



ELSEVIER

European Journal of Operational Research 120 (2000) 228–249

EUROPEAN  
JOURNAL  
OF OPERATIONAL  
RESEARCH

www.elsevier.com/locate/orms

Invited Review

## Scheduling with batching: A review

Chris N. Potts <sup>a,\*</sup>, Mikhail Y. Kovalyov <sup>b</sup>

<sup>a</sup> Faculty of Mathematical Studies, University of Southampton, Southampton SO17 1BJ, UK

<sup>b</sup> Institute of Engineering Cybernetics, National Academy of Sciences of Belarus, Surganova 6, 220012 Minsk, Belarus

---

### Abstract

There is an extensive literature on models that integrate scheduling with batching decisions. Jobs may be batched if they share the same setup on a machine. Another reason for batching occurs when a machine can process several jobs simultaneously. This paper reviews the literature on scheduling with batching, giving details of the basic algorithms, and referencing other significant results. Special attention is given to the design of efficient dynamic programming algorithms for solving these types of problems. © 2000 Elsevier Science B.V. All rights reserved.

*Keywords:* Scheduling; Batching; Families; Review; Complexity; Algorithm; Dynamic programming; Approximation

---

### 1. Introduction

In the last decade, there has been significant interest in scheduling problems that involve an element of batching. In this context, the motivation for batching jobs is a gain in efficiency: it may be cheaper or faster to process jobs in a batch than to process them individually.

One situation where benefits may result from batching occurs when machines require setups if they are to process jobs that have differing characteristics. The setup may reflect the need to change a tool or to clean the machine. In a *family scheduling model*, jobs are partitioned into families according to their similarity, so that no setup is required for a job if it belongs to the same family

of the previously processed job. However, a setup time is required at the start of the schedule and on each occasion when the machine switches from processing jobs in one family to jobs in another family. In this model, a *batch* is a maximal set of jobs that are scheduled contiguously on a machine and share a setup. Large batches have the advantage of high machine utilization because the number of setups is small. On the other hand, processing a large batch may delay the processing of an important job belonging to a different family.

There are two variants of the family scheduling model depending on when the jobs become available (either for dispatch to the customer or to be processed on the next machine). Under *batch availability*, a job only becomes available when the complete batch to which it belongs has been processed. For example, this situation occurs if the jobs in a batch are placed on a pallet, and the pallet is only moved from the machine when all of

---

\*Corresponding author. Tel.: +44 1703 593 651; fax: +44 1703 595 147; e-mail: cnp@maths.soton.ac.uk

these jobs are processed. An alternative assumption is *job availability* (usually known in the literature as item availability), in which a job becomes available immediately after its processing is completed. Unless stated otherwise, we adopt the assumption of job availability.

Another situation where batching may result in improved efficiency occurs when a *batching machine* is capable of processing several jobs simultaneously. For example, ‘burn in’ operations in the manufacture of circuit boards are performed in ovens that can accommodate several jobs. Other applications occur for chemical processes that are performed in tanks or kilns. In these cases, a batching machine processes a batch of jobs at the same time, where there is sometimes an upper limit on the batch size.

There are reviews of models which combine scheduling with batching by Potts and Van Wassenhove [63] and Webster and Baker [79]. In this paper, we provide an updated review which includes many of the more recent results in this area. Our review provides a classification of problems as polynomially or pseudopolynomially solvable, binary or unary NP-hard (NP-hard in the ordinary or strong sense), or open. Also, we describe enumerative algorithms, heuristic procedures and local search methods, and indicate their efficiency and effectiveness. Moreover, our review covers approximation algorithms and their worst-case performance.

The remainder of this paper is organized as follows. In Section 2, we give a more formal definition of the models under consideration. Section 3 explains how dynamic programming algorithms can be designed to give efficient algorithms for several of the problems that we consider. Sections 4 and 5 review the family scheduling model and the batching machine model, respectively. Finally, some concluding remarks are contained in Section 6.

## 2. Problem specification

We consider problems involving the scheduling of a single machine, of  $m$  parallel machines, and of  $m$ -machine flow shops, job shops and open shops.

For the case of parallel machines, the machines may either be identical, uniform, or unrelated, which means that the processing time of a job on a machine depends only on the job, depends only on the job and the speed of the machine to which it is assigned, or depends both on the job and the machine to which it is assigned, respectively. For the flow shop, each job has  $m$  operations, the first of which requires processing on machine 1, the second requires processing on machine 2, and so on, and the last requires processing on machine  $m$ . For the job shop, each job has a given sequence of operations on the machines, so that different jobs pass through the machines in a different order. For the open shop, each job has an operation on each of the  $m$  machines, but the processing order of these operations can be chosen.

Each machine may either be of a classical type that can handle at most one job at a time, or it may be a batching machine that can process several jobs simultaneously. For a batching machine, the processing time of a batch is the maximum of the processing times of jobs or operations contained in the batch. Further, each job or operation in the batch has the same completion time, which is the time that the batch completes processing.

Let  $\{1, \dots, n\}$  denote the set of jobs to be processed. For a single-machine problem, we denote the processing time of job  $j$ , for  $j = 1, \dots, n$ , by  $p_j$ . (For other machine environments, we do not need to introduce notation for processing times.) Other parameters of job  $j$  that occur in some problems include a release date  $r_j$ , a deadline  $\bar{d}_j$ , a due date  $d_j$ , and a weight  $w_j$ .

In the family scheduling model, the jobs are partitioned into  $F$  families. Let  $n_f$  denote the number of jobs in family  $f$ , for  $f = 1, \dots, F$ . No setup is required between jobs of the same family. However, the *family setup time* on machine  $i$  when a job of family  $g$  is immediately preceded by a job of a different family  $f$  is  $s_{ifg}$ , or  $s_{i0g}$  if there is no preceding job. If, for each  $g$ , we can write  $s_{ifg} = s_{i0g} = s_{ig}$  for all  $f \neq g$ , then the setup times on machine  $i$  are *sequence independent*; otherwise, they are *sequence dependent*. If, for each machine  $i$ ,  $s_{ifg} = s_{jfg}$  for all families  $f$  and  $g$  including the case  $f = 0$ , then the setup times are *machine independent*; otherwise, they are *machine dependent*. For

the case of a single machine, setup times are, by definition, machine independent. Further, we make the reasonable assumption that the *triangle inequality* holds for each machine  $i$ , which means that  $s_{ifh} \leq s_{ifg} + s_{igh}$ , for all distinct families  $f, g$  and  $h$ , including the case  $f = 0$ . Unless stated otherwise, the setups are assumed to be *anticipatory*, which means that a setup on a machine does not require the presence of any job. When there are release dates and for shop problems, we sometimes allow setups to be *non-anticipatory*, which means that the setup preceding the processing of some batch cannot start on the current machine before all jobs of this batch are released and have completed their processing on any previous machine.

For the case of sequence-independent family setup times and batch availability, there may be two or more successive batches of the same family. In such a case, a family setup time is required before a batch is processed, even it is of the same family as the previous batch.

If there is a maximum batch size for any machine  $i$ , we denote it  $b_i$ . In the case of a single machine, or if restrictions on batch sizes are the same for all machines, we let  $b$  denote any maximum batch size that is imposed.

We define the following variables for each job  $j$  in a schedule:

- $C_j =$  the completion time of job  $j$ ;
- $L_j = C_j - d_j$ , the lateness of job  $j$ ;
- $U_j = \begin{cases} 1, & \text{if job } j \text{ is late, i.e., } C_j > d_j, \\ 0, & \text{if job } j \text{ is early, i.e., } C_j \leq d_j; \end{cases}$
- $T_j = \max\{C_j - d_j, 0\}$ , the tardiness of job  $j$ .

The standard classification scheme for scheduling problems (Graham et al. [38]) is  $\psi_1|\psi_2|\psi_3$ , where  $\psi_1$  indicates the scheduling environment,  $\psi_2$  describes the job and family characteristics and any restrictive requirements, and  $\psi_3$  defines the objective function to be minimized.

We let  $\psi_1 = \alpha m$ , where  $m$  is the number of machines,  $\alpha \in \{P, Q, R, F, J, O\}$  for classical machines in identical, uniform and unrelated parallel-machine, flow shop, job shop and open shop environments, and  $\alpha \in \{\bar{P}, \bar{Q}, \bar{R}, \bar{F}, \bar{J}, \bar{O}\}$  for batching

machines in identical, uniform and unrelated parallel-machine, flow shop, job shop and open shop environments, respectively. If  $m$  is not shown, then the number of machines is arbitrary. In the case of a single machine, we omit  $\alpha$ , and  $m = 1$  denotes a classical machine and  $m = \bar{1}$  denotes a batching machine. Under  $\psi_2$ , we may have

- $r_j$  each job  $j$  has a release date  $r_j$ ;
- $\bar{d}_j$  each job  $j$  has a deadline  $\bar{d}_j$ ;
- pmtn preemption of jobs is allowed;
- $p_j = p$  all jobs have a common processing time  $p$ ;
- $\bar{d}_j = \bar{d}$  all jobs have a common deadline  $\bar{d}$ ;
- $d_j = d$  all jobs have a common due date  $d$ ;
- $b_i$  the maximum batch size on machine  $i$  is  $b_i$ ;
- $b$  the maximum batch size on all machines is  $b$ ;
- $F = k$  the number of families is a constant  $k$ ;
- $n_f = n/F$  all families have the same number of jobs  $n/F$ ;
- $s_{ifg}$  there are general family setup times;
- $s_{fg}$  there are machine-independent family setup times;
- $s_{if}$  there are sequence-independent family setup times;
- $s_f$  there are machine- and sequence-independent family setup times;
- $s_f = s$  all families have a common (machine- and sequence-independent) setup time  $s$ .

The objectives that we consider under  $\psi_3$  require finding a feasible schedule such that one of the following cost functions is minimized:

- $\cdot$  the constant cost function, which is minimized by any feasible schedule;
- $C_{\max}$   $\max_{j=1, \dots, n} \{C_j\}$ , the completion time of the last job, or the makespan;
- $\sum(w_j)C_j$  the total (weighted) completion time of the jobs;
- $L_{\max}$   $\max_{j=1, \dots, n} \{L_j\}$ , the maximum lateness of the jobs;
- $\sum(w_j)U_j$  the (weighted) number of late jobs;
- $\sum(w_j)T_j$  the total (weighted) tardiness of the jobs.

An example of the classification scheme is problem  $\tilde{F}2|b_1 = 1, b_2 = 2|C_{\max}$ , which denotes the minimization of makespan in a two-machine flow shop, where the first machine is a classical machine, and the second is a batching machine that can process up to two jobs at the same time. Another example is problem  $1|s_{fg}|\sum C_j$ , which denotes the minimization of total completion time on a single (classical) machine, where there are job families and sequence-dependent family setup times.

Throughout the paper, we assume that all data that define a problem instance are integers.

### 3. Dynamic programming

For many of the family scheduling and batching machine models with a single machine that we consider, the sequencing and batching aspects of the problem can be decoupled. More precisely, analysis may show that there exists an optimal schedule in which the jobs within a family appear in a certain sequence. An even stronger result can be established for some single-family models under batch availability and batching machine models: there exists an optimal schedule in which all jobs appear in a given sequence. In both cases, *dynamic programming* is often found to be a useful technique for partitioning the jobs (of a family) into batches.

We illustrate the use of dynamic programming for the case that the jobs are to be sequenced in the order  $(1, \dots, n)$  (possibly after reindexing). To construct a schedule, either a single job or a batch is added to some previous (partial) schedule. In a *forward* algorithm, job  $j$  is appended to a previous schedule (either placing it in the same batch as job  $j-1$ , or starting a new batch) comprising jobs  $1, \dots, j-1$ , or a batch containing jobs  $i+1, \dots, j$  is appended to a previous schedule comprising jobs  $1, \dots, i$ , where  $0 \leq i < j \leq n$ . On the other hand, a *backward* algorithm either inserts job  $j$  at the start of some previous schedule (either placing it in the same batch as job  $j+1$ , or starting a new batch with job  $j+1$  so that job  $j$  ends the previous batch) comprising jobs  $j+1, \dots, n$ , or a batch contain-

ing jobs  $j, \dots, k-1$  is inserted at the start of a previous schedule comprising jobs  $k, \dots, n$ , where  $1 \leq j < k \leq n+1$ . Below we discuss the relative merit of forward and backward algorithms. However, the choice between appending/inserting jobs or batches tends to be problem specific, and it is difficult to provide any general guidelines.

