to be the optimal makespan. A statement by Conway *et al* (1967) that ‘many proficient people have considered the problem, and all have come away essentially empty-handed’ maintained a large element of truth a decade later.

3.4. Other results

In addition to the combinatorial analysis that was used to derive dominance rules for the flow shop, significant research activity was devoted to special cases in which certain restrictions are placed on the processing times of the operations. Nabeshima and Szwarc both contributed substantially to this area with numerous publications starting with Nabeshima (1961) and Szwarc (1968). A survey of results on efficiently solvable special cases of the flow shop is provided by Monma and Rinnooy Kan (1983).

For problems in which the key combinatorial structure is selection or assignment, dynamic programming proved to be a useful solution technique. For example, consider the problem of scheduling jobs on a single machine to minimize the weighted number of late jobs. It is straightforward to observe that there exists an optimal solution in which the early (or on-time) jobs are sequenced first in EDD order followed by the late jobs in an arbitrary order. Thus, the problem reduces to one of selecting the early jobs. Lawler and Moore (1969) introduce a dynamic programming algorithm for solving this problem. Let $F_j(t)$ be the minimum weighted number of late jobs for the subproblem involving jobs $1, \dots, j$, where the last early job completes at time t , with initial value $F_0(0) = 0$. The following recursion computes

$$F_j(t) = \begin{cases} \min\{F_{j-1}(t - p_j), F_{j-1}(t) + w_j\} & \text{for } t = 0, \dots, d_j, \\ F_{j-1}(t) + w_j & \text{for } t = d_j + 1, \dots, T, \end{cases} \quad (2)$$

for $j = 1, \dots, n$, where $T = \min\{d_n, \sum_{j=1}^n p_j\}$ is an upper bound on the completion time of the last on-time job. The minimum weighted number of late jobs is then $\min_{t=0, \dots, T} \{F_n(t)\}$.

Rothkopf (1966) shows that dynamic programming can be used for problems of scheduling jobs on identical parallel machines to minimize the makespan, total weighted completion time and maximum lateness of jobs. For these problems, once the jobs are assigned to machines, their order on each machine is known from the priority rules for the corresponding single machine scheduling problem. Thus, these problems reduce to those of assigning jobs to machines, and consequently are amenable to solution by dynamic programming.

From a chronological viewpoint, this section should contain a discussion of list scheduling approximation algorithms, as developed and analysed by Graham (1966, 1969). However, since we consider the worst-case analysis of approximation

algorithms to be one of the major research topics of Decade 4, Graham's contributions are reviewed later in Section 5.1.

3.5. Scheduling at the end of Decade 2

The end of the second decade of scheduling saw a wider range of models being studied as compared with the first decade, and greater intuition by researchers tackling those problems. However, there were still no tools for undertaking a systematic study of scheduling problems whereby it would be possible to make statements of the form 'this problem is easy if all jobs are released at the same time, but becomes hard if jobs have different release dates'.

Researchers typically resorted to branch and bound if they could not find algorithms based on combinatorial analysis. Even though intuition usually turned out to be correct in that most branch and bound studies were for problems that we now know to be NP-hard, no formal justification could be provided at that time. Generally, the performance of the branch and bound algorithms was disappointing, with sometimes no computational evaluation performed or only small instances being solved to optimality. The latter was partly due to the relative slowness of computers used during the second decade. However, the main drawback was the lack of more sophisticated features in the design of the algorithms that would effectively restrict the search. Lower bounds were often based on a relaxation of many constraints which typically produced a highly simplified problem that lacked most of the structure of the original problem.

Another textbook appeared on the scene, namely that of Baker (1974). It replaced Conway *et al* (1967) as the main scheduling textbook, and held the position of the most widely used book for many years.

4. Decade 3: complexity and classification

It would not be an exaggeration to state that this decade was the most important in the history of scheduling. This is mainly due to the development of the theory of computational complexity, and also the design of a clear classification scheme for scheduling problems. These two events solidified scheduling theory and brought it to its current shape.

4.1. Computational complexity

Scheduling, being a part of combinatorial optimization, for many years had been an object of critics from those who pursued more established areas of research, normally referred to as 'pure mathematics'. A typical feature of a combinatorial optimization problem is the fact that the set of feasible solutions (permutations of the elements of a finite set, 0-1 vectors or matrices, subgraphs of a graph, etc.) is finite. A critical argument usually went along the following lines. 'Mathematics, as we know it, is concerned with two major issues: given a problem, determine whether the problem has a solution, and if yes, how to find it. In combinatorial optimization, to answer these questions is nothing but trivial.

Consider, for example, a three-machine flow shop problem to minimize the makespan. Does it have a solution? It sure does; some permutation of the jobs will deliver it. How to find it? Easy; just compare all permutations, since their number is finite. Therefore, combinatorial optimization and its branches, including scheduling, cannot be regarded as a mathematical discipline'.

Finite the set of feasible solutions may be, but even for scheduling problems of modest size (for example with 50 jobs), full enumeration of all feasible solutions even on the fastest computers is impossible within a reasonable time (often impossible within the period from the Big Bang until now). Thus, not every method of solving a problem should be seen as suitable, and the issue of finding a solution should be looked at from a different perspective: we need a method that will find a solution fast. But in turn, this poses new questions such as what is meant by fast: 'within one hour; on your computer, on my computer, by hand, or with pen and paper; are you really sure that you at all can find an algorithm that will run fast enough; or if you cannot find a fast algorithm, how do you know that somebody clever could not do it either?' During the early days in the 1950s and 1960s, researchers were asking these and similar questions, to themselves, to friends, and to rivals, but only in 1965 everybody agreed what should be called a fast, or put simply, a good solution algorithm.

Edmonds (1965) in his seminal paper on the matching problem, argues that a 'good' algorithm is one whose running time depends polynomially on the length of the input (or size) of the problem. Since a computer uses the binary representation of the numbers, the length L of the input is essentially bounded by the product of the number of input parameters and the maximum length of the binary number corresponding to any of the input parameters. For example, the length of input for the problem of minimizing the weighted sum of completion times on m unrelated machines is bounded by $L = nm \log(\max p_{ij}) + n \log(\max w_j)$. An algorithm that requires $O(L^k)$ time, where k is a constant that does not depend on L , is called a *polynomial-time* (or simply a *polynomial*) algorithm.

Thus, what we should be attempting to find are polynomial-time algorithms. Since sorting n numbers takes $O(n \log n)$ time, the algorithms of Johnson (1954), Jackson (1955, 1956), Smith (1956) and Moore (1968) each require $O(n \log n)$ time each (see Section 2).

Some dynamic programming algorithms are also polynomial, while others strictly speaking are not. Consider, for example, the algorithm of Lawler and Moore (1969) for solving the problem of minimizing the weighted number of late jobs on a single machine (see (2) in Section 3.4). The algorithm requires $O(nT)$ time, where $T = \min\{\max_{j=1, \dots, n} d_j, \sum_{j=1}^n p_j\}$. If we assume a binary representation of the input parameters, the algorithm requires time that is exponential with respect to the size of the problem. However, assuming the unary representation under which an

integer k is encoded as k bits (so that 5 becomes encoded as 11111), the running time of $O(nT)$ should be regarded as polynomial. Those algorithms that require polynomial time under the unary encoding are called *pseudopolynomial*; they are of some interest too, but less practical than polynomial algorithms and their behaviour strongly depends on the size of input parameters.

What can be said about branch and bound methods? Unfortunately, for most of them, it is possible to exhibit an instance of the problem for which the behaviour of the method is not much better than full enumeration.

A long-standing open question concerned linear programming, which is one of the main techniques within the operational research toolkit. Although the well-known simplex method is quite computationally efficient in practice, it remains an open question whether it can be implemented to run in polynomial time. The first polynomial-time algorithm for linear programming was due to Khachiyan (1979); an alternative method was subsequently developed by Karmarkar (1984). The fact that linear programming problems are polynomially solvable has given rise to a number of algorithms for finding exact and approximate solutions to scheduling problems.

Now the goal within scheduling and combinatorial optimization was clear: search for polynomial-time algorithms. Unfortunately, researchers found that for most interesting problems (from both theoretical and practical viewpoints), they were not able to find such algorithms. The general feeling of the community was that most of these problems had some ‘built-in’ difficulty and consequently would not admit solution in polynomial time.

And finally that feeling of difficulty found a solid justification. The paper of Cook (1971) delivered the message: yes, some problems are ‘easy’, that is, polynomially solvable, while some are ‘hard’ and for those the existence of polynomial-time algorithms is unlikely. It is beyond the scope of this paper to discuss the main statement of Cook (1971), which was formulated and proved in terms of language recognition on deterministic and non-deterministic Turing machines. For the operational research community, Karp (1972) translated Cook’s discovery into more accessible terms and added the first entries into the list of computationally hard problems.

Without going into details, the findings of computational complexity theory can be informally summarized as follows. Standard or deterministic algorithms perform computational instructions one by one in sequence. However, non-deterministic algorithms for each instruction make a ‘good guess’ which of the possible instructions to perform next. Problems that can be solved in polynomial-time by deterministic algorithms form the class \mathcal{P} ; problems that can be solved in polynomial time by non-deterministic algorithms form the class \mathcal{NP} . It is not known whether these two classes coincide, but a widely accepted conjecture assumes that $\mathcal{P} \neq \mathcal{NP}$. There are problems that are no easier than any other problem in \mathcal{NP} . These problems are called *\mathcal{NP} -hard*.

Further, if one of these problems admits a polynomial-time algorithm, then all of them are polynomially solvable, while if we can prove that for one of them there is no polynomial-time algorithm, then none of them can be solved in polynomial time. If a problem is proved \mathcal{NP} -hard, this means that it is no easier than any other hard problem, and it is assumed that the existence of a polynomial-time algorithm for its solution is highly unlikely.

To prove that a decision problem P is \mathcal{NP} -hard (a decision problem is one with an answer ‘Yes’ or ‘No’ such as finding whether there exists a schedule with a value of the objective function no greater than some given constant), we need to take an arbitrary instance of a certain problem Q , which is known to be \mathcal{NP} -hard, and transform it in polynomial time to a specific instance of problem P which has a solution if and only if problem Q has a solution. Informally, to solve the constructed instance of problem P , one has to be able to solve an \mathcal{NP} -hard problem Q , and therefore problem P is no easier than problem Q .

Cook (1971) proved that a certain problem in mathematical logic (3-SATISFIABILITY) is \mathcal{NP} -hard by definition, that is, no easier than any problem in \mathcal{NP} . Using that fact, Karp (1972) proved the \mathcal{NP} -hardness of about 20 problems that arise within operational research. A burst of papers followed; researchers within all branches of combinatorial optimization reported that many problems with the reputation of being challenging (TSP, knapsack, graph colouring, etc) were indeed actually hard to solve.

Soon after, it was observed that some problems were \mathcal{NP} -hard with respect to the standard binary encoding but pseudopolynomially solvable. These problems got the name of *\mathcal{NP} -hard in the ordinary sense* or *binary \mathcal{NP} -hard*. Those problems which remain \mathcal{NP} -hard under the unary encoding were named *\mathcal{NP} -hard in the strong sense* or *unary \mathcal{NP} -hard*.

There were extensive developments in computational complexity throughout the third decade. Several years after the ground-breaking papers by Cook and Karp had been published, the monograph by Garey and Johnson (1979) appeared, which still remains the classical text on the topic. The book not only demonstrates various techniques for proving \mathcal{NP} -hardness, but is accompanied by a detailed description of complexity issues for more than 300 problems from 12 problem areas, including scheduling. The list of \mathcal{NP} -hard problems was permanently updated; the main outlet for that had been the ‘ \mathcal{NP} -Completeness Column’ run by David Johnson in the *Journal of Algorithms*.

4.2. Classification scheme for scheduling

Scheduling with its variety of models was in the forefront of the burst of complexity studies aimed at establishing a clear borderline between the easy and hard problems. For example, a certain problem on a single machine is proved \mathcal{NP} -hard. Then there is no need to study the complexity of the same

problem with parallel machines, or in flow shop environment, or with release dates, or with precedence constraints: none of these additional features will not make the original problem any easier. That observation quite naturally led to the three-field classification scheme for scheduling problems.

The first paper that systematically studies complexity issues for scheduling problems and gives their classification is that of Lenstra *et al* (1977). The issues of polynomial reducibility between various scheduling models are introduced, and numerous proofs of \mathcal{NP} -hardness based on standard \mathcal{NP} -hard problems (PARTITION, 3-PARTITION, CLIQUE, HAMILTONIAN CYCLE, etc) are given. Building on the earlier efforts of Conway *et al* (1967), the paper also introduces the classification scheme for scheduling problem, which uses four fields, with one field indicating the number of jobs.

One of most important works in scheduling is the survey by Graham *et al* (1979) which summarized the development in scheduling up to the time of publication and still remains an excellent reference. It is here that the three-field classification scheme $\alpha|\beta|\gamma$ first appeared in its final form. The field α is used to describe the machine environment, β provides any processing conditions specific to the problem, and γ states the objective function to be minimized. For example, $1|r_j, prec| \sum w_j C_j$ represents the single machine problem of minimizing the sum of weighted completion times with job release dates and precedence constraints.

The advantage of the classification scheme introduced by Graham *et al* (1979) is not only that it provides a fast way of formulating scheduling problems; once the $\alpha|\beta|\gamma$ name of a problem is identified, it is often fairly easy to establish its complexity status ('easy', 'hard' or 'open') by comparing it with what has been known about problems in its immediate neighbourhood. Of course, all scheduling researchers are fluent in this language, which E.L. Lawler called 'Scheduleese' (see Lenstra, 1998). Describing the way he communicated with his collaborators, Lawler recalls 'Soon our technical conversations were laced with shorthand. We know the status of one-deejay-sum-ceedjay and one-arejay-sum-ceedjay, I would say, but what about one-preemption-arejay-deejay-sum-ceed-jay?' (see Lawler, 1991).

For each model, we can track how its complexity is affected by various assumptions and additional restrictions. Let us start with the problem which in 'Scheduleese' is called $1|| \sum w_j C_j$, a single machine problem to minimize the sum of weighted completion times, which we know is solvable in $O(n \log n)$ time due to Smith (1956). Will the problem remain easily solvable if each job j has an individual release date r_j ? The answer is 'Unlikely', since the resulting problem is \mathcal{NP} -hard in the strong sense, even if all weights w_j are equal (see Lenstra *et al*, 1977). Will the latter problem remain hard, if we allow preemption in job processing? No, with preemption allowed, we can find an optimal schedule in $O(n^2)$ time due to an algorithm of Schrage (1968). However, that algorithm only operates if the job weights are equal,

while problem $1|r_j, pmtn| \sum w_j C_j$ is shown by Labetoulle *et al* (1984) to be \mathcal{NP} -hard in the strong sense.