Consider a forward recursion where job  $j$  is appended to some previous schedule comprising jobs  $1, \dots, j-1$ . (Similar comments apply if a batch, rather than a job, is appended.) To evaluate the objective function contribution of job  $j$ , it is necessary to know when this job is completed. This can be achieved by including, as a state variable, the time  $t$  that the schedule comprising jobs  $1, \dots, j-1$  completes. In general, this will produce an algorithm with pseudopolynomial time complexity. In some cases, knowledge of numbers of setups allows  $t$  to be computed, thus enabling replacement of the state variable  $t$  to obtain a polynomial time complexity.

Alternatively, consider a backward recursion in which job  $j$  is inserted at the start of some previous schedule comprising jobs  $j+1, \dots, n$ , where a setup or processing for this schedule starts at time zero. (Similar comments apply to the case that a batch is inserted.) Let  $\tau$  be the delay to the completion of jobs  $j+1, \dots, n$  that is caused by the insertion. For the  $\sum w_j C_j$  objective, the total weighted completion time of jobs  $j+1, \dots, n$  is increased by  $\tau \sum_{h=j+1}^n w_h$ , and for the  $L_{\max}$  objective, the maximum lateness of jobs  $j+1, \dots, n$  is increased by  $\tau$ , as a result of the insertion. For other objectives, the effect of an insertion may be impossible to evaluate without knowledge of the schedule.

To summarize, problems involving the  $\sum w_j C_j$  or  $L_{\max}$  objectives are amenable to a backward dynamic programming algorithm. On the other hand, problems with the  $\sum (w_j) U_j$  or  $\sum (w_j) T_j$  objectives are best tackled with a forward recursion. For the  $\sum U_j$  objective, a gain in efficiency can usually be achieved by choosing the number of late jobs as a state variable in a forward algorithm, so that the makespan of the schedule of early jobs is stored as a function value.

### 4. Family scheduling models

In the first four subsections, we consider family scheduling models under the assumption of job availability. Sections 4.1–4.3 deal with single-machine, parallel-machine and shop problems, respectively, and Section 4.4 addresses problems where each family contains identical jobs that may differ among families. Batch availability problems are considered in Section 4.5. Sections 4.6 and 4.7 discuss single-machine problems in which jobs are delivered to customers in batches, and in which each job comprises several operations in different families, respectively.

#### 4.1. Single machine

Monma and Potts [59] study family scheduling problems for a single machine with various objective functions. Some of their algorithms have been improved, and our review gives the most efficient algorithm that is known. In the description of these algorithms, it is useful to denote the  $j$ th job of each family  $f$  ( $f = 1, \dots, F$ ) by the pair  $(j, f)$ , for  $j = 1, \dots, n_f$ .

##### 4.1.1. Total weighted completion time

Monma and Potts [59] show that, for problem  $1|s_{fg}| \sum w_j C_j$ , there exists an optimal schedule in which the SWPT rule of Smith [66] applies within each family  $f$  (jobs are sequenced in non-decreasing order of  $p_{(j,f)}/w_{(j,f)}$ ). Since jobs within each family  $f$  are sequenced in SWPT order, we reindex the jobs so that

$$p_{(1,f)}/w_{(1,f)} \leq \dots \leq p_{(n_f,f)}/w_{(n_f,f)}.$$

Ghosh [35] proposes a backward dynamic programming algorithm with job insertion for problem  $1|s_{fg}| \sum w_j C_j$ . Let  $G(q_1, \dots, q_F, g)$  denote the minimum total weighted completion time for schedules containing jobs  $(q_f, f), \dots, (n_f, f)$  for  $f = 1, \dots, F$ , where job  $(q_g, g)$  is the first job to be processed, and its processing starts at time zero. Note that  $q_g \leq n_g$ , and the setup for the batch of jobs of family  $g$  at the start of the schedule is not currently included. The initialization for  $f = 1, \dots, F$  is

$$G(n_1 + 1, \dots, n_F + 1, f) = 0$$

and the recursion for  $q_f = n_f + 1, n_f, \dots, 1$  and  $f = 1, \dots, F$ , and  $g = 1, \dots, F$ , where  $q_g \neq n_g + 1$ , is

$$G(q_1, \dots, q_F, g) = \min_{g'=1, \dots, F} \left\{ G(q'_1, \dots, q'_F, g') + (p_{(q_g, g)} + s_{g, g'}) \sum_{f=1}^F \sum_{j=q_f}^{n_f} w_{(j, f)} - s_{g, g'} w_{(q_g, g)} \right\},$$

where  $q'_f = q_f$  for  $f \neq g$ ,  $q'_g = q_g + 1$  and  $s_{g, g'} = 0$  if  $g = g'$ . The minimization selects a previous schedule into which job  $(q_g, g)$  is inserted at the start. If the first job of the previous schedule is from family  $g$ , then this previous schedule is delayed by time  $p_{(q_g, g)}$ . On the other hand, if the first job of the previous schedule is from family  $g'$ , where  $g' \neq g$ , then the delay is equal to  $p_{(q_g, g)} + s_{g, g'}$ , since the first job in the previous schedule starts a batch and the associated setup is also inserted. The optimal solution value is then equal to

$$\min_{g=1, \dots, F} \left\{ G(1, \dots, 1, g) + s_{0, g} \sum_{f=1}^F \sum_{j=1}^{n_f} w_{(j, f)} \right\},$$

where the final term in the minimization accounts for the delay that results from inserting the relevant setup at the beginning of the schedule. The time complexity of the algorithm is  $O(n^F)$ , which is polynomial for fixed  $F$ . For problem  $1|s_{fg}| \sum C_j$ , Ahn and Hyun [2] propose a forward dynamic programming algorithm with job appending that also requires  $O(n^F)$  time.

Rinnooy Kan [67] shows that problem  $1|s_{fg}| \sum C_j$  is unary NP-hard for arbitrary  $F$ . However, the complexity status of  $1|s_f| \sum C_j$  and  $1|s_f| \sum w_j C_j$  is open for arbitrary  $F$ .

Mason and Anderson [57] and Crauwels et al. [24] propose branch and bound algorithms for problem  $1|s_f| \sum w_j C_j$ . Mason and Anderson use a forward branching rule, and make extensive use of dominance rules to restrict the size of the branch and bound search tree. Their lower bound is derived using objective splitting: the total weighted

completion time can be partitioned into contributions from the processing times and from the setup times, which are optimized separately. Crauwels et al. obtain a lower bound by performing a Lagrangean relaxation of the machine capacity constraints in a time-indexed formulation of the problem. Values of the multipliers are obtained either by a constructive ‘multiplier adjustment’ method or by subgradient optimization. The first of their algorithms uses a forward branching rule that incorporates numerous dominance rules, and the multiplier adjustment method for obtaining lower bounds. Their second algorithm uses a binary branching rule that fixes adjacent jobs (with respect to the SWPT ordering) in the same family to be in the same or in different batches, and subgradient optimization for computing lower bounds. Computational results show that the algorithm which uses forward branching rule and the multiplier adjustment method solves instances with up to 70 jobs, and is more efficient than both Mason and Anderson’s algorithm, and the algorithm which uses binary branching and subgradient optimization.

For problem  $1|s_f|\sum C_j$ , heuristics are proposed by Gupta [39], Williams and Wirth [80] and Ahn and Hyun [2]. Gupta’s heuristic builds schedules by successively appending jobs, where each job is chosen so that it has the smallest completion time. A similar approach is adopted by Williams and Wirth, but they use the dominance rules of Mason and Anderson [57] to eliminate some candidates, and allow certain interchanges of batches and backward insertions of jobs into previous batches. Ahn and Hyun propose a descent heuristic in which sub-batches are moved to earlier or later positions in the sequence of batches. Computational results show that the heuristics of Williams and Wirth, and Ahn and Hyun generate better quality solutions than those obtained with Gupta’s heuristic. An alternative heuristic approach, as suggested by Liao and Liao [55] for a more general problem, is to introduce complete families one by one into the current schedule using dynamic programming.

There are two studies that develop local search heuristics for problem  $1|s_f|\sum w_j C_j$ . Mason [56] designs a genetic algorithm from the observation

that knowledge of the first job in each batch enables a solution to be constructed by ordering the batches using a generalization of the SWPT rule. Thus, he uses a binary representation of solutions, where each element in the representation indicates whether or not the corresponding job starts a batch, to which standard genetic operators are applied. Crauwels et al. [27] develop several neighborhood search heuristics (descent, simulated annealing, threshold accepting and tabu search). They use a neighborhood that selects a sub-batch of jobs at the beginning (end) of a batch and moves this sub-batch to an earlier (later) position in the sequence. The temperature in simulated annealing follows a periodic pattern, and a descent algorithm is applied before each temperature change. Threshold accepting is applied in an analogous way to simulated annealing. In their tabu search method, sub-batches are restricted to contain a single job, and a limited reordering of batches according to a SWPT rule applied to batches is used. Computational tests for problems with 40, 50, 70 and 100 jobs, and 4, 6, 8 and 10 families, show that all local search methods generate solutions which are close in value to the optimum. The best results are obtained with a multi-start version of tabu search when the number of families is small, and with Mason’s genetic algorithm when the number of families is large.

Herrmann and Lee [42] propose a genetic algorithm for problem  $1|s_{fg}, \bar{d}_j|\sum C_j$ . They use a representation which consists of a binary encoding of perturbations of the original deadlines. To obtain a solution from the representation, a backward scheduling heuristic is used which aims to minimize the time spent on setups and to process the longer jobs as late as possible. Since the resulting schedule is not guaranteed to be feasible with respect to the original (or the perturbed) deadlines, a penalty function approach is used to drive the solution towards feasibility.

#### 4.1.2. Maximum lateness

A dynamic programming algorithm for problem  $1|s_{fg}|L_{\max}$  can be developed using similar arguments to those for problem  $1|s_{fg}|\sum w_j C_j$ . The key property, which is derived by Monma and Potts [59], shows that there exists an optimal

schedule in which the EDD rule of Jackson [47] applies within each family  $f$  (jobs are sequenced in non-decreasing order of  $d_{(j,f)}$ ). Since jobs within each family  $f$  are sequenced in EDD order, we reindex the jobs so that  $d_{(1,f)} \leq \dots \leq d_{(n_f,f)}$ .

A backward dynamic programming algorithm with job insertion is proposed by Ghosh and Gupta [36] for problem  $1|s_{fg}|L_{\max}$ . Let  $G(q_1, \dots, q_F, g)$  denote the minimum value of the maximum lateness for schedules containing jobs  $(q_f, f), \dots, (n_f, f)$  for  $f = 1, \dots, F$ , where job  $(q_g, g)$  is processed first starting at time zero, and the setup for the batch of jobs of family  $g$  at the start of the schedule is not currently included. The initialization for  $f = 1, \dots, F$  is

$$G(n_1 + 1, \dots, n_F + 1, f) = -\infty$$

and the recursion for  $q_f = n_f + 1, n_f, \dots, 1$  and  $f = 1, \dots, F$ , and  $g = 1, \dots, F$ , where  $q_g \neq n_g + 1$ , is

$$\begin{aligned} &G(q_1, \dots, q_F, g) \\ &= \min_{g'=1, \dots, F} \left\{ \max \{ G(q'_1, \dots, q'_F, g') + p_{(q_g, g)} \right. \\ &\quad \left. + s_{g, g'}, p_{(q_g, g)} - d_{(q_g, g)} \} \right\}, \end{aligned}$$

where  $q'_f = q_f$  for  $f \neq g$ ,  $q'_g = q_g + 1$ , and  $s_{g, g'} = 0$  if  $g = g'$ . The optimal solution value is then equal to

$$\min_{g=1, \dots, F} \{ G(1, \dots, 1, g) + s_{0, g} \}.$$

The time complexity of the algorithm is  $O(n^F)$ , which is polynomial for fixed  $F$ .