Now consider problem $1|| \sum w_j C_j$, and assume that there is a precedence relation \rightarrow imposed on the set of the jobs such that $i \rightarrow j$ means that job i must precede job j in any schedule; more specifically, job j can only start if job i is completed. The complexity of the resulting problem depends on the structure of the precedence constraints: as proved by Lawler (1978), the problem can be solved in $O(n \log n)$ time if the relation is represented by a series-parallel graph, while under arbitrary precedence constraints the problem is shown by Lenstra and Rinnooy Kan (1978) to be \mathcal{NP} -hard in the strong sense.

Again we start with problem $1|| \sum w_j C_j$, and this time let us increase the number of machines. Bruno *et al* (1974) show that the problem with two identical parallel machines is \mathcal{NP} -hard in the ordinary sense, while if the weights are equal there is a polynomial time algorithm to minimize $\sum C_j$ on any number of unrelated parallel machines. The two-machine flow shop to minimize $\sum C_j$ is \mathcal{NP} -hard in the strong sense, as proved by Garey *et al* (1976).

Given information on the computational complexity of basic scheduling problems, the process of drawing conclusions on the complexity status of a certain extended problem can be automated. Lageweg *et al* (1982) report on a computer tool for determining the complexity of scheduling problems. Given a problem, the system gives a fairly full description of the complexity of related problems, including the 'minimum hard', 'maximum easy' and open problems. Similar information can also be found at the website <http://www.mathematik.uni-osnabrueck.de/research/OR/class/> initiated by Peter Brucker and maintained by Sigrid Knust, University of Osnabrück, Germany.

Although there were numerous pieces of evidence that preemption, if allowed, could simplify the problem, still a solid conclusion that a preemptive version of a problem is no harder than the non-preemptive one could not be drawn. A 'half counter-example' remained: the problem of minimizing the sum of completion times on a variable number of unrelated parallel machines is polynomially solvable, while its preemptive counterpart remained open, even for two machines. The complexity status of the preemptive problem was resolved much later (and to everybody's surprise); for further details, we refer to Section 6.6.

4.3. Open shop scheduling

This decade saw the arrival of a new multi-stage scheduling system, the *open shop*. For this model, each job j has to be processed on each machine i for p_{ij} time units; but, unlike for the flow shop and job shop, the order of a job's operations is 'open', that is, not fixed in advance, and has to be chosen with different jobs being allowed different processing routes. The term 'open shop' was coined in the paper of Gonzalez and Sahni (1976), who gave a fairly complete analysis of

the complexity of the problem of minimizing the makespan. However, several years earlier, De Werra (1970) presented a problem that can be classified as the preemptive open shop problem and reduced it to one of finding the optimal edge coloring in a bipartite multigraph.

In all open shop problems reviewed in this section, the objective is to minimize the makespan C_{\max} . It is clear that in any schedule the makespan can be less than neither the total load of any machine nor the total duration of any job, so that a lower bound on the makespan is

$$\text{LB} = \max \left\{ \max_{i=1, \dots, m} \sum_{j=1}^n p_{ij}, \max_{j=1, \dots, n} \sum_{i=1}^m p_{ij} \right\}. \quad (3)$$

The two-machine open shop problem can be solved in linear time for $m = 2$, with the optimal value of C_{\max} achieving the lower bound (3). An approach by Gonzalez and Sahni (1976) is motivated by the similarity of the flow shop and open shop models: a special flow shop schedule is created and the route of one of the jobs is changed. An alternative algorithm due to Pinedo and Schrage (1982) combines greedy scheduling with a special arrangement regarding the longest operation. Yet another algorithm by de Werra (1989) creates a schedule that organizes the jobs into three blocks on each machine and schedules them to avoid any block overlap.

In terms of extended versions of the non-preemptive open shop problem, the lower bound (3) is no longer tight, and the three-machine problem is \mathcal{NP} -hard in the ordinary sense, as is the problem with four machines and at most two operations per job (see Gonzalez and Sahni, 1976). If the number of machines is variable, the problem is \mathcal{NP} -hard in the strong sense, as reported in Graham *et al* (1979) with reference to an unpublished manuscript of J.K. Lenstra. From the complexity point of view, it is still unknown whether: (i) the three-machine problem is \mathcal{NP} -hard in the strong sense; and (ii) whether the problem with three machines and two operations per job can be solved in polynomial time.

For the open shop, allowing preemption means that the processing of any operation on any machine can be interrupted at any time and resumed later from the point of interruption, and in between the interruption and the resumption of an operation any other operation of the same job can be (preemptively) processed. It is clear that allowing preemption creates no advantage for the two-machine case. For an arbitrary number of machines, a preemptive schedule exists with makespan that achieves the lower bound (3), and such a schedule can be found in polynomial time. One of the approaches to finding an optimal preemptive open shop schedule is based on the so-called Birkhoff–von Neumann theorem on decomposition of double stochastic matrices (see Gonzalez and Sahni, 1976; Lawler and Labetoulle, 1978). Another way to solve the preemptive problem is to reduce it to the optimal edge colouring of a bipartite multigraph (see de Werra, 1970; Gabow and Kariv, 1982).

The fact that the preemptive open shop problem is polynomially solvable has given rise to a number of two-stage algorithms for preemptive scheduling in other machine environments. Typically, in the first stage of such an algorithm, a linear programming problem is solved to determine the total duration of each operation on each machine, and then in the second stage a preemptive open shop schedule is found which essentially delivers an optimal solution to the original problem. As examples of problems handled by this approach, we mention the preemptive problem of minimizing the makespan on unrelated parallel machines as solved by Lawler and Labetoulle (1978), and the preemptive multi-processor (or hybrid) open shop problem with parallel machines in each processing stage as solved by Lawler *et al* (1982b).

4.4. Mathematical programming approaches

Many techniques and approaches within combinatorial optimization are applied to the TSP at an early stage of their development. Success of a technique in solving the TSP would often inspire researchers to apply the same approach to other problems such as scheduling. Various techniques from mathematical programming fall within this category.

A *polyhedral approach* starts with an integer linear programming formulation of the problem and then introduces valid inequalities with a view to obtaining a tighter relaxation. The resulting lower bound is then used within a branch and bound or branch and cut framework. This type of approach was used for the TSP by several researchers such as Miliotis (1976, 1978), Crowder and Padberg (1980), Grötschel (1980) and Padberg and Hong (1980) who obtained optimal solutions to instances with up to 100 or more cities. Surprisingly, polyhedral methods have been notably unsuccessful for obtaining optimal solutions to machine scheduling problems, with branch and bound algorithms based on combinatorial approaches usually performing better. Owing to lack of computational success, the algorithms are typically not reported in the literature. However, some interesting polyhedral theory has been developed that aids our understanding of scheduling problems (eg, see Queyranne and Wang, 1991).

On a more positive note, the use of *Lagrangean relaxation* has been very useful in obtaining lower bounds for use in branch and bound algorithms for scheduling problems. It was originally developed for the TSP by Held and Karp (1971). The key idea is that complex problems are often simple problems with a few added constraints. By performing a Lagrangean relaxation in which the complicating constraints are included in the objective function using a suitably chosen Lagrange multiplier, the resulting problem may be simple to solve and thereby provide a lower bound. Having chosen which constraints to relax, the main issue is often how to find values of the multipliers. Although a general-purpose iterative technique known as *subgradient optimization* (see Held *et al*, 1974) is available for finding values of the multipliers, it is computationally time consuming.

One of the first studies to use Lagrangean relaxation in scheduling is that of Fisher (1976) who considers the problem of scheduling a single machine to minimize the total tardiness. He performs a Lagrangean relaxation of the machine capacity constraints that restrict the machine to processing exactly one job in each unit time interval throughout the period of processing. In the relaxed problem, the machine can process several jobs simultaneously, and the value of each Lagrange multiplier can be viewed as the price for using the machine in that interval. Using precedence constraints on the jobs that are determined from the dominance rules of Emmons (1969), tight lower bounds are obtained. Lagrange multiplier values are computed using subgradient optimization. Even though the lower bound requires pseudopolynomial time ($O(nP)$ time, where P is the sum of the processing times), the algorithm successfully solves instances with up to 50 jobs.

There was also interest in using Lagrangean relaxation to obtain quickly computed bounds. This meant avoiding the computationally demanding subgradient procedure for obtaining values of Lagrange multipliers. A *multiplier adjustment method* is developed in the PhD thesis of Van Wassenhove (1979) that allows the efficient computation of Lagrange multiplier values. For example, Potts and Van Wassenhove (1983) consider single machine scheduling to minimize the total weighted completion time $\sum w_j C_j$ with constraints $C_j \leq \bar{d}_j$ for each job j , where \bar{d}_j is the deadline of job j . After performing a Lagrangean relaxation of the deadline constraints, the resulting Lagrangian problem is to minimize $\sum (w_j + \lambda_j) C_j - \sum \lambda_j \bar{d}_j$, where λ_j is the multiplier associated with the constraint $C_j \leq \bar{d}_j$. The weights associated with the completion times become $w_j + \lambda_j$ under this relaxation. The multiplier adjustment method fixes the optimal sequence of jobs in the Lagrangean problem (usually by applying a heuristic method), and then computes the values of the multipliers to maximize the lower bound subject to the given sequence being an optimal solution of the Lagrangean problem. A similar multiplier adjustment method was also applied by Hariri and Potts (1983) to the corresponding problem with release date constraints instead of deadline constraints, and by Potts and Van Wassenhove (1985) to the problems of scheduling a single machine to minimize the total weighted tardiness.

4.5. Other results

In terms of branch and bound, a breakthrough was made by Carlier (1982). He was the first to develop a branch and bound algorithm that could routinely solve instances with several hundred jobs. Specifically, Carlier proposes a branch and bound algorithm for the problem of scheduling jobs with release dates on a single machine to minimize the maximum lateness of the jobs. Recall from Section 3.3 that this problem provides lower bounds for more general problems, such as the job shop, since the due dates are mathematically equivalent to the delivery times. The key to the success of

Carlier's algorithm is the novel branching rule. He defines a job with the property that this job either is processed before all jobs of a certain subset or is processed after all jobs of this subset, and thereby created a binary branching procedure.

Another elegant result is due to Lawler (1977), who demonstrates that the single machine total tardiness scheduling problem is solvable in pseudopolynomial time (although it was not known at the time whether it was \mathcal{NP} -hard). Lawler's key contribution is to show that the problem decomposes: those jobs with due dates less than or equal to some due date d_k are processed before the longest job and those jobs with due dates greater than d_k are processed after this longest job. This type of decomposition is extremely rare in scheduling, and intricate analysis is needed to establish that the decomposition property holds. Lawler designs a dynamic programming algorithm based on his decomposition principle. Later, Potts and Van Wassenhove (1982) exploit this decomposition method to produce a practical method of solving instances with up to 100 jobs.

During this decade, progress was achieved in solving preemptive problems on parallel machines in the presence of release and due dates. For identical parallel machines, a network flow approach by Horn (1974) creates a test of whether there exists a feasible schedule in which each job j is processed within the time interval $[r_j, d_j]$. Using binary search, this algorithm is converted by Labetoulle *et al* (1984) into a polynomial algorithm for minimizing the maximum lateness L_{\max} . To solve the latter problem on uniform machines, Martel (1982) suggests computing the maximum polymatroidal flow. A more efficient algorithm by Federgruen and Groenvelt (1986) uses a classical maximum flow approach and requires $O(n^3s)$ time, where s is the number of distinct machine speeds.

Fully polynomial-time approximation schemes for problems of scheduling to minimize the makespan on a fixed number of (not necessarily identical) parallel machines were designed by Sahni (1976) within this decade. However, we defer the discussion of these results to Section 5.1.1 which is devoted to approximation algorithms for single and parallel machine scheduling.

4.6. Scheduling at the end of Decade 3

By the end of this decade, scheduling problems could be described by the three-field classification scheme, and a natural first step in researching a problem was to establish its complexity status (polynomially solvable or \mathcal{NP} -hard). Scheduling had suddenly become a much more structured field than had been the case a decade earlier.

The discovery of computational complexity and its consequences for scheduling captured in the survey by Graham *et al* (1979) made a great impact on further development in the area. Several other surveys were issued later with the same core team of co-authors. Here, we only mention the reviews

of Lawler *et al* (1982a, 1993), which were followed by an annotated bibliography of Hoogeveen *et al* (1997).

As a result of the developments during this decade, many young researchers joined the field and many established researchers changed direction to work in scheduling theory. The net result was that scheduling as a research topic became much more mature with many more techniques and concepts from other research fields being applied. Computational testing of algorithms was also performed more rigorously, and branch and bound algorithms started to reach a level of sophistication that they could solve instances of a practical size.

This decade saw the first major conference devoted to scheduling. Sponsored by NATO, an Advanced Study and Research Institute on Theoretical Approaches to Scheduling Problems was held in Collingwood College, University of Durham, UK, during the period 6–17 July 1981. It was organized by Michael Dempster, Jan Karel Lenstra and Alexander Rinnooy Kan, and attended by 91 participants from 15 countries. Well-known researchers who attended the Institute and are still active in scheduling today include Jacques Carlier, Kevin Glazebrook, Rolf Möhring, Mike Pinedo, Chris Potts, Leen Stougie and Gideon Weiss. Some of the more significant presentations at the Institute appeared in a book (see Dempster *et al*, 1982).

A fairly complete exposition of known scheduling results, mainly on the computational complexity of scheduling problems, was given in a two-volume research monograph written in Russian by Tanaev *et al* (1984) on single machine and parallel machine models and by Tanaev *et al* (1989) on shop scheduling models. Chapter 3 of the book by Tanaev *et al* (1984) is based on the research conducted by Yakov Shafransky in the 1970s on the theory of minimizing of so-called priority-generating functions over partially ordered sets of jobs. This is related to the contributions of Monma and Sidney (1979) in the area of scheduling with precedence constraints.