Bruno and Downey [8] show that, for arbitrary  $F$ , problem  $1|s_f|L_{\max}$  is binary NP-hard, even if there are two jobs in each family, or all setup times are one unit and there are three jobs in each family. This problem is open with respect to unary NP-hardness.

Hariri and Potts [40] propose a branch and bound algorithm for problem  $1|s_f|L_{\max}$ . They obtain an initial lower bound by ignoring setups, except for those associated with the first job in each family, and solving the resulting problem with the EDD rule. This lower bound is improved by a procedure that considers whether or not certain families are split into two or more batches. A binary branching rule fixes adjacent jobs (with

respect to the EDD ordering) in the same family to be either in the same or in different batches. Computational results show that the algorithm is successful in solving instances with up to about 60 jobs. Schutten et al. [65] develop a branch and bound algorithm for problem  $1|s_f, r_j|L_{\max}$ . In the presence of release dates, no results are known about the order of jobs within a family. A key component of their algorithm is the use of dummy jobs to represent setups. Quickly computed lower bounds are obtained by relaxing setups and solving the corresponding preemptive problem, and a forward branching rule is used. Computational results show that the algorithm is effective in solving instances with up to about 40 jobs.

Hariri and Potts [40], Zdrzałka [83] and Woeginger [81] design approximation algorithms for problem  $1|s_f|L_{\max}$  under the assumption that due dates are non-positive (to ensure that the objective function is positive). The algorithm of Hariri and Potts selects the better of two schedules, where the first schedule assigns all jobs of a family to a single batch, and the second schedule splits each family into at most two batches according to the due dates. This algorithm requires  $O(n \log n)$  time, and it generates a schedule with maximum lateness that is no more than  $5/3$  times the optimal value. Zdrzałka's algorithm has a better performance guarantee at the expense of extra computation. The algorithm starts with a schedule in which each batch contains all jobs from a family, and allows each family to be split into at most two batches. At each iteration, a job is removed from the first batch of its family and is inserted into the second batch. The algorithm requires  $O(n^2)$  time, and it generates a schedule with maximum lateness that is no more than  $3/2$  times the optimal value. Moreover, these performance bounds of  $5/3$  and  $3/2$  are tight. Zdrzałka observes that his algorithm can be adapted to problem  $1|s_f, r_j|L_{\max}$  to generate a schedule with maximum lateness that is no more than  $5/2$  times the optimal value. Woeginger derives a *polynomial-time approximation scheme*, which is a family of approximation algorithms  $\{A_\epsilon\}$  with the property that, for any  $\epsilon > 0$ , algorithm  $A_\epsilon$  generates a schedule with maximum lateness that is no more than  $1 + \epsilon$  times the optimal value, and has a time requirement that is

polynomial in the size of the input (but may be exponential in  $1/\epsilon$ ).

### 4.1.3. Weighted number of late jobs

Monma and Potts [59] develop a forward dynamic programming algorithm with job appending for problem  $1|s_{fg}|\sum w_j U_j$ . This algorithm uses the property that there exists an optimal schedule in which the early jobs within each family are sequenced in EDD order. Assume that the jobs within each family  $f$  are reindexed so that  $d_{(1,f)} \leq \dots \leq d_{(n_f,f)}$ .

We present the algorithm of Monma and Potts for problem  $1|s_{fg}|\sum U_j$ . Let  $G(q_1, \dots, q_F, u, g)$  denote the minimum makespan for schedules containing early jobs selected from  $(1, f), \dots, (q_f, f)$  for  $f = 1, \dots, F$ , where  $u$  is the number of late jobs, and the last (early) job in the schedule is from family  $g$ . Note that  $q_g \geq 1$ . In a forward dynamic programming algorithm with job appending, the initialization for  $q_f = 0, 1, \dots, n_f$  and  $f = 1, \dots, F$ , and  $u = 0, 1, \dots, \sum_{f=1}^F q_f$ , is

$$G(q_1, \dots, q_F, u, 0) = \begin{cases} 0, & \text{for } u = \sum_{f=1}^F q_f, \\ \infty, & \text{otherwise,} \end{cases}$$

and the recursion for  $q_f = 0, 1, \dots, n_f$  and  $f = 1, \dots, F$ ,  $u = 0, 1, \dots, \sum_{f=1}^F q_f$ , and  $g = 1, \dots, F$ , where  $q_g \neq 0$ , is

$$G(q_1, \dots, q_F, u, g) = \min \left\{ G(q'_1, \dots, q'_F, u - 1, g), \min_{g' \in H(q_1, \dots, q_F, u, g)} \{ G(q'_1, \dots, q'_F, u, g') + \tau \} \right\},$$

where  $q'_f = q_f$  for  $f \neq g$ ,  $q'_g = q_g - 1$ ,  $\tau = s_{g',g} + p_{(q_g,g)}$ ,  $s_{g',g} = 0$  if  $g = g'$ , and  $H(q_1, \dots, q_F, u, g) = \{g' | g' \in \{0, 1, \dots, f\}, G(q'_1, \dots, q'_F, u, g') + \tau \leq d_{(q_g,g)}\}$ . The minimum number of late jobs is then equal to the smallest value  $u$  for which  $\min_{f=0,1,\dots,F} \{G(n_1, \dots, n_F, u, f)\} < \infty$ . The time complexity of the algorithm is  $O(n^{F+1})$ , which is polynomial for fixed  $F$ .

Problem  $1|s_{fg}|\sum w_j U_j$  is pseudopolynomially solvable for fixed  $F$ . This result follows from the observation that the above dynamic programming algorithm generalizes to minimize the weighted

number of late jobs in  $O(n^F W)$  time, where  $W = \sum_{j=1}^n w_j$ . An alternative dynamic programming algorithm in which the completion time of the schedule of early jobs is a state variable, and the weighted number of late jobs is a function value, has a pseudopolynomial time complexity of  $O(n^F \min\{d_{\max}, P\})$ , where  $d_{\max} = \max_{j=1,\dots,n} \{d_j\}$  and  $P = \sum_{j=1}^n p_j + \sum_{g=1}^F n_g \max_{f=0,1,\dots,F} \{s_{fg}\}$ .

The binary NP-hardness of problem  $1|s_f|\sum U_j$  for arbitrary  $F$  follows from the corresponding result for problem  $1|s_f|L_{\max}$ . However, Rote and Woeginger [64] show that if all jobs in the same family have a common due date, then problem  $1|s_f|\sum U_j$  is solvable in  $O(n^2)$  time by dynamic programming. Problems  $1|s_f|\sum U_j$  and  $1|s_f|\sum w_j U_j$  are open with respect to unary NP-hardness.

Crauwels et al. [25] propose branch and bound algorithms for problem  $1|s_f|\sum U_j$ . They develop two lower bounding schemes. In the first of these schemes, the setup times are relaxed, except for one setup time for each family which is distributed among the job processing times, and the resulting problem is solved using the algorithm of Moore [61]. The second lower bounding scheme uses due date relaxation. After setting each due date to be equal to  $d_{\max}$ , the largest among all original due dates, the resulting problem is solved by dynamic programming. These lower bound are incorporated in branch and bound algorithms that either use a standard forward branching rule, or use ternary branching. In the ternary branching rule, each job is either set to be late, set to be early and start a batch, or set to be early and not start a batch. Forward branching has the advantage of allowing numerous dominance rules to be applied, although ternary branching is more flexible. Computational results for the different combinations of lower bounding schemes and branching rules show that the most efficient algorithm has a lower bound based on setup time relaxation and uses forward branching. Moreover, this algorithm is successful in solving instances with up to about 50 jobs.

Crauwels et al. [26] propose multi-start versions of descent, simulated annealing and tabu search, and a genetic algorithm, for problem  $1|s_f|\sum U_j$ . The neighborhood search algorithms use either a

job or batch neighborhood. In the job neighborhood, a job is removed from the sequence of early jobs, and an attempt is made to insert one or more late jobs into the resulting sequence so that all jobs are early. The batch neighborhood is similar except that a complete batch is removed and then insertions of late jobs are attempted. The genetic algorithm uses a representation in which two binary elements are associated with each job, one indicating whether the job is early or late, and the other indicating whether the job ends a batch. To obtain a corresponding schedule of early jobs, a due date is associated with each batch of early jobs, and the batches are sequenced in EDD order. If the resulting solution is infeasible, the algorithm of Moore [61] is applied to remove the smallest number of batches. In computational tests with 30, 40 and 50 jobs, and 4, 6, 8 and 10 families, all heuristics perform well. The best quality solutions are obtained with a version of the genetic algorithm, which includes a procedure that attempts to improve each solution of the final population.

#### 4.2. Parallel machines

As observed by Monma and Potts [59], the orderings of jobs within each family that are used to derive dynamic programming algorithms for problems  $1|s_{fg}|\sum w_j C_j$ ,  $1|s_{fg}|L_{\max}$  and  $1|s_{fg}|\sum (w_j)U_j$  also apply to each machine for identical parallel-machine scheduling. By including state variables to store the total setup plus processing time on each machine, forward dynamic programming algorithms with job appending can be developed for problems  $P|s_{fg}|\sum w_j C_j$ ,  $P|s_{fg}|L_{\max}$  and  $P|s_{fg}|\sum (w_j)U_j$ , which require pseudopolynomial time for fixed  $m$  and  $F$ .

Most NP-hardness results for scheduling an arbitrary number of families of jobs on parallel machines are deduced from the corresponding results for classical parallel-machine scheduling. An exception is for problem  $P|s_f|\sum C_j$ , which Webster [77] shows to be unary NP-hard. However, the complexity of the corresponding problem with a fixed number of machines is open.

Monma and Potts [59] show that problem  $P2|s_f, \text{pmtn}|C_{\max}$  is binary NP-hard. In another

paper [60], they propose an approximation algorithm for problem  $P|s_f, \text{pmtn}|C_{\max}$  that resembles McNaughton's algorithm [58] for the classical scheduling problem  $P|\text{pmtn}|C_{\max}$ . This algorithm requires  $O(n)$  time, and it generates a schedule with makespan that is no more than  $2 - 1/(\lfloor m/2 \rfloor + 1)$  times the optimal makespan. For a special class of instances in which the setup plus total processing time for each single family does not exceed the optimal makespan, Monma and Potts [60] and Chen [10] show that this performance can be improved through the use of an approximation algorithm that first uses list scheduling for complete families, and then splits families between selected pairs of machines. In particular, Chen's algorithm requires  $O(n \log n)$  time, and it generates a schedule with makespan that is no more than  $\max\{3m/(2m+1), (3m-4)/(2m-2)\}$  times the optimal makespan. Each of these performance bounds is tight.

#### 4.3. Shop problems

For the classical two-machine flow shop problem  $F2||C_{\max}$ , there exists an optimal permutation schedule (the job sequence is the same on all machines). However, the issue of whether this property extends to problem  $F2|s_{ifg}|C_{\max}$  remains unresolved. Potts and Van Wassenhove [63] observe that, for anticipatory setups, the search for an optimal permutation schedule may be restricted to schedules in which the jobs within each family are sequenced according to the algorithm of Johnson [48]. In this case, similar arguments to those in Sections 4.1.1 and 4.1.2 can be used to develop a backward dynamic programming algorithm with job insertion, which finds an optimal permutation schedule in  $O(n^F)$  time.

Kleinau [50] shows that problem  $F2|s_{if}|C_{\max}$ , with machine-dependent and sequence-independent setup times, is binary NP-hard for both anticipatory and non-anticipatory setups. However, this problem is open if setup times are machine independent. Kleinau also proves binary NP-hardness of problem  $O2|s_f|C_{\max}$ , even if there are only two families. For other objective functions, NP-hardness results for scheduling an arbitrary

number of families are deduced from the corresponding results for classical shop scheduling.

Chen et al. [11] propose two approximation algorithms for problem  $F2|s_{if}|C_{\max}$ , each of which requires  $O(n \log n)$  time. The first algorithm assigns all jobs of a family to a single batch, and then schedules the batches. This schedule has makespan that is no more than  $3/2$  times the optimal makespan. The second algorithm uses properties of the schedule created by the first algorithm to generate another schedule by splitting each family into at most two batches. The better of the two schedules has makespan that is no more than  $4/3$  times the optimal makespan. Moreover, these performance bounds of  $3/2$  and  $4/3$  are tight.

Vakharia and Chang [74] and Skarin-Kapov and Vakharia [71] propose simulated annealing and tabu search methods, respectively, for the permutation flow shop problem  $F|s_{if}|C_{\max}$ , but restrict their attention to schedules in which each family is processed as a single batch. Sotskov et al. [72] compare heuristics for the permutation flow shop problems  $F|s_{if}|C_{\max}$  and  $F|s_{if}|\sum C_j$  in which the search is restricted to schedules with the same batches and with the same processing order of batches on each machine. They develop constructive heuristics that are based on the successive insertion of jobs into the sequence, and local search methods (simulated annealing, threshold accepting and tabu search). Their neighbourhood is similar to that used by Crauwels et al. [27] (see Section 4.1.1). In computational tests with 40, 60 and 80 jobs, and 5 and 10 machines, the best results are obtained with simulated annealing and tabu search, with the former providing slightly better quality solutions.

#### 4.4. Identical jobs in each family

In this subsection, setups are assumed to be sequence and machine independent. Moreover, family  $f$  contains  $n_f$  identical jobs, each with processing time  $p_f$ , and where appropriate deadline  $\bar{d}_f$ , due date  $d_f$ , and weight  $w_f$ , for  $f = 1, \dots, F$ .

For problems  $1|s_f|\sum w_f C_j$  and  $1|s_f|L_{\max}$ , Dobson et al. [29] and Santos [68], respectively, show that there exists an optimal schedule in which no

family is split. Further, the batches that are formed from complete families of jobs are sequenced according to a generalization of the SWPT rule, and according to the EDD rule, respectively. Thus, these problems are both solvable in  $O(F \log F)$  time.

A variety of polynomial time algorithms and NP-hardness proofs are available for various special cases of problem  $1|s_f|\sum w_j U_j$ . Specifically, the problem is proved to be binary NP-hard for the following cases:  $1|s_f, p_j = 1, d_j = d|\sum U_j$  and  $1|s_f = s, d_j = d|\sum U_j$  [52]; and  $1|s_f = s, p_j = 1, d_j = d|\sum w_j U_j$  [51]. Moreover,  $O(F \log F)$  algorithms, in the same spirit as the algorithm of Moore [61], are available for the following cases:  $1|s_f = s, p_j = p|\sum U_j$  [52];  $1|s_f, n_f = n/F, d_j = d|\sum U_j$  and  $1|s_f = s, n_f = n/F, p_j = p|\sum U_j$  [16]. Also, problem  $1|s_f = s, n_f = n/F, p_j = p, d_j = d|\sum w_j U_j$  is solvable in  $O(n \log n)$  time. However, the complexity of some special cases with general due dates remains open.

Pseudopolynomial dynamic programming algorithms are proposed by Kovalyov et al. [52] and Cheng and Kovalyov [16] for problems  $1|s_f|\sum U_j$  and  $1|s_f|\sum w_j U_j$ , respectively. Using rounding techniques, both algorithms can be used to derive a *fully polynomial approximation scheme*, which is a family of approximation algorithms  $\{A_\epsilon\}$  with the property that, for any  $\epsilon > 0$ , algorithm  $A_\epsilon$  generates a schedule with weighted number of late jobs that is no more than  $1 + \epsilon$  times the optimal value, and has a time requirement that is polynomial in  $n$  and  $1/\epsilon$ . Based on the scheme of Kovalyov et al. [52], but improved by an idea of Hassin [41], algorithm  $A_\epsilon$  has a time complexity of  $O(n^3/\epsilon + n^3 \log \log n)$ . As a by-product of their analysis, Kovalyov et al. [52] also derive an  $O(n \log n)$  algorithm for the problem of minimizing the maximum number of late jobs in each family.

Problems  $P2|s_f, p_j = p|\sum C_j$  are  $P|s_f|\sum C_j$  are shown by Cheng and Chen [12] and Webster [77] to be binary and unary NP-hard, respectively. Due to different size of the input, the result for problem  $P2|s_f, p_j = p|\sum C_j$  does not imply NP-hardness of the corresponding problem without the restriction of identical jobs [78]. As indicated in Section 4.2, the complexity of problem  $P2|s_f|\sum C_j$  is open.

Brucker et al. [7] establish the computational complexity for various special cases of the unrelated parallel-machine problem  $R|s_f, \bar{d}_j|$  of finding a schedule that is feasible with respect to the deadlines. Specifically, problem  $P2|s_f = 1, p_j = 1, \bar{d}_j = \bar{d}|$  is binary NP-hard, whereas problems  $P|s_f = 1, p_j = 1, \bar{d}_j = \bar{d}|$  and  $Q|s_f = 1, n_f = n/F, p_j = 1, \bar{d}_j = 1|$  are unary NP-hard. The latter problem is solvable in  $O(m \cdot m!)$  time by a modification of the wrap around rule of McNaughton [58], and therefore is polynomially solvable for a fixed number of machines  $m$ . Moreover, polynomial solvability by this algorithm also applies to problem  $Qm|s_f = s, n_f = n/F, p_j = p, \bar{d}_j = \bar{d}|$  when  $s$  is a multiple of  $p$ . For the case of identical machines, an iterative algorithm with  $O(\log m)$  time complexity is derived. In each iteration of this algorithm, the maximal possible batch size is computed, and batches of this size are assigned to all machines. Kovalyov and Shafransky [53] prove that, without a common deadline, these problems with an arbitrary number of machines become unary NP-hard, even for problem  $P|s_f = 1, p_j = 1, n_f = n/F, \bar{d}_j|$ . Note that, for scheduling families of identical jobs on parallel machines, the binary (or unary) NP-hardness of a special case implies that the corresponding general problem is binary (or unary) NP-hard. Thus, NP-hardness results for problems with identical parallel machines, and with the zero cost function, imply corresponding results for unrelated parallel-machine problems, and for the other objective functions introduced in Section 2, respectively. The only open questions are the complexities of certain problems with  $s_f = s$  and  $p_j = p$ , where  $s$  is not a multiple of  $p$ .

For problem,  $R|s_f, \bar{d}_j|$ , Brucker et al. [7] present a family of approximation algorithms  $\{DP_\epsilon\}$ . For any  $\epsilon > 0$ , algorithm  $DP_\epsilon$  generates a schedule for which  $C_j \leq (1 + \epsilon)d_j$  for  $j = 1, \dots, n$ , if there exists a feasible schedule. The time requirement of algorithm  $DP_\epsilon$  is  $O(F^{2m+1}/\epsilon^{2m})$ .

#### 4.5. Batch availability

Batch availability in the case of a single machine or parallel machines means that all jobs of

the same batch are completed together when the last job in this batch has finished its processing. Similarly, for shop problems, batch availability stipulates that all jobs of a batch become available for processing on the next machine, or are completed if they have no further operations, at the time that the batch completes its processing. In both cases, jobs in a batch are processed sequentially, so that the processing time of a batch is equal to the sum of the processing times of its jobs or operations. Single-machine, parallel-machine and shop problems are discussed in separate subsections. Note that most of the complexity results apply to the case of a single family.

##### 4.5.1. Single machine

We first consider problem  $1|s_f = s, F = 1|\sum C_j$ . Since Coffman et al. [23] show that there exists an optimal schedule in which jobs are sequenced in SPT order, we reindex the jobs so that  $p_1 \leq \dots \leq p_n$ .

A backward dynamic programming algorithm with batch insertion is proposed by Coffman et al. [23] for problem  $1|s_f = s, F = 1|\sum C_j$ . Let  $G_j$  denote the minimum value of the total completion time for schedules containing jobs  $j, \dots, n$ , where the schedule starts at time zero with a setup time followed by the processing of a batch that contains job  $j$ . The initialization is

$$G_{n+1} = 0,$$

and the recursion for  $j = n, n-1, \dots, 1$  is

$$G_j = \min_{k=j+1, \dots, n+1} \left\{ G_k + (n-j+1) \left( s + \sum_{h=j}^{k-1} p_h \right) \right\}.$$

The minimization selects a batch  $\{j, \dots, k-1\}$  to insert at the start of the previous schedule containing jobs  $k, \dots, n$ . Batch  $\{j, \dots, k-1\}$  completes at time  $s + \sum_{h=j}^{k-1} p_h$ , and the processing of the batches containing jobs  $k, \dots, n$  is delayed by time  $s + \sum_{h=j}^{k-1} p_h$  as a result of the insertion. The optimal solution value is then equal to  $G_1$ . Under the most natural implementation, the algorithm requires  $O(n^2)$  time.

Coffman et al. [23] provide an implementation that solves the above recursion in  $O(n)$  time once the jobs have been reindexed. Thus, problem

$1|s_f = s, F = 1|\sum C_j$  is solvable in  $O(n \log n)$  time. In this more efficient implementation, a queue stores the jobs  $k$  that are candidates to start the second batch, where the first batch starts with job  $j$ , for  $j = n, n - 1, \dots, 1$ . By deriving various properties of the dynamic program, Coffman et al. show that the queue can be initiated and maintained in  $O(n)$  time. Alternatively, an implementation that uses the geometric techniques of van Hoesel et al. [75] provides the same time complexity.

Albers and Brucker [3] prove that problem  $1|s_f = s, F = 1|\sum w_j C_j$  is unary NP-hard. They also show that, for a given job sequence, a dynamic program of the type given above solves problem  $1|s_f = s, F = 1|\sum w_j C_j$  in  $O(n)$  time. Since Albers and Brucker show that problem  $1|s_f = s, F = 1, p_j = p|\sum w_j C_j$  has an optimal schedule in which jobs are sequenced in non-increasing order of weights, this problem with a common processing time is solvable in  $O(n \log n)$  time.

Cheng et al. [14] consider problem  $1|s_f|\sum C_j$  with an arbitrary number of families. They show that there exists an optimal schedule in which the jobs within each family are sequenced in SPT order. Using ideas from Monma and Potts [59] and Coffman et al. [23], Cheng et al. [14] propose a backward dynamic programming algorithm with batch insertion that requires  $O(n^F)$  time, which is polynomial for fixed  $F$ .

For problem  $1|s_f = s, F = 1, p_j = p|\sum C_j$  in which all jobs are identical, a solution can be obtained in  $O(n)$  time using the above dynamic programming algorithm. However, this is not a polynomial algorithm, since the input comprises only  $n, s$  and  $p$ . A key to the development of a polynomial algorithm is provided by Santos [68] and Santos and Magazine [69] who analyze a continuous relaxation in which batch sizes are not constrained to be integer. Specifically, they show that the optimal number of batches is  $B = \lfloor \sqrt{1/4 + 2np/s} - 1/2 \rfloor$ , and that optimal batch sizes are  $n/B + s(B + 1)/(2p) - ks/p$  for  $k = 1, \dots, B$ . Shallcross [70] derives an  $O(\log p \log(np))$  algorithm for the discrete problem. Although there are resemblances between the integer and continuous solutions, the algorithm is rather intricate.

Hurink [46] compares tabu search algorithms for problem  $1|s_f|\sum w_j C_j$ . He represents solutions as sequences, and uses the above  $O(n)$  dynamic programming algorithm to partition the sequence into batches. Initial results indicate that this representation is superior to one in which solutions are represented as a set of batches, that are then sequenced using a generalization of the SWPT rule. The transpose neighbourhood is compared with a restricted version of the insert or shift neighborhood. For the transpose neighbourhood, two adjacent jobs are interchanged; Hurink derives an  $O(n)$  algorithm to find a best transpose neighbour. For the restricted insert neighbourhood, a job is removed from its current position in the sequence and reinserted in a new position so that the absolute difference of the positions is at most 3, 5 or 10. Computational results for instances with up to 200 jobs show that differences in performance between the two methods are small. Moreover, the best quality solutions are obtained with a combined method that iterates between the two neighbourhoods.