5. Decade 4: approximate solutions

The fact that most scheduling problems of interest are \mathcal{NP} -hard delivers an informal message: it is unlikely that any of these problems allow an exact optimal solution to be found in polynomial time. Thus, if we need to find the exact optimum, we should be prepared to accept that finding such a solution will require a considerable computation time. On the other hand, for most practical needs it is sufficient to adopt an approximate solution that is anticipated to be relatively close to the optimum.

Traditionally, among the algorithms that deliver approximate solutions to \mathcal{NP} -hard problems, we distinguish between *approximation* algorithms and *heuristic* algorithms (usually abbreviated to heuristics).

The performance of an approximation algorithm is evaluated by either worst-case analysis or probabilistic analysis of

its behaviour. Below we mainly concentrate on most important results regarding worst-case analysis of scheduling approximation algorithms.

Under our terminology, there is no associated analytic evaluation of how well heuristics perform. Heuristics include, but are not limited to, constructive heuristics, local search heuristics, more elaborated metaheuristics and hyperheuristics, etc. Typically, the quality of a heuristic algorithm is judged by its performance on randomly generated and/or benchmark instances, and normally we do not know how well a heuristic will behave on a particular instance. Our discussions of heuristics will focus on local search since this is often the method of choice when tackling practical scheduling problems with the associated complications that occur in real life.

5.1. Approximation algorithms

We start by introducing the terminology and main concepts that are used in the worst-case analysis of approximation algorithms. For a scheduling problem with the goal of minimizing a function $F(S)$ over all feasible schedules S , let S^* denote an optimal schedule. A polynomial-time algorithm that finds a feasible solution S_H such that $F(S_H)$ is at most ρ (where $\rho \geq 1$) times the optimal value $F(S^*)$ is called a ρ -approximation algorithm; the value of ρ is called a *worst-case ratio bound*. If a problem admits a ρ -approximation algorithm, then it is said to be *approximable within a factor ρ* . A family of ρ -approximation algorithms is called a *polynomial-time approximation scheme*, or a *PTAS*, if $\rho = 1 + \varepsilon$ for any fixed $\varepsilon > 0$; if additionally the running time is polynomial with respect to both the length of the problem input and $1/\varepsilon$, then the scheme is called a *fully polynomial-time approximation scheme*, or an *FPTAS*.

Before proceeding, it is useful to refer to an excellent review of Schuurman and Woeginger (1999) on approximation algorithms for scheduling. They include a list of 10 major open problems related to approximability in scheduling. As far as we are aware, all of them are still open.

5.1.1. Single machine and parallel machines Worst-case analysis of scheduling approximation algorithms was originated by Graham (1966, 1969). The problem that he addressed was that of minimizing the makespan on m identical parallel machines. The first result concerns the performance of a list scheduling algorithm (see Section 2.5 for a discussion of list scheduling with the SPT list for minimizing the sum of completion times on identical parallel machines, as presented in the book of Conway *et al*, 1967). Suppose that a schedule S_L is found by a list scheduling procedure with an arbitrary list, and some job k is the last job completed in that schedule, that is, formally, $C_{\max}(S_L) = C_k$. Suppose that job k starts at time t . The ‘greedy’ nature of list scheduling implies that all machines are busy until time t ; otherwise job k would have been assigned to start earlier on an available machine.

Noticing that

$$t \leq \left(\sum_{j \in N} p_j - p_k \right) / m,$$

we obtain

$$\begin{aligned} C_{\max}(S_L) &= t + p_k \leq \left(\sum_{j \in N} p_j - p_k \right) / m + p_k \\ &= \frac{1}{m} \sum_{j \in N} p_j + \frac{m-1}{m} p_k. \end{aligned}$$

Using the lower bound (1) on the optimal makespan, we finally derive

$$\begin{aligned} C_{\max}(S_L) &= \text{LB} + \frac{m-1}{m} \text{LB} = \left(2 - \frac{1}{m} \right) \text{LB} \\ &\leq \left(2 - \frac{1}{m} \right) C_{\max}(S^*), \end{aligned}$$

that is, for the problem under consideration the list scheduling algorithm is a $(2 - 1/m)$ -approximation algorithm. Notice that a worst-case ratio of $2 - 1/m$ is guaranteed for any sequence of jobs in the list and for any instance of the problem. Can the analysis be improved so that a smaller ratio can be achieved for this algorithm? To dismiss these doubts, Graham exhibits a tightness example with $m^2 - m + 1$ jobs, which includes $m(m - 1)$ ‘small’ jobs of unit length each and one ‘big’ job with a processing time of m time units. In an optimal schedule S^* , the big job is assigned to one of the machines, while each of the remaining $m - 1$ machines processes m small jobs, so that $C_{\max}(S^*) = m$. On the other hand, if the jobs are organized into a list with the big job in the last position, the list scheduling algorithm will assign exactly $m - 1$ jobs to each of the m machines and then at time $m - 1$ starts the big job on one of the machines, so that $C_{\max}(S_L) = 2m - 1$. This implies that for the list scheduling algorithm, a worst-case ratio of $2 - 1/m$ represents a tight bound.

The tightness example shows that it is not wise to delay the processing of longer jobs. This delay can be avoided by arranging the list in the LPT order, that is, in non-increasing order of the processing times. Graham (1969) shows that the LPT list scheduling algorithm delivers a schedule S_{LPT} such that $C_{\max}(S_{LPT})/C_{\max}(S^*) \leq 4/3 - 1/(3m)$, and this bound is tight.

Moreover, Graham (1969) develops an algorithm that first schedules k longest jobs optimally, and then assigns the remaining jobs to machines by list scheduling. The worst-case ratio bound for this algorithm is $1 + (1 - 1/m)/(1 + \lfloor k/m \rfloor)$. For fixed m , such an algorithm requires $O(n^{km})$ time; therefore, a family of these algorithms applied to different values of k can be regarded as a PTAS, although its running time is impractical.

One of the first fully polynomial-time approximation schemes in combinatorial optimization is due to Ibarra and

Kim (1975), who apply the rounding technique to convert dynamic programming algorithms for the knapsack and subset sum problems into FPTASs. Recall that in the subset sum problem there are given items with weights, and it is required to find a subset of items having maximum total weight that does not exceed a given value. This problem is closely related to the scheduling problem of minimizing the makespan on two parallel identical machines, so the results of Ibarra and Kim establish that the latter problem admits an FPTAS. Sahni (1976) demonstrates that for the problem of minimizing the makespan on m identical machines, an FPTAS exists for any fixed m , while Horowitz and Sahni (1976) extend this result to uniform and unrelated machines.

The problem of minimizing the makespan on parallel machines can be considered as dual with respect to a well-known *bin-packing* problem in which items of various sizes are to be packed into a minimum number of identical bins of a given capacity. Hochbaum and Shmoys (1987) interpret each of the identical machines as bins of capacity d , and for any ρ , where $\rho > 1$, define a ρ -dual approximation algorithm that finds an assignment of jobs to the minimum number of machines so that the last job on each machine completes no later than time ρd . This leads to a PTAS for the original scheduling problem with a variable number of machines. An extension of this result to uniform machines is provided by Hochbaum and Shmoys (1988).

For the problem of minimizing the makespan on m unrelated parallel machines, several algorithms use the idea of rounding of the solution of the linear relaxation of the corresponding integer programming problem. Let variable x_{ij} be defined by

$$x_{ij} = \begin{cases} 1 & \text{if job } j \text{ is assigned to be processed on machine } i, \\ 0 & \text{otherwise.} \end{cases}$$

The optimal makespan is equal to the smallest value of the objective function of the integer program:

$$\begin{aligned} &\text{Minimize } C \\ &\text{subject to } \sum_{j=1}^n p_{ij} x_{ij} \leq C, \quad i = 1, \dots, m, \\ &\sum_{i=1}^m x_{ij} = 1, \quad j = 1, \dots, n, \\ &x_{ij} \in \{0, 1\}, \quad i = 1, \dots, m, j = 1, \dots, n. \end{aligned}$$

The linear relaxation is obtained by replacing the integrality constraints by $0 \leq x_{ij} \leq 1$. The resulting linear programming problem can be solved in polynomial time. Potts (1985) demonstrates that the number of fractional variables in a basic optimal solution is linear in m , and he derives a 2-approximation algorithm for any fixed m . For $m = 2$, there is only one fractional variable and Potts (1985) offers an improved $(3/2)$ -approximation algorithm. As shown by Lenstra *et al* (1990), a fractional solution to the linear

relaxation can be rounded in polynomial time to yield a 2-approximation algorithm for variable m . Additionally, Lenstra *et al* prove that finding a schedule S for this problem with $C_{\max}(S) \leq 2$ is \mathcal{NP} -hard. This means that, unless $\mathcal{P} = \mathcal{NP}$, it is impossible to derive a polynomial time approximation algorithm with a worst-case ratio that is strictly less than $3/2$.

Approximability issues of various scheduling problems on a single machine and parallel machines to minimize the sum of weighted completion times $\sum w_j C_j$ became a topic of considerable interest in the mid-1990s. It is remarkable how many new algorithmic ideas were generated and how fast the area was advanced from being almost completely unstudied to almost completely resolved. Here, we only mention several most influential contributions. The paper of Hall *et al* (1997) reports on the first successful attempts to design constant-ratio approximation algorithms for problems of minimizing $\sum w_j C_j$, including models with release dates and/or precedence constraints. The algorithms rely on solving linear programming relaxations (in particular, interval-indexed formulations). The paper by Afrati *et al* (1999) summarizes the efforts of 11 researchers in designing PTASs for several versions of the problem of scheduling jobs with individual release dates to minimize $\sum w_j C_j$ on a single machine, on a variable number of identical parallel machines and on a fixed number of unrelated parallel machines. Novel algorithmic ideas employed in this work included (i) geometric rounding (given an $\varepsilon > 0$, the processing times and release dates are rounded to integral powers of $(1 + \varepsilon)$ thereby breaking the time horizon into geometrically increasing intervals); (ii) time stretching (the insertion of small idle periods which allows the creation of a good schedule by enumerating a limited number of jobs to be scheduled last on a machine); and (iii) weight-shifting (provided that many jobs are released simultaneously, some of them are delayed to be processed in later intervals, so that each job can be scheduled within $O(1/\varepsilon^2)$ intervals from its release time). Skutella (2001) combines convex quadratic relaxations and semidefinite programming relaxations with randomized rounding to design improved constant-ratio approximation algorithms for minimizing $\sum w_j C_j$ on unrelated parallel machines, including the case in which jobs have machine-dependent release dates.

5.1.2. Shop scheduling We now consider approximation algorithms for the flow shop, open shop and job shop problems. Unless stated otherwise, we focus on the minimization of makespan.

Gonzalez and Sahni (1978) suggested searching for an approximate solution in the class of so-called *busy* schedules, where at any time at least one machine is processing some operation. This class obviously contains an optimal schedule, but, on the other hand, contains very bad schedules, including those in which exactly one machine is busy at a time, so that the makespan is equal to the sum of the processing times of all operations. This means that for a busy flow shop schedule

S_B the bound $C_{\max}(S_B)/C_{\max}(S^*) \leq m$ holds. A natural idea of designing an approximation algorithm is to replace the original problem by an artificial two-machine flow shop, and to convert the solution to that artificial problem found by Johnson's algorithm into an approximate solution to the original problem. However, no algorithm of this type is known to deliver a ratio better than $\lceil m/2 \rceil$ (see Gonzalez and Sahni, 1978). Notice that for $m = 3$ we have that $\lceil m/2 \rceil = 2$, and it is very easy to design a 2-approximation algorithm: schedule the first two machines optimally by Johnson's algorithm and concatenate the resulting schedule with a block of operations on the third machine. For $m = 3$, an improved $(5/3)$ -approximation algorithm is due to Chen *et al* (1996). The best known (and essentially best possible) approximation result for the flow shop with a fixed number of machines is a PTAS developed by Hall (1998), which also can be extended to handle job release dates. Still, it remains unknown whether the problem with a variable number of machines is approximable within a constant factor, that is, whether there exists a polynomial-time algorithm that finds a schedule S such that $C_{\max}(S)/C_{\max}(S^*) \leq \rho$ for some constant ρ that does not depend on m .

For the m -machine flow shop problem to minimize the makespan, there is an interesting link with a so-called Steinitz Lemma on compact vector summation. Instead of citing the original papers containing early studies on the application of this approach to scheduling (most of which are in Russian and Hungarian), we refer to two papers of Sevastianov (1994, 1995), one of which presents a detailed survey. One of the implications of this approach is that for the m -machine flow problem, there exists a permutation schedule S with $C_{\max}(S) \leq \Pi_{\max} + \phi(m)p_{\max}$, where Π_{\max} is the maximum load over all the machines, p_{\max} is the duration of the longest operation and $\phi(m) \leq m(m-1)$ is a function that depends only on m . Moreover, finding such a permutation schedule takes at most $O(n^2 m^2)$ time. For $m = 3$, it can be proved that $\phi(3) = 3$ and this value cannot be further reduced.

Unlike its flow shop counterpart, minimizing the makespan in the open shop is approximable within a factor of 2. This is achieved by finding a *dense* schedule in which no machine is idle if there is a job that can be started on that machine. A dense schedule can be found by a greedy algorithm that requires $O(mn \min\{m, n\})$ time. This approach to open shop scheduling is reported by Bárány and Fiala (1982) who attribute it to A. Racsomány. For any dense open shop schedule S , Bárány and Fiala show that $C_{\max}(S) - C_{\max}(S^*) \leq (m-1)p_{\max}$, where as above p_{\max} is the duration of the longest operation. Furthermore, Aksjonov (1988) shows that $C_{\max}(S)/C_{\max}(S^*) < 2$. This bound is not tight; in fact, it is conjectured that for $m \geq 2$ and for any dense open shop schedule S , a tight bound is given by $C_{\max}(S)/C_{\max}(S^*) \leq 2 - 1/m$. Chen and Strusevich (1993) prove this conjecture for $m \leq 3$; the proofs for several other small values of m are known. It remains an open question whether for some fixed $\varepsilon > 0$, the problem with a variable

number of machines admits an approximation algorithm with a worst-case ratio bound of $2 - \varepsilon$.

Sevastianov and Woeginger (1998) suggest the following approach to the open shop problem with a fixed number of machines: the jobs are classified as big, small and tiny; the big jobs are scheduled 'almost' optimally, the tiny jobs are scheduled by the greedy algorithm and the small jobs are appended to the current partial schedule. This approach leads to a PTAS. Kononov and Sviridenko (2002) describe a PTAS for the problem with a fixed number of machines and individual release dates for all operations. On the other hand, for any fixed number of machines m , where $m \geq 3$, it remains unknown whether an FPTAS exists for the open shop problem.

From the non-approximability point of view, Williamson *et al* (1997) establish a lower bound on a worst-case ratio for approximation algorithms for the flow shop and the open shop problems with a variable number of machines and the makespan objective. They show that for any of these problems with integer processing times, there is a polynomial-time algorithm that determines whether a schedule S with $C_{\max}(S) = 3$ exists; by contrast, unless $\mathcal{P} = \mathcal{NP}$, it is impossible to verify in polynomial time whether there exists a schedule S with $C_{\max}(S) = 4$. Thus, the existence of a polynomial-time approximation algorithm with a worst-case ratio strictly less than $5/4$ would imply that $\mathcal{P} = \mathcal{NP}$.

Hoogeveen *et al* (2001) demonstrate that many scheduling problems with a variable number of machines to minimize the sum of the completion times (including the flow shop and the open shop) are Max SNP-hard, and this implies that for any of these problems the existence of a PTAS would imply that $\mathcal{P} = \mathcal{NP}$.

5.2. Local search

The concept of local search dates back to Decade 1 when methods of searching for improved travelling salesman tours based on removing edges from the tour and replacing them with new ones were first introduced. For example, Croes (1958) first proposes the 2-opt move for exchanging two edges so that 'crossings' in a tour can be eliminated, and Lin (1965) extends the approach to 3-opt which allows exchanges of three edges. In scheduling, Nicholson (1967) is among the first to propose the use of local search. For minimizing the makespan in a three-machine flow shop, he uses moves that remove a job from its current position in the sequence and insert it in another position. However, local search methods failed to attract much attention until much later. The reason is probably twofold: computers were not fast enough for an adequate search of the solution space for problem instances of a practical size, and the methods were simplistic so that the development of these methods and the associated computer experimentation may not have been viewed as a worthy research contribution.

The pioneering publications of Kirkpatrick *et al* (1983) and Cerny (1985) that proposed simulated annealing as an

optimization technique were instrumental in establishing local search as a thriving research area. The subsequent introduction of tabu search by Glover (1986, 1989, 1990) provided further stimulation for researchers to contribute in the area of local search. In parallel, the publication of a book by Goldberg (1989) brought genetic algorithms to the attention of researchers in optimization.

The simplest local search algorithm is *descent* in which repeated attempts are made to improve a current solution. The attempts at improvement involve perturbing the current solution, and if the perturbation leads to a better solution a move is made to the new solution. A *neighbourhood* defines which potential moves are allowed. The search usually continues until no further improvement on the current solution can be made in which case a *local minimum* has been reached if the problem is one of minimization. The local minimum might not represent a very good solution. A multi-start descent that executes a descent algorithm several times using a different starting solution for each execution may help to produce better quality solutions. A possibly preferable approach is to adopt an *iterated descent* algorithm in which, upon reaching a local minimum, a modification is made (usually random) to the solution at which stage descent is reapplied.

There are some well-known strategies that allow the search to progress to worse solutions, which may be necessary in the process of traversing the solution space to reach better solutions. In *simulated annealing*, worse solutions are accepted with an *acceptance probability* of $e^{-\Delta/T}$, where Δ is the amount by which the objective function is worse in the new solution and T is a parameter called the *temperature* that is reduced during the course of the algorithm according to a *cooling schedule*. In *tabu search*, a move to the best solution in the neighbourhood is made, even if this solution is worse. To prevent the method from cycling and also to help direct the search into different areas of the solution space, a *tabu list* stores properties of recently visited solutions. Potential moves that lead to solutions with properties stored on the tabu list are forbidden.

By contrast to neighbourhood search algorithms that focus on improving a single solution, *genetic algorithms* aim at improving a population of solutions. Solutions in genetic algorithms are normally represented as strings. To create a new population, pairs of solutions are selected according to their *fitness*, where fitness is normally determined from the objective function value, and are used as parents to create two offspring. A *crossover* operator exchanges sections of the strings of the two parents (with some adjustments being used if valid strings are not obtained) to create two new solutions. A *mutation* operator then randomly changes some elements of the strings of the two solutions thereby creating two offspring. The new population is formed by a combination of solutions from the previous population and the offspring.

Many of the early publications on applications of local search in scheduling used neighbourhoods chosen in a fairly

natural way and followed the generic descriptions of the algorithms closely. The main focus was to evaluate how well local search performed relative to other heuristic methods (usually the conclusion favoured local search in terms of solution quality), and which type of local search algorithm was more effective.

5.2.1. Flow shop The permutation flow shop was used by several researchers as a problem on which to evaluate the potential of local search for generating solutions of superior quality to those obtained from other heuristic methods. Representative of contributions of this type were the studies of Osman and Potts (1989), Ogbu and Smith (1990), Widmer and Hertz (1989) and Taillard (1990) for the permutation flow shop problem with the makespan objective. Osman and Potts (1989) and Ogbu and Smith (1990) both compare the *swap neighbourhood* (in which a new solution is created by interchanging a pair of jobs) and the *insert neighbourhood* (in which a job is removed from its current position in the sequence and inserted in a different position) within a simulated annealing algorithm. Both studies find that the insert neighbourhood provides better solutions. Similar conclusions are reached for tabu search, where Taillard (1990) produces better results with the insert neighbourhood than Widmer and Hertz (1989) with the swap neighbourhood. Genetic algorithms have also studied, the most notable of which is that of Reeves (1995). He represents solutions as sequences and uses the reorder crossover which reorders those elements of one parent that lie between a pair of crossover points according to their order in the other parent.

Owing the fast development of local search, the algorithms proposed by the end of the decade often displayed a variety of features that enhanced their ability to produce high-quality solutions using modest computational resources. An example is the tabu search algorithm of Nowicki and Smutnicki (1996b) that uses a neighbourhood that destroys the critical path in a network representation of the schedule. Specifically, Nowicki and Smutnicki define *blocks* of operations with respect to a critical path in the schedule defined by the current solution, where a block is a maximal set of successive critical operations that require the same machine. The basic neighbourhood comprises insert moves, but only those moves are considered that change the block structure, that is, by inserting the first operation in a block later in the sequence on the relevant machine, by inserting the last operation in a block earlier in the sequence, or by inserting an operation currently between the first and last operation in a block so that it appears either before the first or after the last operation in its block. Another device within the algorithm that helps improve solution quality is a backtracking procedure. When reaching an iteration limit since a best solution is found, the algorithm returns to the best solution found thus far and proceeds with a different neighbourhood move to those performed previously.

5.2.2. Single machine total weighted tardiness problem

There were also various early studies on local search for the single machine total weighted tardiness problem. The first is a descent algorithm proposed by Wilkerson and Irwin (1971) which uses the *transpose neighbourhood* in which two adjacent jobs in the sequence are interchanged (although it is now known that this neighbourhood is too small to be effective for the majority of problems). Matsuo *et al* (1989) use the transpose neighbourhood within a variant of simulated annealing in which the acceptance probability is independent of the solution value. The swap neighbourhood was used in descent and in several implementations of simulated annealing with different cooling schedules within a comparative computational study by Potts and Van Wassenhove (1991). They find that descent performs surprisingly well relative to simulated annealing if moves with the same objective function value are accepted.

Crauwels *et al* (1998) explored the issue of solution representation for the single machine total weighted tardiness problem. In addition to the natural representation of solutions as sequences, they consider a binary representation where each element indicates whether the corresponding job is completed by its due date or is late. A decoding procedure is used to convert this binary representation into a sequence of jobs. Results of a computational study comparing different types of local search algorithms show that tabu search algorithms using both types of representation perform well, as does a genetic algorithm that uses the binary representation of solutions.

5.2.3. Job shop The job shop problem offers a greater challenge than single machine scheduling or the permutation flow shop where a sequence is sufficient to define a solution. Specifically, there is a larger solution space resulting from the need to find a sequence of operations on every machine, and the effect of a modification to one sequence may require this change to be propagated to other sequences if the resulting solution is to be feasible and of good quality. In line with most research contributions, we shall restrict our discussion to the problem of minimizing the makespan in a job shop.

The challenging nature of the job shop problem inspired the development of several novel approaches that can be applied to a variety of problems. Storer *et al* (1992) were among the first researchers to understand the potential that different representations of solutions could have on solution quality. They introduced a *data perturbation* representation in which a search is performed over different perturbations of some parts of the original problem input data, with a specific heuristic used to construct a schedule from the perturbed problem data. They also proposed a *heuristic set* representation in which a search is performed over a set of heuristics to decide which heuristic should be used in the next stage of the schedule construction.

Another important development in job shop scheduling, but being of wider applicability, is that of the *shifting bottleneck* procedure. This procedure was proposed by Adams *et al* (1988), and can be regarded more generally as a decomposition approach for problems with multiple machines. The original shifting bottleneck procedure for job shop scheduling works with the disjunction graph formulation (see Section 3.3), and the observation that a solution can be constructed by selecting each machine in turn and orienting all of the corresponding edges. To orient these edges, a single machine problem with release dates and delivery times is solved (typically using the algorithm of Carlier (1982); see Section 4.5). The so-called bottleneck machine is the next machine to be selected, where the bottleneck machine is defined as the one having the largest objective function value among the solutions of these single machine problems. The shifting bottleneck approach can also be used within a local search framework by performing a search over the order in which the machines are selected for disjunctive edge orientation; for example, see the genetic algorithm of Dorndorf and Pesch (1995) and the guided local search method of Balas and Vazacopoulos (1998).

For the more obvious neighbourhood search algorithms, a promising idea that was adopted by many researchers was to use the block structure of a critical path in a similar way to the approach of Nowicki and Smutnicki (1996b) for the flow shop, thus creating a restricted set of possible neighbourhood moves.

One of the earliest of local search algorithms is the simulated annealing method of Matsuo *et al* (1988). It uses a restricted transpose neighbourhood, and utilizes a look-ahead mechanism before making a decision about accepting or rejecting a move. Other devices for improving the effectiveness of local search include the approximate evaluation of neighbours as used in the tabu search algorithms of Dell'Amico and Trubian (1993) and Taillard (1994), and the use of backtracking as used in the simulated annealing algorithm of Yamada *et al* (1994) and the tabu search algorithms of Barnes and Chambers (1993) and Nowicki and Smutnicki (1996a).

Genetic algorithms employing various representations of solutions have also been the subject of much research. Storer *et al* (1992) use data perturbation and heuristic set representations in genetic algorithms, together with a hybrid representation. Other studies using a heuristic set representation include those of Smith (1992) and Dorndorf and Pesch (1995). Priority representations have also been proposed in which a solution is represented by a priority order of operations for each machine, with a schedule construction procedure used to create the corresponding solution. Davis (1985), Falkenauer and Bouffouix (1991), Yamada and Nakano (1992) and Della Croce *et al* (1995) have designed genetic algorithms based on priority representations.

Among the local search algorithms discussed above, the tabu search algorithm of Nowicki and Smutnicki (1996a)

and the guided local search algorithm employing the shifting bottleneck procedure of Balas and Vazacopoulos (1998) are the most effective.

5.3. Other results

A long-standing open question regarding the complexity status of the problem of minimizing the total (unweighted) tardiness on a single machine was finally resolved by Du and Leung (1990) who prove that the problem is \mathcal{NP} -hard in the ordinary sense by polynomial reduction from the odd–even partition problem. Recall from Section 4.5 that there is a pseudopolynomial algorithm for this problem.

There were significant developments in the design of branch and bound algorithms for the job shop problem, most notably by Carlier and Pinson (1989). They devised tests based on lower and upper bounds for fixing the orientation of disjunctive arcs in the disjunctive graph formulation. Fixing disjunctive arcs allows release dates and delivery times to achieve higher values in the single machine subproblems from which lower bounds are computed. In turn, the resulting tighter lower bounds enable more nodes of the search tree to be pruned. Carlier and Pinson achieved a landmark when their branch and bound algorithm found a solution with a makespan of 930, and verified optimality, for the famous job shop instance of Fisher and Thompson with 10 jobs, 10 machines and 10 operations per job.

Column generation emerged as a successful technique for minimizing total weighted completion time on identical parallel machines through algorithms developed by Chen and Powell (1999) and Van den Akker *et al* (1999). The main idea is to use a set partitioning formulation of the problem in which each binary variable indicates whether or not a single machine schedule is included in the solution of the parallel machine problem. Owing to the large number of potential single machine schedules, only a limited number are considered at the outset. The dual variables for the solution of the linear programming relaxation of the set partitioning formulation allow new solutions to be generated by applying a dynamic programming algorithm.

In the 1990s, scheduling problems to minimize the sum of total (weighted) earliness and total (weighted) tardiness received considerable attention. Here, a job is said to be *early* if it completes by the due date, and the earliness of job j is defined by $E_j = d_j - C_j$. In terms of production scheduling, the penalty for earliness can be understood as a holding cost of a finished product before it is delivered to the customer at its due date, while the penalty for tardiness can be interpreted as a fee paid for a late delivery. In combination, minimizing the earliness–tardiness objective requires that the jobs are completed as close to their due dates as possible; this creates a similarity with just-in-time (JIT) manufacturing. Baker and Scudder (1990) reviewed the main results in this area obtained by 1990, most of which were for single machine

scheduling. Many of the studies are for a version of the problem with a common due date for all jobs. Two versions of the problem are traditionally distinguished: if the total machine load is less than the due date, so that it is possible to schedule all jobs by the due date, the due date is called *large* or *unrestricted*; otherwise the due date is called *small* or *restricted*.

Hall *et al* (1991) prove that the single machine earliness–tardiness problem with a restricted due date to minimize the total (unweighted) earliness and tardiness $\sum(E_j + T_j)$ is \mathcal{NP} -hard in the ordinary sense and they provide a pseudo-polynomial dynamic programming algorithm; an alternative complexity proof is due to Hoogeveen and Van de Velde (1991). Using Lagrangean relaxation for deriving lower bounds, Hoogeveen *et al* (1994) develop a $(4/3)$ -approximation algorithm for the problem that requires $O(n \log n)$ time. The version of the problem with symmetric weights, that is, to minimize $\sum w_j(E_j + T_j)$ is solvable in pseudopolynomial time, and is also \mathcal{NP} -hard (see Hoogeveen and Van de Velde, 1991). For the case of equal weights, the problem is solvable in $O(n \log n)$ time. For symmetric weights, Kovalyov and Kubiak (1999) design an FPTAS. The main open question in this area is whether the problem with a common (either restricted or unrestricted) due date to minimize the functions $\sum(w_j E_j + w'_j T_j)$ with asymmetric weights is \mathcal{NP} -hard in the strong sense.