We now turn our attention to the maximum lateness objective function. For problem  $1|s_f = s, F = 1|L_{\max}$ , Webster and Baker [79] show that there exists an optimal schedule in which jobs are sequenced in EDD order. We reindex the jobs so that  $d_1 \leq \dots \leq d_n$ .

A backward dynamic programming algorithm with batch insertion is proposed by Webster and Baker [79] for problem  $1|s_f = s, F = 1|L_{\max}$ . Let  $G_j$  denote the minimum value of the maximum lateness for schedules containing jobs  $j, \dots, n$ , where the schedule starts at time zero with a setup time followed by the processing of a batch that contains job  $j$ . The initialization is

$$G_{n+1} = -\infty$$

and the recursion for  $j = n, n - 1, \dots, 1$  is

$$G_j = \min_{k=j+1, \dots, n+1} \left\{ \max \left\{ G_k + s + \sum_{h=j}^{k-1} p_h, s + \sum_{h=j}^{k-1} p_h - d_j \right\} \right\}.$$

The optimal solution value is then equal to  $G_1$ . The algorithm requires  $O(n^2)$  time.

For problem  $1|s_f = s, F = 1, \bar{d}_j| \cdot$  in which it is required to find a feasible schedule with respect to given deadlines, a more efficient algorithm is proposed by Hochbaum and Landy [43]. Assume that the jobs are indexed in EDD order with respect to the deadlines. The first batch contains jobs  $1, \dots, j$ , where  $j$  is the maximum number of jobs that can be included in the first batch without exceeding the deadline  $\bar{d}_1$ . Similarly, the second batch contains a maximum number of jobs with the smallest indices (excluding those in the first batch) that can be scheduled so that they complete by time  $\bar{d}_{j+1}$ . This process is repeated until job  $n$  is included in some batch, in which case a feasible schedule is obtained, or until some job cannot be included in a batch without violating the deadline of this batch, in which case no feasible schedule exists. The algorithm requires  $O(n \log n)$  time for reindexing the jobs, and  $O(n)$  time for constructing the batches.

We now consider the (weighted) number of late jobs objective. For problem  $1|s_f = s, F = 1| \sum w_j U_j$ , Hochbaum and Landy [43] show that there exists an optimal schedule in which early jobs are sequenced in EDD order, following by any late jobs. We reindex the jobs so that  $d_1 \leq \dots \leq d_n$ .

Brucker and Kovalyov [6] propose a forward dynamic programming algorithm with job appending for problem  $1|s_f = s, F = 1| \sum U_j$ . Let  $G_j(u, h)$  denote the minimum makespan for schedules containing early jobs selected from  $1, \dots, j$ , where  $u$  is the number of late jobs, and  $h$  is the first job in the final batch of early jobs. If there are no early jobs in the schedule, then we set  $h = 0$ , and define  $d_0 = 0$ . The initialization is

$$G_0(0, 0) = 0$$

and the recursion for  $j = 1, \dots, n, u = 0, 1, \dots, j$  and  $h = 0, 1, \dots, j$  is

$$G_j(u, h) = \begin{cases} G_{j-1}(u-1, h), & \text{if } h < j, \\ G_{j-1}(u, h) + p_j, & \text{if } h < j \text{ and } G_{j-1}(u, h) \\ & \quad + p_j \leq d_h, \\ \min_{h \in H_j(u)} \{G_{j-1}(u, h')\} \\ \quad + s + p_j, & \text{if } h = j, \\ \infty, & \text{otherwise,} \end{cases}$$

where  $H_j(u) = \{h|h \in \{1, \dots, j-1\}, G_{j-1}(u, h) + s + p_j \leq d_j\}$ . The three terms in the minimization correspond to the decisions that job  $j$  is late, job  $j$  is early but does not start a batch, and job  $j$  is early and starts a batch, respectively. The minimum number of late jobs is then equal to the smallest value of  $u$  for which  $\min_{h=0,1,\dots,n} \{G_n(u, h)\} < \infty$ . The time complexity of the algorithm is  $O(n^3)$ .

Problem  $1|s_f = s, F = 1| \sum w_j U_j$  is pseudopolynomially solvable in  $O(n^2 W)$  time, where  $W = \sum_{j=1}^n w_j$ , by a straightforward generalization of the above algorithm. An alternative backward dynamic programming algorithm with job insertion is proposed by Hochbaum and Landy [43], which requires  $O(n^2 \min\{d_{\max}, P\})$  time, where  $d_{\max} = \max_{j=1,\dots,n} \{d_j\}$  and  $P = ns + \sum_{j=1}^n p_j$ . For problem  $1|s_f = s, F = 1, p_j = p| \sum w_j U_j$ , this algorithm can be implemented in  $O(n^4)$  time. Using standard rounding techniques, the dynamic programming algorithm of Brucker and Kovalyov [6] leads to a fully polynomial approximation scheme: the time complexity to generate a schedule with weighted number of late jobs that is no more than  $1 + \epsilon$  times the optimal value is  $O(n^3/\epsilon + n^3 \log \log n)$ .

Cheng and Kovalyov [17] study various problems with a maximum batch size. In particular, they consider special cases of problems

$$\begin{aligned} &1|s_f, F = 1, b| \sum w_j C_j, \\ &1|s_f = s, F = 1, b|L_{\max}, \\ &1|s_f = s, F = 1, b| \sum w_j U_j \text{ and} \\ &1|s_f = s, F = 1, b| \sum w_j T_j. \end{aligned}$$

They show that there exists an optimal schedule in which jobs with the same processing time are sequenced in EDD order for problems

$$\begin{aligned} &1|s_f = s, F = 1, b|L_{\max}, \\ &1|s_f = s, F = 1, b| \sum w_j U_j \text{ and} \\ &1|s_f = s, F = 1, b| \sum T_j, \end{aligned}$$

where the result for the second problem applies to the schedule of early jobs, and are sequenced in non-increasing order of weights for problems

$$\begin{aligned} &1|s_f = s, F = 1, b| \sum w_j C_j \text{ and} \\ &1|s_f = s, F = 1, b, d_j = d| \sum w_j T_j. \end{aligned}$$

Further, there exists an optimal schedule in which jobs with the same due date are sequenced in SPT order for problems

$$1|s_f = s, F = 1, b|L_{\max},$$

$$1|s_f = s, F = 1, b|\sum w_j U_j \text{ and}$$

$$1|s_f = s, F = 1, b|\sum T_j,$$

where the result for the second problem applies to the schedule of early jobs.

Suppose that there are either  $q$  distinct processing times or  $q$  distinct due dates. Using the ideas of Monma and Potts [59] for family scheduling problems, forward dynamic programming algorithms with batch appending are derived by Cheng and Kovalyov [17] for problems

$$1|s_f = s, F = 1, b|\sum w_j C_j,$$

$$1|s_f = s, F = 1, b|L_{\max},$$

$$1|s_f = s, F = 1, b|\sum T_j \text{ and}$$

$$1|s_f = s, F = 1, b, d_j = d|\sum w_j T_j.$$

These algorithms require  $O(b^q n^{q+1})$  time, which is polynomial for fixed  $q$ . Further, a forward dynamic programming algorithm with job appending for problem  $1|s_f = s, F = 1, b|\sum w_j U_j$  requires  $O(bq^2 n^q W)$  time if there are  $q$  distinct due dates, and  $O(bqn^{q+1}W)$  time if there are  $q$  distinct processing times, where  $W = \sum_{j=1}^n w_j$ . If  $q$  is fixed, these time requirements are polynomial for problem  $1|s_f = s, F = 1, b|\sum U_j$ , and pseudopolynomial for problem  $1|s_f = s, F = 1, b|\sum w_j U_j$ .

#### 4.5.2. Parallel machines

There are two relevant studies for parallel-machine scheduling under batch availability. Cheng et al. [13] derived a backward dynamic programming algorithm with batch insertion for problem  $P|s_f = s, F = 1|\sum C_j$ , which is based on the ideas of Coffman et al. [23] that are described in the previous subsection. It uses the properties that there exists an optimal schedule in which the jobs on each machine are sequenced in SPT order, and that each batch contains adjacent jobs (with respect to the SPT ordering). The time requirement of the algorithm is  $O(mn^{m+1})$  time, which is polynomial for fixed  $m$ . Cheng et al. also show that

when there is a common processing time, problem  $P|s_f = s, F = 1, p_j = p|\sum C_j$  reduces to a single-machine case, and therefore is solvable in polynomial time using the algorithm of Shallcross [70].

Problem  $R|s_f = s, F = 1, \bar{d}_j|$  is studied by Cheng and Kovalyov [18]. They derive a dynamic programming algorithm and a family of approximation algorithms  $\{A_\epsilon\}$ . For any  $\epsilon > 0$ , algorithm  $A_\epsilon$  constructs a schedule in which the completion time of each job is at most  $(1 + \epsilon)$  times the value of its deadline if there exists a feasible schedule with respect to the deadlines. The time complexity of  $A_\epsilon$  is  $O(n^{2m+1}/\epsilon^m)$ . Cheng and Kovalyov also study special cases of which most are NP-hard, since the corresponding classical parallel-machine problems with  $s = 0$  are NP-hard. Moreover, problem  $Q|s_f = s, F = 1, p_j = p, \bar{d}_j|$  is unary NP-complete, unlike its classical counterpart with  $s = 0$  which is polynomially solvable. A dynamic programming algorithm with a time complexity of  $O(m^2 n^{2m+1})$  is presented for this problem. The special case of problem  $P|s_f = s, F = 1, p_j = p, \bar{d}_j|$  for which  $s = p$  is solvable in  $O(n \log n)$  time.

#### 4.5.3. Shop problems

Glass, Potts and Strusevich [37] provide complexity results and approximation algorithms for problems

$$F2|s_{if}, F = 1|C_{\max} \text{ and } O2|s_{if}, F = 1|C_{\max}.$$

Specifically, problem  $F2|s_{if}, F = 1|C_{\max}$  is unary NP-hard, and has an optimal solution with the same batches on each machine (the batches are consistent). They also develop an approximation algorithm for problem  $F2|s_{if}, F = 1|C_{\max}$  that generates a schedule with makespan that is no more than  $4/3$  times the optimal value, and this bound is tight. Problem  $O2|s_{if}, F = 1|C_{\max}$  is shown to be binary NP-hard, and has an optimal solution with one, two or three consistent batches on each machine. By demonstrating that problem  $O2|s_{if}, F = 1|C_{\max}$  is closely related to one of minimizing makespan on two and on three identical parallel machines, they prove that problem  $O2|s_{if}, F = 1|C_{\max}$  is pseudopolynomially solvable. Moreover, approximation algorithms for scheduling two and three identical parallel machines to

minimize the makespan provide corresponding performance guarantees for problem  $O2|s_{if}, F=1|C_{\max}$ . Cheng and Wang [21] consider a special case of problem  $F2|s_{if}, F=1|C_{\max}$  in which the setup time on the first machine is zero, so that there is a batch for each job's first operation. The problem is shown to be binary hard for both anticipatory and non-anticipatory setups, and some polynomially solvable special cases are presented.

Cheng et al. [20] study problem  $F2|s_{if}, F=1|C_{\max}$  in which the setups are non-anticipatory, but with a constraint that batches are consistent. The problem is shown to be NP-hard even if, for each job, the processing time on the first machine does not exceed the processing time on the second machine, or vice versa. It is solvable in  $O(n^2)$  time when all jobs have a common processing time on the first machine or all jobs have a common processing time on the second machine by algorithms of Cheng et al. [20], and in  $O(\sqrt{n})$  time when all operations have a common processing time by an algorithm of Tanaev et al. [73]. However, the latter time complexity is not polynomial, and therefore the problem with a common processing time for all operations is open. Tanaev et al. [73] adapt all of these results to the case that setups are anticipatory.