5.4. Scheduling at the end of Decade 4

By the end of this decade, the complexity status of most classical scheduling problems was known. Other than the recently emerging column generation techniques, there were few new algorithmic ideas that could be incorporated into enumerative algorithms for obtaining exact algorithms.

Many researchers turned their attention to the study of approximation algorithms and heuristics. Establishing the approximability status of a problem in terms of deriving lower and upper bounds on the worst-case ratio for a problem became a focus of many research studies. On the more practical side, local search approaches were developed for a multitude of scheduling problems. Several new types of local search applicable to scheduling problems appeared in the literature, and new devices were developed to enhance the performance of well-known methods such as simulated annealing, tabu search and genetic algorithms for scheduling problems.

Some excellent books appeared towards the end of the decade. The book by Pinedo (1995) superseded Baker (1974) as the main scheduling textbook. Pinedo's coverage of stochastic scheduling is a useful element of the book. Two other very good books, by Brucker (1995) and Blazewicz *et al* (1993) cover most important aspects of scheduling and gained popularity among researchers, lecturers and students. There have been several editions of each book.

The fourth decade saw the initiation of two important conference series. Luís Valadares Tavares and Jan Węglarz set up a Project Management and Scheduling (PMS) EURO Working Group in 1986. This Group organizes a conference every 2 years, with the first in the series taking place in Lisbon in 1988. The other series, Models and Algorithms for Planning and Scheduling Problems (MAPSP), was launched in 1993 by Rolf Möhring, Franz Radermacher and Grazia Speranza. The first conference was held at Villa Vigoni, Lake Como, Italy in 1993, with subsequent meetings taking place every 2 years.

6. Decade 5: enhanced scheduling models

Scheduling research carried out during the current, fifth decade is diverse in nature. Thus, it is difficult to detect strong themes that have attracted most attention. Even though many researchers were motivated by the need to create scheduling models that capture more of the features that arise in practice, the enhancements to classical scheduling models cannot be embedded into a unified framework. We only mention a small number of these enhanced models to give a flavour of the types of developments that have been made from the classical models introduced in the earlier decades.

The enhanced scheduling models studied during this decade include: online scheduling in which knowledge of jobs becomes available over time; scheduling with batching; models that take into account other parts of the supply chain; and scheduling with machine availability constraints, for example caused by periods of planned maintenance. This list is not exhaustive, since there have been many other interesting and important modelling developments during the decade, for example, scheduling with non-constant processing times, scheduling customer orders containing several jobs, and issues of handling disruptions and rescheduling. Several of these enhanced models are reviewed in the edited book of Leung (2004).

The fifth decade has also seen many practical applications of scheduling in a variety of industries and service organizations. As an example of this stream of research, we review some of the work on runway scheduling at London's Heathrow Airport, which has been the subject of several studies. A selection of other applications of scheduling are also described in the book of Leung (2004).

6.1. Online scheduling

In practice, information about jobs arriving in the future is unknown, whereas the assumption in classical scheduling is that full information about all jobs is known at the outset. In *online scheduling*, jobs become available over time. The quality of an online scheduling algorithm is typically evaluated by its *competitive ratio*. A polynomial-time algorithm

that finds a feasible solution with a value that is at most ρ times the optimal offline solution value is called a ρ -competitive algorithm. In online scheduling, it is possible to obtain a lower bound on the competitive ratio of any online algorithm by providing an instance and showing that whatever scheduling decisions are made, there are patterns of arriving jobs that always provide a certain ratio when comparing the schedule with the offline optimum. In some cases, it is possible to design a 'best possible' online algorithm in which the competitive ratio of the algorithm matches the lower bound that can be achieved by any algorithm.

As an illustration of the main concepts, consider an online problem with the goal of minimizing the total weighted completion time $\sum w_j C_j$. We derive the result of Hoogeveen and Vestjens (1996) that the competitive ratio of any online scheduling algorithm is at least 2. At time zero, there is a single job available: $r_1 = 0$, $p_1 = p$ and $w_1 = 1$, where $p > 0$. Suppose that, in the absence of any other job arrival, a scheduling algorithm decides to start this job at some time t , where $t \geq 0$. If $t \geq p$, then no other jobs are released. In this case, for the resulting schedule S , we have $\sum w_j C_j(S) = t + p$, while the optimal offline schedule S^* starts the job at time zero to give $\sum w_j C_j(S^*) = p$. Therefore, $\sum w_j C_j(S) / \sum w_j C_j(S^*) = 1 + t/p \geq 2$. Alternatively, suppose that $t < p$. In this case, a second job is released at time $t + \varepsilon$, where ε is a small and positive: $r_2 = t + \varepsilon$, $p_2 = \varepsilon$ and $w_2 = w$, where w is a large positive integer. In this case, we have $\sum w_j C_j(S) = (t + p) + w(t + p + \varepsilon)$, while the optimal offline schedule starts job 2 at time $t + \varepsilon$ and job 1 at time $t + 2\varepsilon$ to give $\sum w_j C_j(S^*) = w(t + 2\varepsilon) + (t + 2\varepsilon + p)$. In the limiting case when $w \rightarrow \infty$, we obtain $\sum w_j C_j(S) / \sum w_j C_j(S^*) \rightarrow (t + p + \varepsilon) / (t + 2\varepsilon) > 2$ for $\varepsilon < (p - t)/3$. Thus, whatever decision is made to create the schedule S , a ratio of at least 2 is obtained.

Anderson and Potts (2004) show that the competitive ratio of the so-called delayed SWPT algorithm is 2, and hence the algorithm is best possible. In the delayed SWPT algorithm, at any time t when the machine becomes available, a job j with the smallest value of p_j/w_j is chosen. If $p_j \leq t$, job j is scheduled to start at time t ; otherwise, the machine is kept idle until either time p_j or another job arrives, whichever occurs earliest.

General approaches in online scheduling typically involve delaying the processing of jobs until more information about the future is gathered. Unfortunately, to obtain a tight analysis, the delaying strategy must be problem specific. Thus, there are few general techniques that can be applied to derive the competitive ratio of an online algorithm. For a review on online scheduling results, we refer to the comprehensive review of Pruhs *et al* (2004).

It is worth noting that competitive analysis is far more common than probabilistic analysis when jobs may potentially arrive in the future. A major difficulty with probabilistic analysis is to specify appropriate probability distributions for the future jobs.

6.2. Scheduling with batching

There are some manufacturing environments where processing jobs singly leads to inefficiencies, while processing jobs in batches leads to a reduction in setup times or processing times. A typical example that allows a single setup to be shared among several jobs is a machine for which different machine tools are available. A setup is required to load a machine tool onto the machine. A batch is a set of consecutively processed jobs that require the same machine tool, so that a single setup is required prior to the processing of a batch. An example of a gain in processing efficiency occurs when several jobs can be processed together. A burn-in oven of the type used in semiconductor manufacturing provides a practical instance where this type of simultaneous processing occurs (for more details, we refer to Hochbaum and Landy, 1997).

We now provide some of the concepts and assumptions to be used when describing the various scheduling with batching models that we address. In the general *family* model, a splitting of the jobs into groups, called families, is specified in advance. Further, the jobs of each family can be partitioned by the decision-maker into smaller groups, called *batches*. Normally, a *setup* is required before a machine can start processing the job that starts a batch; on the other hand, due to similarity of jobs that belong to the same family, no setup is needed between the jobs in a batch. To classify various versions of the general model, several aspects should be taken into consideration. One of them is due to different possible interpretations of the availability of a job, either for further processing or for delivery to the customer. Under *job availability*, a job becomes available immediately after its processing is finished, while under *batch availability*, all jobs in a batch become available simultaneously upon completion of the batch. Another important aspect relates to the processing time of a batch: under *sequential batch processing* (also known as *s-batch* or *sum-batch*) the batch processing time is equal to the sum of the processing times of its jobs, while under *parallel batch processing* (also known as *p-batch* or *max-batch*) all jobs in a batch are assumed to be processed simultaneously, so that the batch processing time is defined by the maximum processing time of its jobs. An additional source of variety in batching models concerns the type of the setup and its duration: anticipatory *versus* non-anticipatory, zero (typical for some max-batch models), constant, machine dependent, family dependent, etc. Further features of some models include the size of a batch (restricted or not), compatibility constraints (some jobs cannot be batched together), etc. We refer to the surveys of Potts and Van Wassenhove (1992) and Potts and Kovalyov (2000), which give a detailed account of the models and results in scheduling with batching up to the year 2000.

Below we give a brief overview of results for scheduling with batching models, focusing on complexity and approximation. We start with the family model, where the jobs are

split into F families and job availability is assumed. Further, the setup times for each batch are family dependent, machine dependent for the flow shop and open shop, and anticipatory (ie, a setup can be performed on a machine before the jobs of the corresponding batch are available on that machine). Monma and Potts (1989) show that for the case of a single machine, there are several objective functions that allow the jobs within each family to be sequenced using well-known priority rules. Thus, the problem reduces to one of merging F sequences of jobs, which can be solved dynamic programming algorithms that require polynomial time when F is fixed. For arbitrary F , Bruno and Downey (1978) show that problems with objective functions based on due dates are \mathcal{NP} -hard, but open with respect to pseudopolynomial solvability. Also, minimizing the sum of (weighted) completion times $\sum (w_j)C_j$ is open for arbitrary F .

The two-machine flow shop and open shop for the family model with the makespan objective function are NP-hard, as proved by Kleinau (1993). For approximation algorithms that process each family as a single batch, Chen *et al* (1998) establish that the worst-case ratio is $3/2$ for the flow shop, while Strusevich (2000) and independently Liu and Yu (2000) establish that the worst-case ratio is $5/4$ for the open shop. By allowing a family to be split at most once, Chen *et al* (1998) design a $4/3$ -approximation algorithm for the flow shop, while Billaut *et al* (2008) develop a $6/5$ -approximation algorithm for the open shop and show that there exists an optimal schedule in which no family is split more than once.

We now turn our attention to the max-batch model which is applicable to a burn-in oven in semiconductor manufacturing. In this model, no families of jobs are specified and no setup time is required. A systematic complexity analysis of single machine max-batch problems with various objective functions and assumptions on the size of a batch is provided by Brucker *et al* (1998). An analogous study by Potts *et al* (2001) considers the two-machine flow shop, job shop and open shop max-batch scheduling problems to minimize the makespan. Typically, those problems with a restriction on batch size are harder to solve than those in which the size of a batch can be arbitrary.

6.3. Supply chain scheduling

Production is a key function within a supply chain. Therefore, rather than considering production scheduling in isolation, several researchers have designed models that integrate several of the operational functions that form the supply chain. The motivation is that by coordinating these functions, rather than considering each in isolation, there are likely to be gains in efficiency and corresponding cost reductions.

One example of an integrated model within a supply chain involves production and distribution operations. After the manufacture of a product, it must be shipped to the customer. A shipment can be an individual product, or a batch of products for several customers can be shipped together. Further,

for shipments in batches, the customers may be in different locations so that routing decisions are necessary, or the batches may be constrained in composition so that they are all for customers at the same location. Within the integrated model, the scheduling and delivery decisions should be coordinated if there is to be a high level of customer service provided in a cost-effective way. Research on these types of models was inspired by the contributions of Cheng and Kahlbacher (1993) and Lee and Chen (2001); a thorough survey of models and results is provided by Chen (2009). Since many of the models require batches to be formed for delivery, there are linkages with the scheduling and batching models discussed in Section 6.2.

Another example of an integrated model involves suppliers who provide components to a manufacturer. The manufacturer can only start processing when components from all suppliers have been received. Chen and Hall (2007) consider such a two-stage assembly system where manufacturing is assumed to be a non-bottleneck operation. Each supplier's objectives is to minimize total completion time, while the manufacturer's objective is to minimize either total completion time or maximum lateness. The main contribution of Chen and Hall is an analysis of conflict. Specifically, they analyse how far from optimal the best suppliers' schedule is when applied to the manufacturer's problem, and vice versa.

There are integrated models that consider more than two stages of the supply chain. Hall and Potts (1993) consider a supply chain in which a supplier delivers to several manufacturers, who in turn deliver to customers. As above, batches are used for delivery between supplier and manufacturer, and between manufacturer and customers. The supplier's scheduling and delivery decisions define release dates of the jobs at the manufacturer. Dynamic programming algorithms are developed for the supplier's scheduling problem, for the manufacturer's scheduling problem, and for the combined supply chain problem where both supplier and manufacturer are considered. Using examples, Hall and Potts show that decisions with the supplier and manufacturer acting independently can be much more costly than the solution of the combined problem.

6.4. Scheduling with machine availability constraints

The assumption in classical scheduling models is that all machines are continuously available from the start of the planning horizon. However, there are several situations under which this assumption is invalid. For example, periods of planned maintenance may result in machines becoming unavailable for certain time intervals. Another situation that creates unavailability intervals on machines occurs when high-priority orders are pre-assigned machine time in advance of creating a schedule of regular jobs.

Since the beginning of the 1990s, there has been a noticeable interest in models with machine non-availability intervals (MNAIs). Most research has concentrated on a pure

deterministic situation when the start of an unavailability interval and its duration are known in advance. There are several good surveys that address various aspects and provide results for models of this type, namely those of Lee (1996), Sanlaville and Schmidt (1998), Schmidt (2000) and Lee (2004). There are several different interpretations of how the processing of jobs is affected by an MNAI, denoted by I . The processing of such an operation after interval I , that is, when the machine becomes available again is: (i) resumed from the point of interruption, known as the *resumable* scenario; (ii) starts from scratch, known as the *non-resumable* scenario; or (iii) partially reprocessed before the remaining processing can be performed, known as the *semi-resumable* scenario.