The heuristics of Sotskov et al. [72], that we describe in Section 4.3 for the permutation flow shop problem  $F|s_{if}|\sum C_j$ , are also applied, with some minor variations, to the corresponding problems with batch availability and mixed availability (some machines operate under job availability and others operate under batch availability). Results are similar to those for job availability, although tabu search is slightly preferred to simulated annealing in the case of batch availability.

#### 4.6. Batch delivery scheduling

In this subsection, we assume that there is a single family of jobs, and a setup time is required before each batch is processed. All jobs of a batch are delivered when the processing of the last job in the batch is complete. For any schedule in which

job  $j$  belongs to some batch  $k$ , we define  $H_j = D_k - C_j$ , where  $D_k$  is the completion time of batch  $k$ , to be the *holding time* during which job  $j$  is waiting to be delivered.

Cheng et al. [15] show that single- and parallel-machine problems with criteria dependent on job holding times  $H_j$ , for  $j = 1, \dots, n$ , are equivalent to classical parallel-machine problems. Therefore, existing algorithms for parallel-machine scheduling can be suitably adapted. Cheng et al. [19] prove that the single-machine problem of minimizing  $\sum_{j=1}^n w_j H_j + (1/B) \sum_{k=1}^B D_k$ , where  $B$  is the (unknown) number of batches, is unary NP-hard, but is solvable in  $O(n^2)$  time when all jobs have a common weight or a common processing time.

Yang [82] studies single-machine problems in which  $D_1 \leq \dots \leq D_B$  are given batch delivery dates, where  $D_B = \sum_{j=1}^n p_j$ . The jobs are to be assigned to the  $B$  batches so that batch  $k$  is completed no later than time  $D_k$ , for  $k = 1, \dots, B$ , and the objective is either to minimize either the total weighted holding time  $\sum_{j=1}^n w_j H_j$ , or the maximum weighted holding time  $\max_{j=1, \dots, n} \{w_j H_j\}$ . He proves that these problems are unary NP-hard, and are binary NP-hard when  $B = 2$  and all weights are equal. For the case of a common processing time, the problems are shown to be solvable in  $O(n \log n)$  time.

#### 4.7. Multi-operation jobs

This subsection considers the single-machine, multi-operation job model in which each job  $j$  comprises one operation in family  $f$ , for  $f = 1, \dots, F$ , which has processing time  $p_{jf}$  (although some of these operations may be missing). The completion time of a job is the time when the processing of its last operation is complete. As observed by Julien and Magazine [49], this model arises when a job contains several orders that must be delivered to the customer at the same time. Other applications that are described by Gerodimos et al. [32] occur when different components are manufactured for subsequent assembly into a final product, or ingredients are produced for subsequent blending or mixing into a final product. In this context, we assume that the time

required for final assembly or the blending stage is negligible, so that the problem reduces to one of scheduling a single machine. We modify our problem classification scheme that is described in Section 2, by allowing the entry multi-op in  $\psi_2$  to indicate the presence of multi-operation jobs.

Gerodimos et al. [34] provide complexity results for several single-machine multi-operation job problems. They point out an equivalence between problem  $1|s_f, \text{multi-op}|L_{\max}$  and the single-operation problem  $1|s_f|L_{\max}$  that is formed by treating operations as jobs and assigning to each operation the due date of the corresponding job. Using the results in Section 4.1.2, this equivalence establishes that problem  $1|s_f, \text{multi-op}|L_{\max}$  is binary NP-hard for arbitrary  $F$  from the result of Bruno and Downey [8], but is solvable in  $O(n^F)$  time by the dynamic programming algorithm of Ghosh and Gupta [36]. Gerodimos et al. also show that problem  $1|s_f = s, \text{multi-op}|\sum U_j$  is binary NP-hard, even if there are only two families, and problem  $1|s_f = 1, \text{multi-op}|\sum U_j$  is unary NP-hard, even if all operations (except those that are missing) have a processing time of one unit. However, for fixed  $F$ , a backward dynamic programming algorithm solves problem  $1|s_f, \text{multi-op}|\sum w_j U_j$  in  $O(nd_{\max}^F)$  time, where  $d_{\max}$  is the largest due date. The key observation is that when the final operation of a job is scheduled, the other operations are placed in a sub-batch that waits to be scheduled, and it is sufficient to store the processing time of such sub-batches as state variables. For a special case of problem  $1|s_f, \text{multi-op}|\sum C_j$  in which the processing times of operations between families are agreeable (there is a job indexing for which  $p_{1,f} \leq \dots \leq p_{n,f}$  for  $f = 1, \dots, F$ ), a similar dynamic programming algorithm provides a solution in  $O(n^F)$  time. However, the complexity of the general case of problem  $1|s_f, \text{multi-op}|\sum C_j$  is open. Coffman et al. [22] show that the two-operation problem  $1|s_f = s, \text{multi-op}|\sum C_j$  is solvable in  $O(\sqrt{n})$  time when all operations in the first family have a common processing time and all operations in the second family have a common processing time, although this time complexity is not polynomial. Gerodimos [31] shows that the above NP-hardness results also hold under the assumption of batch

availability. Moreover, the dynamic programming algorithms for problems  $1|s_f, \text{multi-op}|L_{\max}$  and  $1|s_f, \text{multi-op}|\sum w_j U_j$  can be adapted to the case of batch availability with a slight increase in time complexity.

Several complexity results are available for a special case of the two-operation job model that is introduced by Baker [4]. For this special case, each job has a standard and a specific operation. Each standard operation is in the same family, whereas each specific operation has its own family. Gerodimos et al. [33] derive several structural properties that allow dynamic programming algorithms to be developed. Specifically, for a special case of problem  $1|s_f, \text{multi-op}|\sum C_j$  in which the processing times of the two operations are agreeable, they propose a backward dynamic programming algorithm that requires  $O(n^2)$  time, which improves on an  $O(n^3)$  algorithm of Vickson et al. [76]. A similar algorithm solves problem  $1|s_f, \text{multi-op}|L_{\max}$  in  $O(n^2)$  time. Gerodimos et al. [33] also show that problem  $1|s_f, \text{multi-op}|\sum U_j$  is binary NP-hard, but is solvable by a forward dynamic programming algorithm in  $O(n^2 d_{\max})$  time. Further, the special case of problem  $1|s_f, \text{multi-op}|\sum U_j$  in which each standard operation has a common processing time is polynomially solvable, and problem  $1|s_f, \text{multi-op}|\sum w_j U_j$  is pseudo-polynomially solvable. Under batch availability, Gerodimos et al. [32] show that there exists an optimal schedule in which the standard operation of any job precedes its specific operation (which does not hold under job availability). For problem  $1|s_f, \text{multi-op}|\sum C_j$  under batch availability, with the above assumption of agreeability, an  $O(n \log n)$  algorithm of Coffman et al. [23] improves upon a previous  $O(n^2)$  algorithm of Baker. Results of Gerodimos et al. [32] show that the complexity results listed above for problems  $1|s_f, \text{multi-op}|L_{\max}$  and  $1|s_f, \text{multi-op}|\sum (w_j) U_j$  under job availability, also apply for batch availability, although the algorithms and proofs are different.

## 5. Batching machine models

Recall that a batching machine processes jobs of the same batch in parallel, so that the processing

time of a batch is equal to the maximum of the processing times of its jobs or operations. All jobs or operations of the same batch begin processing at the same time, and they all have a common completion time, which is the time that the longest job or operation in this batch completes its processing. In this section, we assume that there are no setup times. The three subsections consider a single batching machine, parallel batching machines, and shop problems with batching machines, respectively.

### 5.1. Single batching machine

Brucker et al. [5] study the complexity of scheduling a single batching machine both for unrestricted batch sizes, and for batches that can contain at most  $b$  jobs. If  $\mathcal{B}$  is a set of jobs that form a batch, then its processing time is  $\max_{j \in \mathcal{B}} \{p_j\}$ , and the completion time of every job in  $\mathcal{B}$  is the time at which the processing of batch  $\mathcal{B}$  finishes.

Assume that jobs are indexed according to the SPT rule, so that  $p_1 \leq \dots \leq p_n$ . An *SPT-batch schedule*, is one in which adjacent jobs in the sequence  $(1, \dots, n)$  may be grouped to form batches. For example, a possible batch schedule for a 10-job problem is the sequence of four batches specified by  $(\{1, 2, 3\}, \{4\}, \{5, 6, 7, 8\}, \{9, 10\})$ . For problems  $\tilde{1} \parallel \psi_3$ , where there is no restriction on the batch size and  $\psi_3$  is any objective functions introduced in Section 2, Brucker et al. [5] use a straightforward argument to show that there exists an optimal solution that is an SPT-batch schedule.

Brucker et al. [5] propose dynamic programming algorithms for several problems where no restriction on the batch size is imposed. These algorithms, together with other relevant results, are reviewed below.

#### 5.1.1. Total weighted completion time

For problem  $\tilde{1} \parallel \sum w_j C_j$ , Brucker et al. [5] propose a backward dynamic programming algorithms with batch insertion. Let  $G_j$  be the minimum total weighted completion time for SPT-batch schedules containing jobs  $j, \dots, n$  where processing of the first batch in the schedule starts at time zero. The initialization is

$$G_{n+1} = 0,$$

and the recursion for  $j = n, n - 1, \dots, 1$  is

$$G_j = \min_{k=j+1, \dots, n+1} \left\{ G_k + p_{k-1} \sum_{h=j}^n w_h \right\}.$$

The minimization selects a batch  $\{j, \dots, k - 1\}$ , which has processing time  $p_{k-1}$ , to be inserted at the start of a previous schedule comprising jobs  $k, \dots, n$ . The optimal solution value is then equal to  $G_1$ . Under the most natural implementation, the algorithm requires  $O(n^2)$  time. However, since this dynamic program has a structure which allows the geometric techniques of van Hoesel et al. [75] to be applied, the time complexity can be reduced to  $O(n \log n)$ .

In the case of a restricted batch size  $b$ , Brucker et al. [5] develop an  $O(n^{b(b-1)})$  dynamic programming algorithm for problem  $\tilde{1} | b | \sum C_j$ . However, for arbitrary  $b$  the complexity of  $\tilde{1} | b | \sum C_j$  is open, as is the complexity of  $\tilde{1} | b | \sum w_j C_j$  for fixed and arbitrary  $b$ . For problem  $\tilde{1} | b | \sum C_j$  when there are  $q$  distinct processing times, dynamic programming algorithms are developed by Chandru et al. [9], Hochbaum and Landy [44] and Brucker et al. [5]. The most efficient is the latter, which has a time complexity of  $O(b^2 q^2 2^q)$ .

#### 5.1.2. Maximum lateness

For problem  $\tilde{1} \parallel L_{\max}$ , Brucker et al. [5] propose a backward dynamic programming algorithm with batch insertion that is similar to the one in the previous subsection and the algorithm for the batch availability problem  $\tilde{1} | s_f, F = 1 | L_{\max}$  that is described in Section 4.5.1. Let  $G_j$  be the minimum value of the maximum lateness for SPT-batch schedules containing jobs  $j, \dots, n$ , where processing starts at time zero. The initialization is

$$G_{n+1} = -\infty$$

and the recursion for  $j = n, n - 1, \dots, 1$  is

$$G_j = \min_{k=j+1, \dots, n+1} \left\{ \max \left\{ G_k + p_{k-1}, \max_{h=j, \dots, k-1} \{p_{k-1} - d_h\} \right\} \right\}.$$

The optimal solution value is then equal to  $G_1$ . The algorithm requires  $O(n^2)$  time.

In the case of a restricted batch size  $b$ , Brucker et al. [5] show that problem  $\tilde{I}|b|L_{\max}$  is unary NP-hard. Lee et al. [54] analyze the complexity of several special cases of problem  $\tilde{I}|b, r_j|L_{\max}$  that arise by making assumptions about the release dates, processing times and due dates.