Since most problems with fixed MNAIs are \mathcal{NP} -hard, the most interesting issue is to draw a sharp borderline between the problems for which there is an approximation algorithm that delivers a fixed worst-case ratio and those for which that is not possible. For example, under the non-resumable scenario, a single machine problem to minimize the makespan reduces to a knapsack problem in the case of a single MNAI and is not approximable within any constant factor in the case of two or more MNAIs. Under the resumable scenario, the two-machine flow shop problem to minimize the makespan may admit a constant ratio approximation algorithm only if there are several MNAIs on the first machine or a single MNAI on the second machine. The problem with a single MNAI under the resumable scenario admits an FPTAS, such as the one designed by Ng and Kovalyov (2004), while under the semi-resumable scenario it can be solved by a PTAS developed by Kubzin *et al* (2009).

In some environments, a MNAI can be a mandatory machine maintenance activity that must take place within a prescribed time interval. There are alternative models which address the effect of machine maintenance: the maintenance activity may improve the processing rate of the machine; the maintenance activity may have to be completed before a given deadline; or the duration of the maintenance activity will increase if its start is delayed, etc. For further details, we refer to Lee and Chen (2000), Lee and Leon (2001) and Kubzin and Strusevich (2006).

6.5. Application to runway scheduling

The runway at an airport limits the amount of air traffic that can be accommodated on any day. Any improvement in runway utilization creates the potential for additional flights and hence greater revenue for the airport. An obvious method of achieving better utilization is through reducing the times between landings and/or take-offs. However, safety considerations due to turbulence caused by wake vortices dictate minimum separation times between successive landings or take-offs. Minimum separation times depend on aircraft weight, and are smaller when a heavier aircraft is preceded by a lighter aircraft and larger when a lighter aircraft is preceded by a heavier aircraft. A key element in the scheduling of

a runway is therefore to aim for a sequence in which the aircraft appear in non-decreasing order of their weight.

As a case study, we use runway scheduling at London's Heathrow airport to show how local search methods are used to tackle practical scheduling problems. Heathrow airport operates in segregated mode, which means that (usually) one of its runways is used for landings and the other is used for takeoffs. This mode of operation creates two independent scheduling problems, one for landings and one for take-offs. We discuss how local search heuristics have been successful in tackling these two problems.

Beasley *et al* (2001) consider the landing problem at Heathrow. Each aircraft to be scheduled for landing is regarded as a job j having a release date r_j (the earliest landing time based on the speed and aircraft's current position), a deadline \bar{d}_j (the latest landing time based on fuel) and a target landing time t_j (based on the timetabled arrival time). Moreover, s_{jk} represents the minimum separation time between aircraft j and aircraft k when k follows j in the landing sequence. The schedule of landings is represented by a continuous variable y_j for each aircraft j , where $0 \leq y_j \leq 1$, that defines a landing time $x_j = r_j + y_j(\bar{d}_j - r_j)$. Beasley *et al* propose an objective function $\sum_j \text{sgn}(t_j - x_j)(t_j - x_j)^2$ that is to be maximized. Thus, a landing before the target time contributes positively and one after the target time contributes negatively to the objective function. A genetic algorithm (population heuristic) is used that allows infeasibility with respect to separation times. The infeasibility penalty is defined as $\sum_{j,k} \max\{0, s_{jk} - (x_k - x_j)\}$, where the summation is over all aircraft j and k with $j \neq k$ and $x_j \leq x_k$ (but not double counting if $x_j = x_k$). A uniform crossover is applied to yield one offspring from two parents, where binary tournament selection is used to choose the two parents. The population is created under a steady-state replacement scheme, where the infeasibility penalty has priority over the objective function when comparing an offspring with the population members.

Beasley *et al* test their genetic algorithm on a single data set that was formed of all aircraft within 100 nautical miles of the London Air Traffic Control Centre at 8.15 am on a Friday in September 1998. The data contained 20 aircraft, with target landing times set as actual recorded landing times. The results of applying the genetic algorithm show that the average delay of the aircraft increases by 40 s under the genetic algorithm compared to the solution actually used, and the maximum delay is increased by 187 s. However, the makespan decreases by 41 s, which is advantageous for the subsequent scheduling of aircraft that enter the system.

The scheduling of take-offs is more complicated than for landings, mainly because the layout of the taxiways at the airport restricts the reordering of aircraft and hence limits the set of feasible take-off sequences. Atkin *et al* (2007) consider the take-off problem at Heathrow. In their model, aircraft can be regarded as jobs, with each job having a target time for take-off, around which is built a 15-min time window. The minimum separation time when aircraft k follows aircraft

j depends on the weight categories of the two aircraft and also on the departure routes because there is also a minimum separation on each route that depends on the speed groups of aircraft j and k if they use the same route. A complex composite objective function with many components is to be minimized, where the primary components relate to delays from the time windows and the target take-off times. However, other terms are also included to avoid short-term solutions, such as giving excess priority to an aircraft that is already late or giving too low a priority to a light aircraft that is likely to eventually be sequenced after a heavier aircraft. A tabu search algorithm is proposed that searches over take-off sequences, using swap and shift moves. However, only a limited subset of neighbours is generated, and each such solution is checked for feasibility by a heuristic that attempts to route the aircraft along the taxiways so that they arrive at the runway in the desired order.

Atkin *et al* apply their tabu search algorithm to six real data sets, each with between 250 and 350 aircraft (with a very small number of light aircraft included). Results show that the schedule provided by the tabu search algorithm misses fewer of the time windows for take-off (19 misses) than the schedule that was actually used (39 misses). Also, the delays from reaching the holding point to take-off are typically about 20% lower for the schedule produced by the tabu search algorithm.

6.6. Other results

In the context of local search, some interesting issues relating to the performance of the algorithms have received only minor attention in the literature. An obvious question to ask is ‘what is the running time of a descent algorithm if an arbitrary starting solution is selected and the algorithm ends at a local optimum?’. Another question is ‘what is the worst-case ratio associated with a local search algorithm that ends at an arbitrary local optimum?’. An answer to the first question is provided by Brucker *et al* (1996, 1997) for certain problems where they show that a local optimum can be obtained in polynomial time. As an illustration, for the problem of minimizing the makespan on identical parallel machines, they show that for a certain implementation of the shift neighbourhood that moves a job from the most heavily loaded machine to the least heavily loaded machine, a local optimum can always be achieved using $O(n^2)$ moves. For the second question, Schuurman and Vredeveld (2001) provide worst-case bounds for problems of minimizing the makespan on m parallel machines for some of the commonly used neighbourhoods. As an illustration, for both the swap neighbourhood in which jobs on two different machines are interchanged and the shift neighbourhood, the worst-case bound of a local optimum is $2 - 2/(m + 1)$.

Another topic within local search is the use of an exponential or large neighbourhood that can be searched in polynomial time. For example, Congram *et al* (2002) introduce *dynasearch* as a local method that explores a neighbourhood

of exponential size by dynamic programming. In addition to its theoretical interest, the computational results of Congram *et al* for the single machine total weighted tardiness problem show that it outperforms previously proposed local search methods. For a review of methodology for exploring large neighbourhoods efficiently, we refer to Ahuja *et al* (2002).

Following a study of statistical properties of solution landscapes for the TSP by Boese *et al* (1994), Reeves (1999) extended this work to the permutation flow shop problem, and subsequently his research in this area led to several follow-up papers. The key idea in this area is to study local minima; relevant issues are the number of local minima for a particular neighbourhood, the number of neighbourhood moves needed to move from one local minimum to another, and the sizes of the basins of attraction of local minima. A better understanding of these issues helps in the estimation of how likely some neighbourhood is going to be in yielding high-quality solutions.

Sitters (2005) proves that minimizing the sum of completion times on two unrelated parallel machines when preemption is allowed is \mathcal{NP} -hard in the strong sense. The interest in this result is that, unlike for essentially all other scheduling problems, the preemptive version appears to be harder than its polynomially solvable non-preemptive counterpart.

6.7. Scheduling at the end of Decade 5

At the end of this decade, scheduling had become much more fragmented. A large number of researchers were working in the area but seemingly on a huge variety of problems and adopting a multitude of different approaches. In spite of the vast body of research being produced, a large gap remains between theory and practice.

The fifth decade saw the launch of the *Journal of Scheduling* with its first issue published in June 1998. It was the brainchild of Edmund Burke, who has been the editor-in-chief throughout.

7. A look to the future

To conclude this paper, we analyse the current state-of-the-art in scheduling research with a view to identifying research areas that deserve to be prioritized.

A major research theme during the past three decades has involved defining the boundary between polynomially solvable problems and those that are \mathcal{NP} -hard. For classical scheduling problems, this activity is almost complete, with very few problems still having an open complexity status. During Decade 5 with the introduction of various enhanced scheduling models, complexity classification remained active since it is a natural first step in addressing a new class of problems. However, unless new complexity classes are introduced, this research theme will only be useful as new models are introduced, and much less influential than during the first 50 years of scheduling.

Since the flurry of activity during Decades 2 and 3, research studies on the design of branch and bound algorithms (and other enumerative approaches) for scheduling problems now feature less frequently in the literature. With the tools currently available, it is difficult to derive lower bounding schemes that are strong enough to provide adequate pruning of the search tree. Although the problem is somewhat alleviated by the speed and memory of modern day computers, the combinatorial growth of the solution space provides an obstacle to the exact solution of practical problems. The hybridization of branch and bound and other techniques such as local search offers an interesting way forward.

In view of the relative stagnation of research on branch and bound algorithms for scheduling problems, the study of approximation algorithms and heuristics remains an important research theme. Major advances have been made in the area of approximation algorithms during Decades 4 and 5. Nevertheless, major challenges remain. One of these arises for those problems where there is a large gap between the worst-case ratio of the best available approximation algorithm and the lower bound on performance that can be established (assuming that $\mathcal{P} \neq \mathcal{NP}$). Closing such gaps is a worthy goal, although it appears difficult to achieve. Another challenge concerns performance measures by which approximation algorithms are evaluated. The standard ratio bound assumes that the optimal objective function value is positive; otherwise, the ratio does not provide a well-defined performance measure. Moreover, focusing on the worst case does not necessarily provide a suitable assessment of performance since a worst-case instance may be atypical of the types of instances that are encountered in the environment motivating the investigation. There is clearly some scope for the introduction of alternative measures of approximation algorithm performance that avoid these drawbacks.

Vast strides have been made in local search during Decades 4 and 5. At the start of Decade 4, very little was known about the topic, while today there is much practical advice as to which types of local search methods perform well and what add-on devices should be used to improve solution quality. Local search is often the method of choice for real-life scheduling problems, and is able to take into account a variety of complicating hard and soft constraints that are invariably present. Although some scope remains for the development of improved algorithms, an important research topic involves gaining a better understanding of why certain local search procedures perform well in practice. Such an understanding could, in turn, provide the necessary insights into the design of superior local search procedures.

In summary, developments in the field of scheduling from the landmark work in the mid-1950s to the present day have been immense. Progress has benefited from contributions by researchers from varied disciplines (eg, mathematics, operational research/management science, computer science, engineering, and economics). The area is mature with many theories and techniques that can be used to tackle

problems, and rich in variety (eg, deterministic or stochastic models, exact or approximate solutions, application-oriented or theory-based). In spite of the progress within scheduling, many challenges remain for new and established researchers in this exciting and rewarding area.

Acknowledgements—The authors are grateful to NG Hall and ME Posner for some useful suggestions on the content of some sections of this paper.

References

- Adams J, Balas E and Zawack D (1988). The shifting bottleneck procedure for job shop scheduling. *Mngt Sci* **34**: 391–401.
- Afrati F, Bampis E, Chekuri C, Karger D, Kenyon C, Khanna S, Mills I, Queyranne M, Skutella M, Stein C and Sviridenko M (1999). Approximation schemes for minimizing average weighted completion time with release dates. In: *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press: Los Alamitos, CA, pp 32–43.
- Ahuja RK, Ergun Ö, Orlin JB and Punnen AP (2002). A survey of very large-scale neighborhood search techniques. *Disc Appl Math* **123**: 75–102.
- Akers Jr SB and Friedman J (1955). A non-numerical approach to production scheduling problems. *Opns Res* **3**: 429–442.
- Aksjonov VA (1988). A polynomial-time algorithm for an approximate solution of a scheduling problem. *Upravlyaemye Sistemy* **28**: 8–11 (in Russian).
- Anderson EJ and Potts CN (2004). Online scheduling of a single machine to minimize total weighted completion time. *Math Opns Res* **29**: 686–697.
- Atkin JAD, Burke EK, Greenwood JS and Reeson D (2007). Hybrid metaheuristics to aid runway scheduling at London Heathrow airport. *Trans Sci* **41**: 90–106.
- Baker KR (1974). *Introduction to Sequencing and Scheduling*. Wiley: New York.
- Baker KR and Scudder GD (1990). Sequencing with earliness and tardiness penalties: A review. *Opns Res* **38**: 22–36.
- Balas E and Vazacopoulos A (1998). Guided local search with shifting bottleneck for job shop scheduling. *Mngt Sci* **44**: 262–275.
- Bárány I and Fiala T (1982). Nearly optimum solution of multi-machine scheduling problems. *Szigma* **15**: 177–191 (in Hungarian).
- Barnes JW and Chambers JB (1993). Solving the job shop scheduling problem with tabu search. *IIE Trans* **27**: 257–263.
- Beasley JE, Sonander J and Havelock P (2001). Scheduling aircraft landings at London Heathrow using a population heuristic. *J Opt Res Soc* **52**: 483–493.
- Billaut J-C, Gribkovskaia IV and Strusevich VA (2008). An improved approximation algorithm for the two-machine open shop scheduling problem with family setup times. *IIE Trans* **40**: 478–493.
- Blazewicz J, Ecker KH, Schmidt G and Węglarz J (1993). *Scheduling in Computer and Manufacturing Systems*. Springer: Berlin.
- Boese KD, Kahng AB and Muddu S (1994). A new adaptive multi-start technique for combinatorial global optimizations. *Opns Res Lett* **16**: 101–113.
- Bratley P, Florian M and Robillard P (1973). On sequencing with earliest starts and due dates with application to computing bounds for the $(n|jm|G|F_{\max})$ problem. *Nav Res Logist Quart* **20**: 57–67.
- Brooks GH and White CR (1965). An algorithm for finding optimal or near optimal solutions to the production scheduling problem. *J Indust Eng* **16**: 34–40.
- Brucker P (1995). *Scheduling Algorithms*. Springer: Berlin.
- Brucker P, Hurink J and Werner F (1996). Improving local search heuristics for some scheduling problems—I. *Disc Appl Math* **65**: 97–122.