5.1.3. *Weighted number of late jobs*

For problem  $\tilde{I}||\sum U_j$ , Brucker et al. [5] propose a forward dynamic programming algorithm with job appending. Let  $G_j(u, k)$  be the minimum makespan for SPT-batch schedules containing jobs  $1, \dots, j$ , where  $u$  is the number of late jobs in the schedule, and the final batch is assigned a processing time  $p_k$  because it is to contain jobs  $j + 1, \dots, k$ , but not job  $k + 1$ , where  $j \leq k$ . The initialization for  $k = 0, 1, \dots, n$  is

$$G_0(0, k) = \begin{cases} 0, & \text{if } k = 0, \\ \infty, & \text{otherwise,} \end{cases}$$

and the recursion for  $j = 1, \dots, n$ ,  $u = 0, 1, \dots, j$ , and  $k = j, \dots, n$  is

$$G_j(u, k) = \min \begin{cases} G_{j-1}(u, k), & \text{if } G_{j-1}(u, k) \leq d_j, \\ G_{j-1}(u - 1, k), & \text{if } G_{j-1}(u - 1, k) > d_j, \\ G_{j-1}(u, j - 1) + p_k, & \text{if } G_{j-1}(u, j - 1) + p_k \leq d_j, \\ G_{j-1}(u - 1, j - 1) + p_k, & \text{if } G_{j-1}(u - 1, j - 1) + p_k > d_j, \\ \infty, & \text{otherwise.} \end{cases}$$

The minimum number of late jobs is then equal to the smallest value  $u$  for which  $G_n(u, n) < \infty$ . The time complexity of the algorithm is  $O(n^3)$ .

For problem  $\tilde{I}||\sum w_j U_j$ , the above dynamic programming algorithm generalizes to minimize the weighted number of late jobs in  $O(n^2 W)$  time, where  $W = \sum_{j=1}^n w_j$ . An alternative pseudopolynomial dynamic programming algorithm of the type given in the following subsection gives a solution in  $O(n^2 P)$  time, where  $P = \sum_{j=1}^n p_j$ . Brucker et al. [5] show that this problem is binary NP-hard.

The unary NP-hardness of problem  $\tilde{I}|b|\sum U_j$  follows from the corresponding result for problem  $\tilde{I}|b|L_{\max}$ . Lee et al. [54] also consider special cases

of problem  $\tilde{I}|b, r_j|\sum U_j$  under a variety of different assumptions about the release dates, processing times and due dates.

5.1.4. *Total weighted tardiness*

For problem  $\tilde{I}||\sum w_j T_j$ , Brucker et al. [5] propose a forward dynamic programming algorithm with batch appending. Let  $G_j(t)$  be the minimum total weighted tardiness for SPT-batch schedules containing jobs  $1, \dots, j$ , where the last batch completes at time  $t$ . The initialization is

$$G_0(t) = \begin{cases} 0, & \text{if } t = 0, \\ \infty, & \text{otherwise,} \end{cases}$$

and the recursion for  $j = 1, \dots, n$  and  $t = p_j, \dots, \sum_{k=1}^j p_k$  is

$$G_j(t) = \min_{h=0,1,\dots,j-1} \left\{ G_h(t - p_j) + \sum_{k=h+1}^j w_k \max\{t - d_k, 0\} \right\}.$$

The optimal solution value is then equal to  $\min_{t=p_n, \dots, P} \{G_n(t)\}$ , where  $P = \sum_{j=1}^n p_j$ . The algorithm requires  $O(n^2 P)$  time.

Brucker et al. [5] show that problem  $\tilde{I}||\sum w_j T_j$  is binary NP-hard. Also, problem  $\tilde{I}|b|\sum T_j$  is unary NP-hard, due to the corresponding result for problem  $\tilde{I}|b|L_{\max}$ .

5.2. *Parallel batching machines*

For the case of  $m$  identical parallel batching machines and an arbitrary regular objective function, where there is no restriction on the batch size for any machine, Brucker et al. [5] observe that there exists an optimal solution comprising an SPT-batch schedule on each machine. For a fixed number of identical parallel machines, therefore, the dynamic programming algorithms for single-machine problems generalize to give pseudopolynomial algorithms.

5.3. *Shop problems with batching machines*

Potts et al. [62] study the complexity of minimizing the makespan in open shops, job shops and

flow shops with two batching machines. For the open shop, they show that problem  $\tilde{O}2||C_{\max}$  is solvable in  $O(n)$  by forming two batches according to whether or not the processing time of a job is less on the first machine than on the second, and then scheduling these batches on the two machines. On the other hand, problem  $\tilde{O}2|b_1|C_{\max}$  is binary NP-hard for fixed  $b_1$ , where  $b_1 \geq 1$ . However, by showing that problem  $\tilde{O}2|b_1|C_{\max}$  has an optimal solution with a maximum of three batches on the second machine, Potts et al. [62] develop a pseudopolynomial dynamic programming algorithm. Moreover, they derive an  $O(n^{b_2(b_2-1)})$  algorithm for problem  $\tilde{O}2|b_1=1, b_2|C_{\max}$ , which can be implemented in  $O(n \log n)$  time when  $b_2 = 2$ . For the flow shop and job shop, they show that problems  $\tilde{F}2||C_{\max}$ , and  $\tilde{J}2||C_{\max}$  in which there are at most two operations per job, are solvable in  $O(n \log n)$  time by forming schedules with at most two batches and at most three batches on each machine, respectively. Binary NP-hardness is established for problems  $\tilde{F}2|b_1, b_2|C_{\max}$  and  $\tilde{J}2|b_1, b_2|C_{\max}$  for fixed  $b_1$  and  $b_2$ , where  $\max\{b_1, b_2\} \geq 2$ , and for fixed  $b_1$ , where  $b_1 \geq 2$  and  $b_2 \geq n$ . Problem  $\tilde{F}2|b_1=1|C_{\max}$  can be formulated as a single-machine problem  $\tilde{I}||L_{\max}$  for which the  $O(n^2)$  algorithm of Brucker et al. [5], as described in Section 5.1.2, can be applied. The corresponding job shop problem  $\tilde{J}2|b_1=1|C_{\max}$  with at most two operations per job is also polynomially solvable with a more complicated dynamic programming algorithm. Note that, by symmetry, the complexity results for shop problems with the makespan objective also hold when  $b_1$  and  $b_2$  are interchanged.

Ahmadi et al. [1] consider the problem of scheduling a two-stage flow shop in which one or both of the stages may comprise a batching machine. In their model, the processing time of a batch on a batching machine is fixed (and is therefore independent of the jobs that it contains). They provide a full complexity classification for the  $C_{\max}$  and  $\sum C_j$  objectives. For the case of a batching machine at the first stage and the  $\sum C_j$  objective, Hoogeveen and van de Velde [45] propose a lower bounding scheme that is based on a Lagrangean relaxation of the constraints that relate the completion times of the operations at the

two stages. They also design an approximation algorithm that generates a schedule with total completion time that is no more than  $5/3$  times the optimal value.

Dannenbergh et al. [28] consider the permutation flow shop problems  $\tilde{F}||s_{if}, b|C_{\max}$  and  $\tilde{F}||s_{if}, b|\sum w_j C_j$  in which jobs up to  $b$  of the same family can be processed in a batch, and the search is restricted to schedules with the same batches and with the same processing order on each machine. Following a similar approach to Sotskov et al. [72], they evaluate insertion heuristics and classical local search methods (multi-start descent, simulated annealing and tabu search) and a ‘multi-level’ simulated annealing approach in which a neighbour is obtained by performing a move in a relatively large neighbourhood and then descent is applied in a smaller neighbourhood. Computational tests for problems with 40, 60, 80, 100 and 120 jobs, and 5 and 10 machines, show that insertion heuristics are effective, and that a multi-level approach is preferred to the other local search methods.

## 6. Concluding remarks

This paper reviews research on two types of scheduling models that require batches to be formed. In the family scheduling model, similar jobs may be batched if they share the same setup. On the other hand, the batching machine model forms batches of jobs that are processed at the same time. Analysis of these models shows that, in some cases, the sequencing and batching of jobs can be decoupled. Once a sequence of jobs (within a family, in the family scheduling model) is known, dynamic programming is shown to be a useful technique for solving the batching problem.

Many of the problems discussed in this paper are known to be polynomially solvable or NP-hard. Among the few open problems, many have the total (weighted) completion time objective function. For the NP-hard problems, the number of studies on the design of branch and bound and approximation algorithms is limited. In view of the interest in scheduling with batching, research effort in designing algorithms for NP-hard problems is worthwhile.

There are other models involving both batching and scheduling that do not fall within the scope of this review. For instance, suppose that jobs are delivered to customers in batches. In this model, forming large batches saves on delivery costs, but increases job completion times (under the natural assumption of batch availability). Another example where batching decisions are necessary is pointed out by Fazle Baki and Vickson [30]. They consider multi-machine problems with a single operator, where each operation requires the operator to be present. When the operator moves between machines, a time delay, which can be regarded as a setup, is incurred.

### Acknowledgements

This research was partially supported by INT-AS (Project INTAS-93-257, INTAS-93-257-Ext, and INTAS-96-0820 for the second author).

### References

- [1] J.H. Ahmadi, R.H. Ahmadi, S. Dasu, C.S. Tang, Batching and scheduling jobs on batch and discrete processors, *Operations Research* 39 (1992) 750–763.
- [2] B.-H. Ahn, J.H. Hyun, Single facility multi-class job scheduling, *Computers and Operations Research* 17 (1990) 265–272.
- [3] S. Albers, P. Brucker, The complexity of one-machine batching problems, *Discrete Applied Mathematics* 47 (1993) 87–107.
- [4] K.R. Baker, Scheduling the production of components at a common facility, *IIE Transactions* 20 (1988) 32–35.
- [5] P. Brucker, A. Gladky, J.A. Hoogeveen, M.Y. Kovalyov, C.N. Potts, T. Tautenhahn, S.L. van de Velde, Scheduling a batching machine, *Journal of Scheduling* 1 (1998) 31–54.
- [6] P. Brucker, M.Y. Kovalyov, Single machine batch scheduling to minimize the weighted number of late jobs, *Mathematical Methods of Operations Research* 43 (1996) 1–8.
- [7] P. Brucker, M.Y. Kovalyov, Y.M. Shafransky, F. Werner, Batch scheduling with deadlines on parallel machines, *Annals of Operations Research* 83 (1998) 23–40.
- [8] J. Bruno, P. Downey, Complexity of task sequencing with deadlines, set-up times and changeover costs, *SIAM Journal on Computing* 7 (1978) 393–404.
- [9] V. Chandru, C.-Y. Lee, R. Uzsoy, Minimizing total completion time on a batch processing machine with job families, *Operations Research Letters* 13 (1993) 61–65.
- [10] B. Chen, A better heuristic for preemptive parallel machine scheduling with batch setup times, *SIAM Journal on Computing* 22 (1993) 1303–1318.
- [11] B. Chen, C.N. Potts, V.A. Strusevich, Approximation algorithms for two-machine flow shop scheduling with batch setup times, *Mathematical Programming* 82 (1998) 255–271.
- [12] T.C.E. Cheng, Z.-L. Chen, Parallel machine scheduling with batch setup times, *Operations Research* 42 (1994) 1171–1174.
- [13] T.C.E. Cheng, Z.-L. Chen, M.Y. Kovalyov, B.M.T. Lin, Parallel-machine batching and scheduling to minimize total completion time, *IIE Transactions* 28 (1996) 953–956.
- [14] T.C.E. Cheng, Z.-L. Chen, C. Oguz, One-machine batching and sequencing of multiple-type items, *Computers and Operations Research* 21 (1994) 717–721.
- [15] T.C.E. Cheng, V.S. Gordon, M.Y. Kovalyov, Single machine scheduling with batch deliveries, *European Journal of Operational Research* 94 (1996) 277–283.
- [16] T.C.E. Cheng, M.Y. Kovalyov, Batch scheduling and common due date assignment on a single machine, *Discrete Applied Mathematics* 70 (1996) 231–245.
- [17] T.C.E. Cheng, M.Y. Kovalyov, Single machine batch scheduling with sequential job processing, in submission.
- [18] T.C.E. Cheng, M.Y. Kovalyov, Algorithms for parallel machine batch scheduling with deadlines, in submission.
- [19] T.C.E. Cheng, M.Y. Kovalyov, B.M.T. Lin, Single machine scheduling to minimize batch delivery and job earliness penalties, *SIAM Journal on Optimization* 7 (1997) 547–559.
- [20] T.C.E. Cheng, A. Toker, B.M.T. Lin, Makespan minimization in the two-machine flow-shop batch scheduling problem, Working paper 04/95-6, The Hong Kong Polytechnic University, Faculty of Business and Information Systems, 1995.
- [21] T.C.E. Cheng, G. Wang, Batching and scheduling to minimize the makespan in the two-machine flowshop, *IIE Transactions* 30 (1998) 447–453.
- [22] E.G. Coffman Jr., A. Nozari, M. Yannakakis, Optimal scheduling of products with two subassemblies on a single machine, *Operations Research* 37 (1989) 426–436.
- [23] E.G. Coffman Jr., M. Yannakakis, M.J. Magazine, C.A. Santos, Batch sizing and job sequencing on a single machine, *Annals of Operations Research* 26 (1990) 135–147.
- [24] H.A.J. Crauwels, A.M.A. Hariri, C.N. Potts, L.N. Van Wassenhove, Branch and bound algorithms for single-machine scheduling with batch set-up times to minimize total weighted completion time, *Annals of Operations Research* 83 (1998) 59–76.
- [25] H.A.J. Crauwels, C.N. Potts, D. Van Oudheusden, L.N. Van Wassenhove, Branch and bound algorithms for single machine scheduling with batching to minimize the number of late jobs, Report, Faculty of Mathematical Studies, University of Southampton, UK, 1999.
- [26] H.A.J. Crauwels, C.N. Potts, L.N. Van Wassenhove, Local search heuristics for single-machine scheduling with batching to minimize the number of late jobs, *European Journal of Operational Research* 90 (1996) 200–213.