- Brucker P, Hurink J and Werner F (1997). Improving local search heuristics for some scheduling problems. Part II. *Disc Appl Math* **72**: 47–69.
- Brucker P, Gladky A, Hoogeveen JA, Kovalyov MY, Potts CN, Tautenhahn T and Van de Velde SL (1998). Scheduling a batching machine. *J Schedul* **1**: 31–54.
- Bruno JL and Downey PJ (1978). Complexity of task sequencing with deadlines, set-up times and changeover costs. *SIAM J Comput* **7**: 393–404.
- Bruno JL, Coffman Jr EG and Sethi R (1974). Scheduling independent tasks to reduce mean finishing time. *Comm ACM* **17**: 382–387.
- Carlier J (1982). The one-machine sequencing problem. *Eur J Opl Res* **11**: 42–47.
- Carlier J and Pinson E (1989). An algorithm for solving the job-shop problem. *Mngt Sci* **35**: 164–176.
- Cerny V (1985). Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *J Opti Theory Appl* **45**: 41–51.
- Charlton JM and Death CC (1970). A method of solution for general machine-scheduling problems. *Opns Res* **18**: 689–707.
- Chen B and Strusevich VA (1993). Approximation algorithms for three-machine open shop scheduling. *ORSA J Comput* **5**: 321–326.
- Chen B, Glass CA, Potts CN and Strusevich VA (1996). A new heuristic for three-machine flow shop scheduling. *Opns Res* **44**: 891–898.
- Chen B, Potts CN and Strusevich VA (1998). Approximation algorithms for two-machine flow shop scheduling with batch setup times. *Math Progr B* **82**: 255–271.
- Chen Z-L (2009). Integrated production and outbound distribution scheduling: Review and extensions. *Opns Res*, to appear.
- Chen Z-L and Hall NG (2007). Supply chain scheduling: Conflict and cooperation in assembly systems. *Opns Res* **55**: 1072–1089.
- Chen Z-L and Powell WB (1999). Solving parallel machine scheduling problems by column generation. *INFORMS J Comput* **11**: 78–94.
- Cheng TCE and Kahlbacher HG (1993). Scheduling with delivery and earliness penalties. *Asia-Pacific J Opl Res* **10**: 145–152.
- Congram RK, Potts CN and Van de Velde SL (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS J Comput* **14**: 52–67.
- Conway RW, Maxwell WL and Miller LW (1967). *Theory of Scheduling*. Addison-Wesley: Reading, MA.
- Cook SA (1971). The complexity of theorem-proving procedures. In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*. The Association for Computing Machinery: New York, pp 151–158.
- Crauwels HAJ, Potts CN and Van Wassenhove (1998). Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS J Comput* **10**: 341–350.
- Croes GA (1958). A method for solving traveling-salesman problems. *Opns Res* **6**: 791–812.
- Crowder H and Padberg MW (1980). Solving large-scale symmetric travelling salesman problems to optimality. *Mngt Sci* **26**: 495–509.
- Davis L (1985). Job shop scheduling with genetic algorithms. In: Grefenstette JJ (ed). *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*. Lawrence Erlbaum: Hillsdale, NJ, pp 136–140.
- Della Croce F, Tadei R and Volta G (1995). A genetic algorithm for the job shop problem. *Comput Opns Res* **22**: 15–24.
- Dell'Amico M and Trubian M (1993). Applying tabu-search to the job-shop scheduling problem. *Ann Opns Res* **41**: 231–252.
- Dempster MAH, Lenstra JJ and Rinnooy Kan AHG (eds). (1982). *Deterministic and Stochastic Scheduling*. Reidel: Dordrecht, the Netherlands.
- de Werra D (1970). On some combinatorial problems arising in scheduling. *CORS J* **8**: 165–175.
- de Werra D (1989). Graph-theoretical models for preemptive scheduling. In: Slowinski R and Weglarz J (eds). *Advances in Project Scheduling*. Elsevier: Amsterdam, pp 171–185.
- Dorndorf U and Pesch E (1995). Evolution based learning in a job shop scheduling environment. *Comput Opns Res* **22**: 25–40.
- Du J and Leung JY-T (1990). Minimizing total tardiness on one machine is NP-hard. *Math Opns Res* **15**: 483–495.
- Eastman WL (1958a). *Linear programming with pattern constraints*. PhD thesis, Report No. BL. 20, The Computation Laboratory, Harvard University.
- Eastman WL (1958b). A solution to the traveling-salesman problem. Presentation at the American Summer Meeting of the Econometric Society, Cambridge, MA.
- Edmonds J (1965). Paths, trees, and flowers. *Canad J Math* **17**: 449–467.
- Elmaghraby SE (1968). The one-machine sequencing problem with delay costs. *J Indust Eng* **19**: 105–108.
- Emmons H (1969). One-machine sequencing to minimize certain functions of job tardiness. *Opns Res* **17**: 701–715.
- Falkenauer E and Bouffouix S (1991). A genetic algorithm for job shop. *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*. IEEE Computer Society Press: Los Alamitos, CA, pp 824–829.
- Federgruen A and Groenvelt H (1986). Preemptive scheduling of uniform machines by ordinary network flow techniques. *Mngt Sci* **32**: 341–349.
- Fisher ML (1976). A dual algorithm for the one-machine scheduling problem. *Math Progr* **11**: 229–251.
- Gabow HN and Kariv O (1982). Algorithms for edge coloring bipartite graphs. *SIAM J Comput* **11**: 117–129.
- Gantt HL (1916). *Work, Wages, and Profits*, 2nd edn. Engineering Magazine Co: New York, (reprinted by Hive Publishing Company: Easton, Maryland, 1973).
- Garey MR and Johnson DS (1979). *Computers and Intractability. A Guide to the Theory of NP-Completeness*. Freeman: San Francisco, CA.
- Garey MR, Johnson DS and Sethi R (1976). The complexity of flowshop and jobshop scheduling. *Math Opns Res* **1**: 117–129.
- Gelders L and Kleindorfer PR (1974). Coordinating aggregate and detailed scheduling decisions in the one-machine job shop: Part I, Theory. *Opns Res* **22**: 46–60.
- Gelders L and Kleindorfer PR (1975). Coordinating aggregate and detailed scheduling decisions in the one-machine job shop: Part II, Computation and structure. *Opns Res* **23**: 312–324.
- Gilmore PC and Gomory RE (1964). Sequencing a one-state variable machine: A solvable case of the traveling salesman problem. *Opns Res* **12**: 665–679.
- Glover F (1986). Future paths for integer programming and links to artificial intelligence. *Comput Opns Res* **13**: 533–549.
- Glover F (1989). Tabu search: Part I. *ORSA J Comput* **1**: 190–206.
- Glover F (1990). Tabu search: Part II. *ORSA J Comput* **2**: 4–32.
- Goldberg DE (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley: Reading, MA.
- Gonzalez T and Sahni S (1976). Open shop scheduling to minimize finish time. *J Assoc Comput Mach* **23**: 665–679.
- Gonzalez T and Sahni S (1978). Flowshop and jobshop schedules: Complexity and approximation. *Opns Res* **26**: 36–52.
- Graham RL (1966). Bounds for certain multiprocessing anomalies. *Bell Syst Techn J* **45**: 1563–1581.
- Graham RL (1969). Bounds on multiprocessing timing anomalies. *SIAM J Appl Math* **17**: 416–429.
- Graham RL, Lawler EL, Lenstra JK and Rinnooy Kan AHG (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann Disc Math* **5**: 287–326.
- Grötschel M (1980). On the symmetric travelling salesman problem: Solution of a 120-city problem. *Math Progr Study* **12**: 61–77.

- Gupta JND (1971). An improved combinatorial algorithm for the flowshop-scheduling problem. *Opns Res* **19**: 1753–1758.
- Hall LA (1998). Approximability of flow shop scheduling. *Math Progr B* **82**: 175–190.
- Hall LA, Schulz A, Shmoys DB and Wein J (1997). Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Math Opns Res* **22**: 513–544.
- Hall NG and Posner ME (1991). Earliness–tardiness scheduling problems, I: Weighted deviation of completion times about a common due date. *Opns Res* **39**: 836–846.
- Hall NG and Potts CN (1993). Supply chain scheduling: Batching and delivery. *Opns Res* **51**: 566–584.
- Hall NG, Kubiak W and Sethi SP (1991). Earliness–tardiness scheduling problems, II: Deviation of completion times about a restrictive common due date. *Opns Res* **39**: 836–846.
- Hariri AMA and Potts CN (1983). An algorithm for single machine sequencing with release dates to minimize total weighted completion time. *Disc Appl Math* **5**: 99–109.
- Held M and Karp RM (1971). The traveling-salesman problem and minimum spanning trees: Part II. *Math Progr* **1**: 6–25.
- Held M, Wolfe P and Crowder HP (1974). Validation of subgradient optimization. *Math Progr* **6**: 62–88.
- Hochbaum DS and Landy D (1997). Scheduling semiconductor burn-in operations to minimize total flowtime. *Opns Res* **45**: 874–885.
- Hochbaum DS and Shmoys DB (1987). Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *J Assoc Comput Mach* **34**: 144–162.
- Hochbaum DS and Shmoys DB (1988). A polynomial approximation scheme for machine scheduling on uniform processors: Using the dual approximating approach. *SIAM J Comput* **17**: 539–551.
- Hooqveen JA and Van de Velde SL (1991). Scheduling around a small common due date. *Eur J Opl Res* **55**: 237–242.
- Hooqveen JA and Vestjens APA (1996). Optimal online algorithms for single-machine scheduling. In: Cunningham WH, McCormick ST and Queyranne M (Eds). *Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science, Vol. 1084. Springer: Berlin, pp 404–414.
- Hooqveen JA, Oosterhout H and Van de Velde SL (1994). New lower and upper bounds for scheduling around a small common due date. *Opns Res* **42**: 102–110.
- Hooqveen JA, Lenstra JK and Van de Velde SL (1997). Sequencing and scheduling. In: Dell’Amico M, Maffioli F and Martello S (eds). *Annotated Bibliographies in Combinatorial Optimization*. Wiley: Chichester, pp 181–197.
- Hooqveen H, Schuurman P and Woeginger GJ (2001). Non-approximability results for scheduling problems with minsum criteria. *INFORMS J Comput* **13**: 157–168.
- Horn WA (1974). Some simple scheduling algorithms. *Nav Res Logist Quart* **21**: 177–185.
- Horowitz H and Sahni S (1976). Exact and approximate algorithms for scheduling nonidentical processors. *J Assoc Comput Mach* **23**: 317–327.
- Ibarra OH and Kim CE (1975). Fast approximation algorithms for the knapsack and sum of subset problems. *J Assoc Comput Mach* **22**: 463–468.
- Ignall E and Schrage L (1965). Application of the branch and bound technique to some flow-shop scheduling problems. *Opns Res* **13**: 400–412.
- Jackson JR (1955). *Scheduling a production line to minimize maximum tardiness*. Research Report 43, Management Science Research Project, University of California, Los Angeles, CA.
- Jackson JR (1956). An extension of Johnson’s result on job lot scheduling. *Nav Res Logist Quart* **3**: 201–203.
- Johnson SM (1954). Optimal two- and three-stage production schedules with setup times included. *Nav Res Logist Quart* **1**: 61–68.
- Karmarkar N (1984). A new polynomial-time algorithm for linear programming. In: *Proceedings of the 16th Annual ACM Symposium on Theory of Computing*. The Association for Computing Machinery: New York, pp 302–311.
- Karp RM (1972). Reducibility among combinatorial problems. In: Miller RE and Thatcher JM (eds). *Complexity of Computer Computations*. Plenum Press: New York, pp 85–103.
- Khachiyan LG (1979). A polynomial algorithm in linear programming. *Sov Math Doklady* **20**: 191–194.
- Kirkpatrick S, Gelatt Jr CD and Vecchi MP (1983). Optimization by simulated annealing. *Science* **220**: 671–680.
- Kleinau U (1993). Two-machine shop scheduling problems with batch processing. *Math Comput Model* **17**: 55–66.
- Kononov A and Sviridenko M (2002). A linear time approximation scheme for makespan minimization in an open shop with release dates. *Opns Res Lett* **30**: 276–280.
- Kovalyov MY and Kubiak W (1999). A fully polynomial approximation scheme for weighted earliness–tardiness problem. *Opns Res* **47**: 757–761.
- Kubzin MA and Strusevich VA (2006). Planning machine maintenance in two-machine shop scheduling. *Opns Res* **54**: 789–800.
- Kubzin MA, Potts CN and Strusevich VA (2009). Approximation results for flow shop scheduling problems with machine availability constraints. *Comput Opns Res* **36**: 379–390.
- Labetoulle J, Lawler EL, Lenstra JK and Rinnooy Kan AHG (1984). Preemptive scheduling of uniform machines subject to release dates. In: Pulleyblank WR (ed). *Progress in Combinatorial Optimization*. Academic Press: New York, pp 245–261.
- Lageweg BJ, Lenstra JK and Rinnooy Kan AHG (1978). A general bounding scheme for the permutation flow-shop problem. *Opns Res* **26**: 53–67.
- Lageweg BJ, Lawler EL, Lenstra JK and Rinnooy Kan AHG (1982). Computer-aided complexity classification of combinatorial problems. *Comm ACM* **25**: 817–822.
- Land AH and Doig A (1960). An automatic method for solving discrete programming problems. *Econometrica* **28**: 497–520.
- Lawler EL (1964). On scheduling problems with deferral costs. *Mngt Sci* **11**: 280–288.
- Lawler EL (1977). A ‘pseudopolynomial’ algorithm for sequencing jobs to minimize total tardiness. *Ann Disc Math* **1**: 331–342.
- Lawler EL (1978). Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Ann Disc Math* **2**: 75–90.
- Lawler EL (1991). Old stories. In: Lenstra JK, Rinnooy Kan AHG and Schrijver A (eds). *History of Mathematical Programming. A Collection of Personal Reminiscences*. CWI and North-Holland: Amsterdam, pp 97–106.
- Lawler EL and Labetoulle J (1978). On preemptive scheduling of unrelated parallel processors by linear programming. *J Assoc Comput Mach* **25**: 612–619.
- Lawler EL and Moore JM (1969). A functional equation and its application to resource allocation and sequencing problems. *Mngt Sci* **16**: 77–84.
- Lawler EL, Lenstra JK and Rinnooy Kan AHG (1982a). Recent developments in deterministic sequencing and scheduling: A survey. In: Dempster MAH, Lenstra JK and Rinnooy Kan AHG (eds). *Deterministic and Stochastic Scheduling*. Reidel: Dordrecht, pp 35–73.
- Lawler EL, Luby MG and Vazirani VV (1982b). Scheduling open shops with parallel machines. *Opns Res Lett* **82**: 161–164.
- Lawler EL, Lenstra JK, Rinnooy Kan AHG and Shmoys DB (1993). Sequencing and scheduling: Algorithms and complexity. In: Graves S, Rinnooy Kan AHG and Zipkin P (Eds). *Handbooks in Operations Research and Management Science, Volume 4, Logistics of Production and Inventory*, North Holland: Amsterdam, pp 445–522.

- Lee C-Y (1996). Machine scheduling with an availability constraint. *J Global Opti* **9**: 395–416.
- Lee C-Y (2004). Machine scheduling with availability constraints. In: Leung JY-T (ed). *Handbook of Scheduling: Algorithms, Models and Performance Analysis*. Chapman & Hall/CRC: Boca Raton, FL, pp 22-1–22-13.
- Lee C-Y and Chen Z-L (2000). Scheduling jobs and maintenance activities on parallel machines. *Nav Res Logist* **47**: 145–165.
- Lee C-Y and Chen Z-L (2001). Machine scheduling with transportation considerations. *J Schedul* **4**: 3–24.
- Lee C-Y and Leon J (2001). Machine scheduling with a rate-modifying activity. *Eur J Opl Res* **128**: 119–128.
- Lenstra JK (1998). The mystical power of twoness: In memoriam Eugene L. Lawler. *J Schedul* **1**: 3–14.
- Lenstra JK and Rinnooy Kan AHG (1978). Complexity of scheduling under precedence constraints. *Opns Res* **26**: 22–35.
- Lenstra JK, Rinnooy Kan AHG and Brucker P (1977). Complexity of machine scheduling problems. *Ann Disc Math* **1**: 343–362.
- Lenstra JK, Shmoys DB and Tardos É (1990). Approximation algorithms for scheduling unrelated parallel machines. *Math Progr* **46**: 259–271.
- Leung JYT (ed). (2004). *Handbook of Scheduling: Algorithms, Models and Performance Analysis*. Chapman & Hall/CRC, Boca Raton, FL.
- Lin S (1965). Computer solutions of the traveling salesman problem. *Bell Syst Techn J* **44**: 2245–2269.
- Liu Z and Yu W (2000). Scheduling groups of jobs in two-machine open shop. *J East China Univ Sci Technol* **26**: 665–669 (in Chinese).
- Lomnicki ZA (1965). A 'branch-and-bound' algorithm for the exact solution of the three-machine scheduling problem. *Opl Res Quart* **16**: 89–100.
- Martel CU (1982). Preemptive scheduling with release times, deadlines and due times. *J Assoc Comput Mach* **29**: 812–829.
- Matsuo H, Suh CJ and Sullivan RS (1988). A controlled search simulated annealing method for the general problem. Working paper 03-04-88, Graduate School of Business, University of Texas at Austin, TX.
- Matsuo H, Suh CJ and Sullivan RS (1989). A controlled search simulated annealing method for the single machine weighted tardiness problem. *Ann Opns Res* **21**: 85–108.
- McMahon GB (1969). Optimal production schedules for flowshops. *CORS J* **7**: 141–151.
- McMahon GB and Burton PG (1967). Flow-shop scheduling with the branch-and-bound method. *Opns Res* **15**: 473–481.
- McMahon GB and Florian M (1975). On scheduling with ready times and due dates to minimize maximum lateness. *Opns Res* **23**: 475–482.
- McNaughton R (1959). Scheduling with deadlines and loss functions. *Mngt Sci* **6**: 1–12.
- Miliotis P (1976). Integer programming approaches to the travelling salesman problem. *Math Progr* **10**: 367–378.
- Miliotis P (1978). Using cutting planes to solve the symmetric travelling salesman problem. *Math Progr* **15**: 177–188.
- Monma CL and Potts CN (1989). On the complexity of scheduling with batch setup times. *Opns Res* **37**: 798–804.
- Monma CL and Rinnooy Kan AHG (1983). A concise survey of efficiently solvable cases of the permutation flow-shop problem. *RAIRO Rech Oper* **17**: 105–119.
- Monma CL and Sidney JB (1979). Sequencing with series-parallel precedence constraints. *Math Opns Res* **4**: 215–234.
- Moore JM (1968). An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Mngt Sci* **15**: 102–109.
- Muth JF and Thompson GL (eds). (1963). *Industrial Scheduling*. Prentice-Hall, Englewood Cliffs, NJ.
- Nabeshima I (1961). The order of n items processed on m machines: I. *J Opns Res Soc Japan* **3**: 170–175.
- Nabeshima I (1967). On the bound of makespans and its application in M machine scheduling problem. *J Opns Res Soc Japan* **9**: 98–136.
- Németi L (1964). Das Reihenfolgeproblem in der Fertigungsprogrammierung und Linear-planung mit logischen Bedingungen. *Mathematica (Cluj)* **6**: 87–99.
- Ng CT and Kovalyov MY (2004). An FPTAS for scheduling a two-machine flowshop with one unavailability interval. *Nav Res Logist* **51**: 307–315.
- Nicholson TAJ (1967). A sequential method for discrete optimization problems and its application to the assignment, travelling salesman, and three machine scheduling problems. *J Inst Math Appl* **3**: 362–375.
- Nowicki E and Smutnicki C (1996). A fast taboo search algorithm for the job shop problem. *Mngt Sci* **42**: 797–813.
- Nowicki E and Smutnicki C (1996). A fast tabu search algorithm for the permutation flow-shop problem. *Eur J Opl Res* **91**: 160–175.
- Ogbu FA and Smith DK (1990). The application of the simulated annealing algorithm to the solution of the $n|m|C_{\max}$ flowshop problem. *Comput Opns Res* **17**: 243–253.
- Osman IH and Potts CN (1989). Simulated annealing for permutation flow-shop scheduling. *Omega* **17**: 551–557.
- Padberg MW and Hong S (1980). On the symmetric travelling salesman problem: A computational study. *Math Progr Study* **12**: 78–107.
- Piebler J (1960). Ein Beitrag zum Reihenfolge problem. *Unternehmensforschung* **4**: 138–142.
- Pinedo M (1995). *Scheduling: Theory, Algorithms, and Systems*. Prentice Hall: Englewood Cliffs, NJ.
- Pinedo M and Schrage L (1982). Stochastic shop scheduling: A survey. In: Dempster MAH, Lenstra JK and Rinnooy Kan AHG (eds). *Deterministic and Stochastic Scheduling*. Riedel: Dordrecht, pp 181–196.
- Potts CN (1974). *The job-machine scheduling problem*. PhD thesis, University of Birmingham, UK.
- Potts CN (1980). An adaptive branching rule for the permutation flow-shop problem. *Eur J Opl Res* **5**: 19–25.
- Potts CN (1985). Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Disc Appl Math* **10**: 155–164.
- Potts CN and Kovalyov MY (2000). Scheduling with batching: A review. *Eur J Opl Res* **120**: 228–249.
- Potts CN and Van Wassenhove LN (1982). A decomposition algorithm for the single machine total tardiness problem. *Opns Res Lett* **1**: 177–181.
- Potts CN and Van Wassenhove LN (1983). An algorithm for single machine sequencing with deadlines to minimize total weighted completion time. *Eur J Opl Res* **12**: 379–387.
- Potts CN and Van Wassenhove LN (1985). A branch and bound algorithm for the total weighted tardiness problem. *Opns Res* **33**: 363–377.
- Potts CN and Van Wassenhove LN (1991). Single machine tardiness sequencing heuristics. *IIE Trans* **23**: 346–354.
- Potts CN and Van Wassenhove LN (1992). Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity. *J Opl Res Soc* **43**: 395–406.
- Potts CN, Strusevich VA and Tautenhahn T (2001). Scheduling batches with simultaneous job processing for two-machine shop problems. *J Schedul* **4**: 25–51.
- Pruhs K, Sgall J and Torng E (2004). Online scheduling. In: Leung JY-T (ed). *Handbook of Scheduling: Algorithms, Models and Performance Analysis*. Chapman & Hall/CRC: Boca Raton, FL, pp 15-1–15-43.

- Queyranne M and Wang Y (1991). Single machine scheduling polyhedra with precedence constraints. *Math Opns Res* **16**: 1–20.
- Reeves CR (1995). A genetic algorithm for flowshop sequencing. *Comput Opns Res* **22**: 5–13.
- Reeves CR (1999). Landscapes, operators and heuristic search. *Ann Opns Res* **86**: 473–490.
- Rinnooy Kan AHG, Lageweg BJ and Lenstra JK (1975). Minimizing total costs in one-machine scheduling. *Opns Res* **23**: 908–927.
- Rothkopf MH (1966). Scheduling independent tasks on parallel processors. *Mngt Sci* **12**: 437–447.
- Roy B and Sussman B (1964). *Les problèmes d'ordonnement avec contraintes disjonctives*. Note DS No. 9 bis, SEMA, Montrouge.
- Sahni S (1976). Algorithms for scheduling independent tasks. *J Assoc Comput Mach* **23**: 116–127.
- Sanlaville E and Schmidt G (1998). Machine scheduling with availability constraints. *Acta Informat* **5**: 795–811.
- Schmidt G (2000). Scheduling with limited machine availability. *Eur J Opl Res* **121**: 1–15.
- Schrage L (1968). A proof of the shortest remaining processing time processing discipline. *Opns Res* **16**: 687–690.
- Schrage L (1970a). Solving resource-constrained network problems by implicit enumeration—Nonpreemptive case. *Opns Res* **18**: 253–278.
- Schrage L (1970b). *A bound based on the equivalence of min-max-completion time and min-max-lateness scheduling objectives*. Report 7042, Department of Economics and Graduate School of Business, Chicago, IL.
- Schuurman P and Vredevelde T (2001). Performance guarantees of local search for multiprocessor scheduling. In: Aardal KI and Gerards AMH (Eds). *Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science, Vol. 2081. Springer: Berlin, pp 370–382.
- Schuurman P and Woeginger GJ (1999). Polynomial time approximation algorithms for machine scheduling: Ten open problems. *J Schedul* **2**: 203–213.
- Sevastianov SV (1994). On some geometric methods in scheduling theory: A survey. *Disc Appl Math* **55**: 59–82.
- Sevastianov SV (1995). Vector summation in Banach space and polynomial algorithms for flow shops and open shops. *Math Opns Res* **20**: 90–103.
- Sevastianov SV and Woeginger GJ (1998). Makespan minimization in open shops: A polynomial time approximation scheme. *Math Progr B* **82**: 191–198.
- Shwimer J (1972). On the N -job one-machine, sequence-independent scheduling problem with tardiness penalties: A branch-bound solution. *Mngt Sci* **18**: B301–B313.
- Sitters RA (2005). Complexity of preemptive minsum scheduling on unrelated parallel machines. *J Algorithms* **57**: 37–48.
- Skutella M (2001). Convex quadratic and semidefinite programming relaxations in scheduling. *J Assoc Comput Mach* **48**: 206–242.
- Smith WE (1956). Various optimizers for single-stage production. *Nav Res Logist Quart* **3**: 59–66.
- Smith SP (1992). Experiment on using genetic algorithms to learn scheduling heuristics. In: Biswas G (ed). *Proceedings of SPIE, Volume 1707, Applications of Artificial Intelligence X: Knowledge-Based Systems*, The International Society for Optical Engineering: Bellingham, WA, pp 378–386.
- Smith RD and Dudek RA (1969). Errata. *Opns Res* **17**: 756.
- Storer RH, Wu SD and Vaccari R (1992). New search spaces for sequencing problems with application to job shop scheduling. *Mngt Sci* **38**: 1495–1509.
- Strusevich VA (2000). Group technology approach to the open shop scheduling problem with batch setup times. *Opns Res Lett* **26**: 181–192.
- Sturm LBJM (1970). A simple optimality proof of Moore's sequencing algorithm. *Mngt Sci* **17**: 116–118.
- Szwarc W (1968). On some sequencing problems. *Nav Res Logist Quart* **15**: 127–155.
- Szwarc W (1971). Elimination methods in the $m \times n$ sequencing problem. *Nav Res Logist Quart* **18**: 295–305.
- Szwarc W (1973). Optimal elimination methods in the $m \times n$ sequencing problem. *Opns Res* **21**: 1250–1259.
- Taillard E (1990). Some efficient heuristic methods for the flow shop sequencing problem. *Eur J Opl Res* **47**: 65–74.
- Taillard E (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA J Comput* **6**: 108–117.
- Tanaev VS, Gordon VS and Shafransky (1984). *Scheduling Theory. Single-Stage Systems* (in Russian). Nauka: Moscow; English translation by Kluwer: Dordrecht, 1994.
- Tanaev VS, Gordon VS and Shafransky (1989). *Scheduling Theory. Multi-Stage System* (in Russian). Nauka: Moscow; English translation by Kluwer: Dordrecht, 1994.
- Van den Akker JM, Hoogeveen JA and Van de Velde SL (1999). Parallel machine scheduling by column generation. *Opns Res* **47**: 862–872.
- Van Wassenhove LN (1979). *Special purpose algorithms for one-machine sequencing problems with single and composite objectives*. PhD thesis, Katholieke Universiteit Leuven, Belgium.
- Widmer M and Hertz A (1989). A new heuristic method for the flow shop sequencing problem. *Eur J Opl Res* **41**: 186–193.
- Wilkerson L and Irwin J (1971). An improved method for scheduling independent tasks. *AIIE Trans* **33**: 239–245.
- Williamson DP, Hall LA, Hoogeveen JA, Hurkens CAJ, Lenstra JK, Sevastianov SV and Shmoys DB (1997). Short shop schedules. *Opns Res* **45**: 288–294.
- Yamada T and Nakano R (1992). A genetic algorithm applicable to large-scale job-shop problems. In: Männer R and Manderick B (eds). *Parallel Problem Solving from Nature 2*. North Holland: Amsterdam, pp 281–290.
- Yamada T, Rosen BE and Nakano R (1994). A simulated annealing approach to job shop scheduling using critical block transition operators. *Proceedings of the IEEE International Conference on Neural Networks, ICNN'94*. IEEE: New York, pp 4687–4692.

Received July 2008;
accepted December 2008 after one revision