- [27] H.A.J. Crauwels, C.N. Potts, L.N. Van Wassenhove, Local search heuristics for single machine scheduling with batch set-up times to minimize total weighted completion time, *Annals of Operations Research* 70 (1997) 261–279.
- [28] D. Dannenberg, T. Tautenhahn, F. Werner, A comparison of heuristic algorithms for flow shop scheduling problems with setup times and limited batch size, Preprint No. 52, Fakultät für Mathematik, Otto-von-Guericke-Universität Magdeburg, Germany, 1997.
- [29] G. Dobson, U.S. Karmarkar, J.L. Rummel, Batching to minimize flow times on one machine, *Management Science* 33 (1987) 784–799.
- [30] M. Fazle Baki, R.G. Vickson, One-operator, two-machine scheduling with setup times for machines and maximum lateness objective, Technical paper 205-MS, Department of Management Sciences, University of Waterloo, Canada, 1997.
- [31] A.E. Gerodimos, Private communication, 1998.
- [32] A.E. Gerodimos, C.A. Glass, C.N. Potts, Scheduling the production of two-component jobs on a single machine, *European Journal of Operational Research*, in press.
- [33] A.E. Gerodimos, C.A. Glass, C.N. Potts, Scheduling customised jobs on a single machine under item availability, Report OR88, Faculty of Mathematical Studies, University of Southampton, UK, 1997.
- [34] A.E. Gerodimos, C.A. Glass, C.N. Potts, T. Tautenhahn, Scheduling multi-operation jobs on a single machine, *Annals of Operations Research*, in press.
- [35] J.B. Ghosh, Batch scheduling to minimize total completion time, *Operations Research Letters* 16 (1994) 271–275.
- [36] J.B. Ghosh, J.N.D. Gupta, Batch scheduling to minimize maximum lateness, *Operations Research Letters* 21 (1997) 77–80.
- [37] C.A. Glass, C.N. Potts, V.A. Strusevich, Scheduling batches with sequential job processing for two-machine flow and open shops, Report, Faculty of Mathematical Studies, University of Southampton, UK, 1998.
- [38] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic machine scheduling: A survey, *Annals of Discrete Mathematics* 5 (1979) 287–326.
- [39] J.N.D. Gupta, Single facility scheduling with multiple job classes, *European Journal of Operational Research* 33 (1988) 42–45.
- [40] A.M.A. Hariri, C.N. Potts, Single machine scheduling with batch set-up times to minimize maximum lateness, *Annals of Operations Research* 70 (1997) 75–92.
- [41] R. Hassin, Private communication, 1996.
- [42] J.W. Herrmann, C.-Y. Lee, Solving a class scheduling problem with a genetic algorithm, *ORSA Journal on Computing* 7 (1995) 443–452.
- [43] D.S. Hochbaum, D. Landy, Scheduling with batching: Minimizing the weighted number of tardy jobs, *Operations Research Letters* 16 (1994) 79–86.
- [44] D.S. Hochbaum, D. Landy, Scheduling semiconductor burn-in operations to minimize total flowtime, *Operations Research* 45 (1997) 874–885.
- [45] J.A. Hoogeveen, S.L. van de Velde, Scheduling by positional completion times: Analysis of a two-stage flow shop with a batching machine, *Mathematical Programming* 82 (1998) 273–289.
- [46] J. Hurink, A tabu search approach for a single-machine batching problem using an efficient method to calculate a best neighbor, *Journal of Scheduling* 1 (1998) 127–148.
- [47] J.R. Jackson, Scheduling a production line to minimize maximum tardiness, Research report 43, Management Science Research Project, University of California, Los Angeles, CA, 1955.
- [48] S.M. Johnson, Optimal two- and three-stage production schedules with setup times included, *Naval Research Logistics Quarterly* 1 (1954) 61–68.
- [49] F.M. Julien, M.J. Magazine, Scheduling customer orders: An alternative production scheduling approach, *Journal of Manufacturing and Operations Management* 3 (1990) 177–199.
- [50] U. Kleinau, Two-machine shop scheduling problems with batch processing, *Mathematical and Computer Modelling* 17 (1993) 55–66.
- [51] M.Y. Kovalyov, Batch scheduling and common due date assignment problem: An NP-hard case, *Discrete Applied Mathematics* 80 (1997) 251–254.
- [52] M.Y. Kovalyov, C.N. Potts, L.N. Van Wassenhove, Single machine scheduling with set-ups to minimize the number of late items, Report, Econometric Institute, Erasmus University Rotterdam, Rotterdam, The Netherlands, 1992.
- [53] M.Y. Kovalyov, Y.M. Shafransky, Batch scheduling with deadlines on parallel machines: An NP-hard case, *Information Processing Letters* 64 (1997) 69–74.
- [54] C.-Y. Lee, R. Uzsoy, L.A. Martin-Vega, Efficient algorithms for scheduling semiconductor burn-in operations, *Operations Research* 40 (1992) 764–775.
- [55] C.-J. Liao, L.-M. Liao, Single machine scheduling with major and minor setup times, *Computers and Operations Research* 24 (1997) 169–178.
- [56] A.J. Mason, Genetic Algorithms and Scheduling Problems, Ph.D. Thesis, Department of Engineering, University of Cambridge, UK, 1992.
- [57] A.J. Mason, E.J. Anderson, Minimizing flow time on a single machine with job classes and setup times, *Naval Research Logistics* 38 (1991) 333–350.
- [58] R. McNaughton, Scheduling with deadlines and loss functions, *Management Science* 6 (1959) 1–12.
- [59] C.L. Monma, C.N. Potts, On the complexity of scheduling with batch setup times, *Operations Research* 37 (1989) 798–804.
- [60] C.L. Monma, C.N. Potts, Analysis of heuristics for preemptive parallel machine scheduling with batch setup times, *Operations Research* 41 (1993) 981–993.
- [61] J.M. Moore, An  $n$  job, one machine sequencing algorithm for minimizing the number of late jobs, *Management Science* 15 (1968) 102–109.
- [62] C.N. Potts, V.A. Strusevich, T. Tautenhahn, Scheduling batches with simultaneous job processing for two-machine

- shop problems, Report, Faculty of Mathematical Studies, University of Southampton, UK, 1998.
- [63] C.N. Potts, L.N. Van Wassenhove, Integrating scheduling with batching and lot-sizing: A review of algorithms and complexity, *Journal of the Operational Research Society* 43 (1992) 395–406.
- [64] G. Rote, G.J. Woeginger, Minimizing the number of tardy jobs on a single machine with batch setup times, Report Woe-23, START Project Y43-MAT, Institut für Mathematik, TU Graz, Austria, 1998.
- [65] J.M.J. Schutten, S.L. van de Velde, W.H.M. Zijm, Single-machine scheduling with release dates, due dates and family setup times, *Management Science* 42 (1996) 1165–1174.
- [66] W.E. Smith, Various optimizers for single-stage production, *Naval Research Logistics Quarterly* 3 (1956) 59–66.
- [67] A.H.G. Rinnooy Kan, *Machine Scheduling Problems*, Martinus Nijhoff, The Hague, 1976.
- [68] C.A. Santos, *Batching and Sequencing Decisions under Lead Time Considerations for Single Machine Problems*, M.Sc. Thesis, Department of Management Sciences, University of Waterloo, Canada, 1984.
- [69] C.A. Santos, M. Magazine, Batching in single operation manufacturing systems, *Operations Research Letters* 4 (1985) 99–103.
- [70] D. Shallcross, A polynomial algorithm for a one machine batching problem, *Operations Research Letters* 11 (1992) 213–218.
- [71] J. Skorin-Kapov, A.J. Vakharia, Scheduling a flow-line manufacturing cell: A tabu search approach, *International Journal of Production Research* 31 (1993) 1721–1734.
- [72] Y.N. Sotskov, T. Tautenhahn, F. Werner, Heuristics for permutation flow shop scheduling with batch setup times, *OR Spektrum* 18 (1996) 67–80.
- [73] V.S. Tanaev, M.Y. Kovalyov, Y.M. Shafransky, *Scheduling Theory. Group Technologies* (in Russian), Institute of Engineering Cybernetics, National Academy of Sciences of Belarus, Minsk, 1998.
- [74] A.J. Vakharia, Y.-L. Chang, A simulated annealing approach to scheduling a manufacturing cell, *Naval Research Logistics* 37 (1990) 559–577.
- [75] S. van Hoesel, A. Wagelmans, B. Moerman, Using geometric techniques to improve dynamic programming algorithms for the economic lot-sizing problem and extensions, *European Journal of Operational Research* 75 (1994) 312–331.
- [76] R.G. Vickson, M.J. Magazine, C.A. Santos, Batching and sequencing of components at a single facility, *IIE Transactions* 25 (1993) 65–70.
- [77] S.T. Webster, The complexity of scheduling job families about a common due date, *Operations Research Letters* 20 (1997) 65–74.
- [78] S.T. Webster, Note on “Parallel machine scheduling with batch setup times”, *Operations Research* 46 (1998) 423.
- [79] S.T. Webster, K.R. Baker, Scheduling groups of jobs on a single machine, *Operations Research* 43 (1995) 692–703.
- [80] D. Williams, A. Wirth, A new heuristic for a single machine scheduling problem with set-up times, *Journal of the Operational Research Society* 47 (1996) 175–180.
- [81] G.J. Woeginger, A polynomial time approximation scheme for single machine sequencing with delivery times and sequence independent batch setup times, Report Woe-17, START Project Y43-MAT, Institut für Mathematik, TU Graz, Austria, 1997.
- [82] X. Yang, Scheduling with generalized batch delivery dates and earliness penalties, *IIE Transactions*, in press.
- [83] S. Zdrzałka, Analysis of approximation algorithms for single-machine scheduling with delivery times and sequence independent batch setup times, *European Journal of Operational Research* 80 (1995) 371–380.